デベロッパーガイド

Amazon Elastic Container Service



Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon Elastic Container Service: デベロッパーガイド

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon の商標およびトレードドレスは Amazon 以外の製品およびサービスに使用することはできません。また、お客様に誤解を与える可能性がある形式で、または Amazon の信用を損なう形式で使用することもできません。Amazon が所有していないその他のすべての商標は Amazon との提携、関連、支援関係の有無にかかわらず、それら該当する所有者の資産です。

Table of Contents

Amazon ECS とは 1
用語とコンポーネント 1
機能 3
プロビジョニング 4
料金 4
関連サービス 4
開始方法 6
セットアップ 6
AWS マネジメントコンソール 7
AWS アカウント へのサインアップ 7
仮想プライベートクラウドを作成する 7
セキュリティグループの作成 8
EC2 インスタンスに接続するための認証情報を作成します。 11
AWS CLI のインストール 12
Amazon ECS を使用するための次のステップ 13
コンテナイメージの作成 13
前提条件 14
Docker イメージの作成 16
Amazon Elastic Container Registry にイメージをプッシュします 18
クリーンアップ 20
次のステップ 20
Amazon ECS マネージドインスタンスのタスクの作成方法について説明します。 20
前提条件 20
ステップ 1: クラスターを作成する 21
ステップ 2: タスク定義を作成する 22
ステップ 3: サービスを作成する 23
ステップ 4: サービスを表示する 24
ステップ 5: クリーンアップ 24
AWS CLI を使用した Amazon ECS マネージドインスタンスのタスクの作成方法について説明します。 25
前提条件 26
ステップ 1: クラスターを作成する 27
ステップ 2: Amazon ECS マネージドインスタンスキャパシティプロバイダーを作成する ... 28
ステップ 3: クラスターのデフォルトのキャパシティプロバイダー戦略を設定する 29

ステップ 4: Linux タスク定義を登録する 29
ステップ 5: タスク定義をリスト表示する 31
ステップ 6: サービスを作成する 31
ステップ 7: サービスをリスト表示する 32
ステップ 8: 実行中のサービスを記述する 32
ステップ 9: テスト 34
ステップ 10: クリーンアップする 36
Fargate 用のタスクを作成する方法について説明します。 37
前提条件 37
ステップ 1: クラスターを作成する 38
ステップ 2: タスク定義を作成する 39
ステップ 3: サービスを作成する 40
ステップ 4: サービスを表示する 40
ステップ 5: クリーンアップ 41
Fargate 用の Windows タスクを作成する方法について説明します。 41
前提条件 42
ステップ 1: クラスターを作成する 42
ステップ 2: Windows タスク定義を登録する 43
ステップ 3: タスク定義を使用してサービスを作成する 44
ステップ 4: サービスを表示する 45
ステップ 5: クリーンアップ 46
EC2 用の Windows タスクを作成する方法について説明します。 46
前提条件 46
ステップ 1: クラスターを作成する 47
ステップ 2: タスク定義を登録する 49
ステップ 3: サービスを作成する 50
ステップ 4: サービスを表示する 51
ステップ 5: クリーンアップ 52
の使用AWS CDK 53
ステップ 1:AWS CDK プロジェクトを設定する 54
ステップ 2: AWS CDK を使用して、Fargate 上のコンテナ化されたウェブサーバーを定義する 56
ステップ 3: ウェブサーバーをテストする 64
ステップ 4: クリーンアップする 64
次のステップ 64
AWS Copilot CLI を使用したリソースの作成 65

AWS Copilot CLI のインストール ... 67
AWS Copilot CLI を使用してサンプル Amazon ECS アプリケーションをデプロイする ... 76
Amazon ECS での CloudFormation ... 79
CloudFormation コンソールを使用した Amazon ECS リソースの作成 ... 80
前提条件 ... 80
ステップ 1: スタックテンプレートを作成する ... 80
ステップ 2: Amazon ECS リソースのスタックを作成する ... 92
ステップ 3: 確認する ... 93
ステップ 4: リソースをクリーンアップする ... 94
AWS CLI コマンドを使用した CloudFormation の Amazon ECS リソースの作成 ... 94
前提条件 ... 94
ステップ 1: スタックを作成する ... 95
ステップ 2: Amazon ECS リソースの作成を検証する ... 107
ステップ 3: クリーンアップする ... 109
CloudFormation テンプレートの例 ... 109
タスク定義 ... 110
キャパシティープロバイダー ... 122
クラスター ... 124
サービス ... 137
Amazon ECS の IAM ロール ... 211
ベストプラクティス ... 215
Amazon ECS のソリューションの構築 ... 219
Capacity ... 220
サービスエンドポイント ... 221
ネットワーク ... 221
機能アクセス ... 222
IAM ロール ... 223
ログ記録 ... 223
コンテナイメージのベストプラクティス ... 223
起動タイプとキャパシティプロバイダー ... 224
ベストプラクティス ... 224
サービスのミュータビリティ ... 225
デュアルスタックのエンドポイントの使用 ... 227
デュアルスタックのエンドポイントの使用 (AWS CLI) ... 228
デュアルスタックのエンドポイントを使用する (AWS SDK) ... 229
REST API からのデュアルスタックのエンドポイントの使用 ... 229

共有サブネット、Local Zones、および Wavelength Zones のアプリケーション 230
共有サブネット 231
ローカルゾーン 232
Wavelength Zone 232
AWS OutpostsのAmazon Elastic Container Service 233
考慮事項 233
前提条件 234
AWS Outposts でのクラスター作成の概要 234
コンテナイメージについてのベストプラクティス 237
ネットワークのベストプラクティス 239
アプリケーションをインターネットに接続する 240
Amazon ECS へのインバウンド接続を受信するためのベストプラクティス 243
AWS サービスに接続するためのベストプラクティス 248
サービスを接続するためのベストプラクティス 251
AWS アカウントと VPC の間のネットワーキングサービスのベストプラクティス 256
AWSネットワーキングのトラブルシューティングのための サービス 257
タスクの起動時間を最適化する 258
大規模な運用 259
Amazon ECS Service Quotas と API スロットリング制限 260
スロットリングに関する問題を処理する 265
自動スケーリングとキャパシティ管理におけるベストプラクティス 268
タスクサイズの決定 269
サービス自動スケーリングの最適化 270
キャパシティとアベイラビリティ 276
クラスターのキャパシティ 281
Fargate タスクサイズの選択 281
クラスターの自動スケーリングの高速化 282
アカウント設定を使用した機能へのアクセス 284
Amazon リソースネーム (ARN) と ID 285
ARN およびリソース ID 形式のタイムライン 287
Container Insights 287
AWS Fargate 連邦情報処理標準 (FIPS-140) コンプライアンス 289
タグ付け認可 291
タグ付け認可のタイムライン 292
AWS Fargate タスク廃止の待機時間 292
AWS FargateEC2 イベントウィンドウを使用した タスクの廃止 294

Linux コンテナインスタンスのネットワークインターフェイスを増やす 294
Runtime Monitoring (Amazon GuardDuty 統合) 295
デュアルスタック IPv6 VPC 296
デフォルトのログドライバーモード 296
コンソールを使用したアカウント設定の表示 297
アカウント設定の変更 298
デフォルトのアカウント設定の復元 299
AWS CLI を使用したアカウント設定の管理 299
Amazon ECS の IAM ロール 301
Amazon ECS マネージドインスタンス 307
開始方法 307
キャパシティープロバイダー 308
インスタンスの選択と最適化 308
タスク定義 309
オペレーティングシステムと CPU アーキテクチャ 310
主な特徴 310
IAM ロール 310
セキュリティとコンプライアンス 311
ネットワーク 311
インスタンスストレージ 312
ローカルストレージ 313
サービスの負荷分散 316
モニタリングとオブザーバビリティ 316
料金とコストの最適化 316
Service Quotas 317
移行に関する考慮事項 317
制約事項と考慮事項 317
組織コントロール 318
Amazon ECS マネージドインスタンス AMI 以外のすべての AMI をブロックする SCP の例 318
Amazon ECS マネージドインスタンス AMI アカウント 319
Amazon ECS マネージドインスタンスのインスタンスタイプ 321
インスタンスの選択方法 321
特定のインスタンスタイプの選択 322
属性ベースのインスタンスタイプの選択 322
請求と購入のオプション 323

Amazon ECS マネージド インスタンスでサポートされているインスタンスタイプ 323
Amazon ECS マネージドインスタンスにおけるインスタンス選択のベストプラクティス 324
Amazon ECS マネージドインスタンスの取り込み動作 326
コンテナイメージの解決 327
Amazon ECS マネージドインスタンスでのパッチ適用 327
インスタンスのライフサイクル 327
イベントウィンドウとメンテナンススケジューリング 327
メンテナンスシーケンス 328
Amazon ECS マネージドインスタンスのセキュリティ上の考慮事項 329
セキュリティモデル 329
マネージドインスタンスの概要 329
セキュリティ機能 330
コンプライアンスと規制への対応 331
Amazon ECS マネージドインスタンスにおける FIPS-140 関連の考慮事項 332
Amazon ECS マネージドインスタンスで FIPS を無効にする 332
セキュリティに関する考慮事項 333
セキュリティのベストプラクティス 335
Amazon ECS マネージドインスタンスの VPC 暗号化コントロールの有効化 335
前提条件 335
互換性のあるインスタンスタイプを特定する 336
VPC 暗号化をサポートするクラスターを作成する 336
考慮事項 337
トラブルシューティング 337
インフラストラクチャの最適化 337
非稼働インスタンスの検出 338
使用率の低いインスタンス検出 338
ScaleInAfter 339
GPU 自動修復 340
仕組み 340
GPU ヘルスのモニタリング 341
モニタリングされた XID エラーコード 341
自動修復の無効化 342
Fargate から Amazon ECS マネージドインスタンスへの移行 344
移行に関する考慮事項 344
前提条件 344

ステップ 1: Amazon ECS マネージドインスタンスを使用するようにクラスターを更新する 345
ステップ 2: Amazon ECS マネージドインスタンス機能に合わせてタスク定義を更新する . 345
ステップ 3: Amazon ECS マネージドインスタンスのキャパシティプロバイダを使用するようにサービスを更新する 346
ステップ 4: 独立型タスクを移行する 346
EC2 から Amazon ECS マネージドインスタンスへの移行 346
移行に関する考慮事項 346
前提条件 347
ステップ 1: Amazon ECS マネージドインスタンスを使用するようにクラスターを更新する 347
ステップ 2: Amazon ECS マネージドインスタンス機能に合わせてタスク定義を更新する . 348
ステップ 3: Amazon ECS マネージドインスタンスのキャパシティプロバイダを使用するようにサービスを更新する 348
ステップ 4: 独立型タスクを移行する 348
Amazon ECS マネージドデーモン 349
デーモンの重要度 349
マネージドデーモンの仕組み 349
主な利点 350
開始方法 351
デーモンタスク定義 351
デーモンタスク定義の作成 351
デーモンタスク定義パラメータ 353
ネットワークモード 354
特権機能 354
PID と IPC 名前空間の共有 355
ボリューム 356
サポートされているコンテナパラメータ 357
デーモンの作成と管理 358
前提条件 358
デーモンの作成 359
デーモンのデプロイの検証 361
デーモンの更新 361
デーモンの削除 363
デーモンデプロイ 363
デーモンデプロイの仕組み 363

デプロイ設定パラメータ ... 364
デプロイのモニタリング ... 365
デプロイ例のシナリオ ... 365
デーモンイベント ... 369
デーモンサービスアクションイベント ... 369
デーモンデプロイ状態変更イベント ... 371
デーモンイベントのルールの作成 ... 373
デーモンメトリクス ... 374
デーモン使用率メトリクス ... 374
デーモンメトリクスの表示 ... 375
デーモン自動修復 ... 376
AWS Fargate ... 379
チュートリアル ... 379
キャパシティープロバイダー ... 380
タスク定義 ... 380
プラットフォームのバージョン ... 380
サービスの負荷分散 ... 381
使用状況メトリクス ... 381
Fargate をどのような場合に使用するかについてのセキュリティ上の考慮事項 ... 382
Fargate セキュリティのベストプラクティス ... 382
AWS KMS を使用して Fargate のエフェメラルストレージを暗号化する ... 382
Fargate を使用したカーネルシステムコールトレーシングの SYS_PTRACE 機能 ... 383
Fargate Runtime Monitoring で Amazon GuardDuty を使用する ... 383
Fargate のセキュリティに関する考慮事項 ... 383
Fargateプラットフォームバージョン ... 384
Linux プラットフォームバージョン 1.4.0 への移行 ... 386
Fargate Linux プラットフォームバージョンの変更ログ ... 387
Linux プラットフォームバージョンの廃止 ... 389
Windows プラットフォームバージョンの変更ログ ... 392
Amazon ECS における Fargate 上の Windows コンテナに関する考慮事項 ... 393
Fargate 上の Linux コンテナにおけるコンテナイメージのプル動作 ... 394
Fargate 上の Windows コンテナにおけるコンテナイメージのプル動作 ... 396
Fargate タスクエフェメラルストレージ ... 396
Fargate Linux コンテナプラットフォームのバージョン ... 397
Fargate Windows コンテナプラットフォームのバージョン ... 398
AWS Fargate エフェメラルストレージ用のカスタマーマネージドキー ... 398

タスクの廃止とメンテナンス ... 411
タスク廃止通知の概要 ... 413
タスク廃止通知の影響を受けるリソースのステータス ... 416
タスク廃止をオプトアウトすることはできますか? ... 417
他の AWS サービスを通じてタスク終了通知を受け取ることはできますか? ... 417
タスクの廃止はスケジュールされた後に変更できますか? ... 417
Amazon ECS はサービスの一部であるタスクをどのように処理しますか? ... 418
Amazon ECS はスタンドアロンタスクを自動的に処理できますか? ... 418
タスク廃止中のサービス可用性のトラブルシューティング ... 418
Amazon ECS での AWS Fargate タスク廃止に備える ... 419
AWS Fargate リージョン ... 424
AWS Fargate 上の Linux コンテナ ... 424
AWS Fargate 上の Windows コンテナ ... 427
EC2 ... 429
EC2 と外部インスタンスでのコンテナイメージのプル動作 ... 430
Amazon ECS Anywhere ... 432
Amazon ECS Express Mode ... 433
Amazon ECS Express Mode サービスの利点 ... 433
料金 ... 434
可用性 ... 434
Express Mode サービスの作成 ... 435
前提条件 ... 435
チュートリアル ... 435
カスタマイズ ... 436
コンソールで最初の Express Mode サービスを作成する ... 436
手順 ... 436
次のステップ ... 438
カスタマイズ ... 438
AWS CLI を使用して最初の Express Mode サービスを作成する ... 440
前提条件 ... 440
ステップ 1: IAM ロールを作成する ... 440
ステップ 2: 最初の Express Mode サービスアプリケーションを作成する ... 442
ステップ 3: カスタム設定で Express Mode サービスアプリケーションを作成する ... 436
ステップ 4: デプロイをモニタリングする ... 444
ステップ 5: アプリケーションにアクセスする ... 445
Express Mode サービスによって作成されたリソース ... 445

クラスターのデフォルト ... 446
タスク定義のデフォルト ... 446
コンテナ定義のデフォルト ... 447
ログ記録のデフォルト ... 448
サービスのデフォルト ... 448
ネットワーク設定のデフォルト ... 450
IAM ロールのデフォルト ... 451
Application Auto Scaling のデフォルト ... 452
アプリケーションスケーリングターゲットのデフォルト ... 452
アプリケーションスケーリングポリシーのデフォルト ... 453
Application Load Balancer のデフォルト ... 454
ターゲットグループのデフォルト ... 454
リソース共有とコストの最適化 ... 455
Express Mode サービスの詳細の表示 ... 456
Amazon ECS Express Mode サービスの更新 ... 457
デプロイの動作 ... 457
手順 ... 457
Express Mode サービスを削除する ... 460
Express Mode サービスの削除とサービスの削除の違い ... 461
手順 ... 462
Amazon ECS Express Mode サービスのベストプラクティス ... 462
セキュリティのベストプラクティス ... 462
パフォーマンスとコンピューティングの最適化 ... 464
運用のベストプラクティス ... 466
トラブルシューティング ... 467
デプロイに関する問題 ... 467
カスタムタスク定義エラー ... 469
接続の問題 ... 470
パフォーマンスの問題 ... 470
モニタリングとデバッグツール ... 471
Express Mode 以外のリソースの更新 ... 472
責任共有モデルを理解する ... 472
Express Mode サービスで Amazon ECS 機能を利用する前に、何かする必要はありますか? ... 473
Express Mode 外での更新の例 ... 473
タスク定義 ... 476

タスク定義の状態 477
削除をブロックできる Amazon ECS リソース 478
アプリケーションの構築 479
タスクサイズのベストプラクティス 480
Amazon ECS マネージドインスタンスのタスクネットワーキング 481
EC2 のタスクネットワーク 487
Fargate 用タスクネットワーキング 503
タスクのストレージオプション 511
コンテナスワップメモリ空間の管理 615
Amazon ECS マネージドインスタンスにおけるタスク定義の違い 617
Fargate のタスク定義の違い 619
Windows を実行している EC2 インスタンスでのタスク定義の違い 628
コンソールを使用したタスク定義の作成 629
JSON 検証 630
CloudFormation スタック 630
手順 630
Amazon Q Developer を使用して Amazon ECS コンソールでタスク定義のレコメンデーションを生成する 661
前提条件 661
手順 662
コンソールを使用したタスク定義の更新 664
JSON 検証 665
手順 665
新しいコンソールを使用したタスク定義リビジョンの登録解除 666
CloudFormation スタック 630
手順 667
コンソールを使用したタスク定義リビジョンの削除 667
削除をブロックできる Amazon ECS リソース 478
手順 668
タスク定義のユースケース 669
GPU ワークロード向けのタスク定義 669
動画トランスコーディングワークロードでのタスク定義 685
AWS Neuron 機械学習ワークロードでのタスク定義 699
深層学習インスタンスでのタスク定義 710
64 ビット ARM ワークロードでのタスク定義 712
CloudWatch にログを送信する 715

AWS サービスまたは AWS Partner にログを送信する 718
AWS 以外のコンテナイメージの使用 746
タスク内の個々のコンテナを再起動する 749
コンテナに機密データを渡す 752
Amazon ECS マネージドインスタンスのタスク定義パラメータ 775
ファミリー 776
Capacity 776
タスクロール 776
タスク実行ロール 777
ネットワークモード 777
ランタイムプラットフォーム 778
タスクサイズ 779
その他のタスク定義パラメータ 780
コンテナ定義 787
Fargate のタスク定義パラメータ 831
ファミリー 832
Capacity 832
タスクロール 832
タスク実行ロール 833
ネットワークモード 833
ランタイムプラットフォーム 833
タスクサイズ 834
コンテナ定義 838
Elastic Inference アクセラレーター名 (廃止) 884
プロキシ設定 885
ボリューム 887
タグ 894
その他のタスク定義パラメータ 895
EC2 のタスク定義パラメータ 897
ファミリー 898
Capacity 898
タスクロール 898
タスク実行ロール 899
ネットワークモード 899
ランタイムプラットフォーム 901
タスクサイズ 902

コンテナ定義 903
Elastic Inference アクセラレーター名 (廃止) 953
タスク配置の制約事項 953
プロキシ設定 954
ボリューム 956
タグ 963
その他のタスク定義パラメータ 964
タスク定義テンプレート 966
タスク定義の例 977
Web サーバー 978
splunk ログドライバー 980
fluentd ログドライバー 981
gelf ログドライバー 982
外部インスタンス上のワークロード 982
Amazon ECR イメージとタスク定義 IAM ロール 984
コマンドによるエントリポイント 984
コンテナの依存関係 985
タスク定義のボリューム 987
Windows のサンプルのタスク定義 987
クラスター 989
適切なクラスタータイプの選択 989
クラスターコンポーネント 990
クラスターの概念 991
キャパシティープロバイダー 992
Amazon ECS マネージドインスタンスのクラスター 993
Amazon ECS マネージドインスタンスのクラスターを作成する 995
Amazon ECS マネージドインスタンスを使用するようにクラスターを更新する 999
マネージドインスタンスのキャパシティプロバイダー 1002
タスクスケールイン保護 1011
Fargate のクラスター 1019
Fargate Spot 終了通知 1020
Fargate ワークロード用のクラスターを作成する 1022
EC2 ワークロード用の Amazon ECS キャパシティプロバイダー 1025
EC2 コンテナインスタンスのセキュリティ 1027
Amazon EC2 ワークロード用のクラスターを作成する 1028
クラスターの自動スケーリング 1034

Amazon EC2 コンテナインスタンス 1071
外部インスタンスのクラスター 1249
サポートされるオペレーティングシステムとシステムアーキテクチャ 1249
考慮事項 1250
外部インスタンスワークロード用のクラスターを作成する 1254
Amazon ECS クラスターに外部インスタンスを登録する 1256
外部インスタンスの登録を解除する 1262
AWS Systems Manager エージェントと Amazon ECS コンテナエージェントを更新する 1268
クラスターの更新 1273
クラスターの削除 1275
コンテナインスタンスの登録解除 1276
手順 1277
コンテナインスタンスのドレイン 1277
サービスのドレイニング動作 1278
スタンドアロンタスクのドレイニング動作 1279
Amazon ECS マネージドインスタンスのドレイニング動作 1064
手順 1280
Auto Scaling グループ外のインスタンスのタイプまたはサイズを変更する 1281
Amazon EC2 コンテナインスタンス 1282
ライフサイクル 1282
Docker サポート 1283
Amazon ECS に最適化された AMI 1284
追加情報 1284
コンテナエージェントの設定 1284
Amazon ECS コンテナエージェントをインストールする 1287
コンテナエージェントのログ設定パラメータ 1293
プライベート Docker イメージ用のコンテナインスタンスを設定する 1295
タスクとイメージのクリーンアップ 1299
コンテナをスケジュールする 1302
コンピューティングオプション 1304
タスクのライフサイクル 1305
ライフサイクル状態 1306
Amazon ECS がタスクをコンテナインスタンスに配置する方法 1308
Amazon ECS マネージドインスタンス 1308
EC2 1309
Fargate 1310

Amazon ECS マネージドインスタンスの自動スケーリングとタスク配置 1310
戦略を使用してタスク配置を定義する 1314
グループに関連するタスク 1319
どのコンテナインスタンスをタスクに使用するかを定義します。 1320
スタンドアロンのタスク 1331
タスクワークフロー 1331
タスクとしてのアプリケーションの実行 1332
Amazon EventBridge スケジューラを使用してタスクをスケジュールする 1344
タスクの停止 1351
サービス 1352
インフラストラクチャコンピューティングオプション 1353
サービスのオートスケーリング 1354
サービスの負荷分散 1354
相互接続サービス 1356
サービスデプロイコントローラーと戦略 1357
サービスのデプロイ 1590
サービスリビジョン 1600
アベイラビリティーゾーンの再調整 1609
ロードバランサーを使用してサービストラフィックを分散する 1618
サービスのオートスケーリング 1633
サービスを相互接続する 1679
タスクスケールイン保護 1757
Amazon ECS と Fargate によるフォールトインジェクション 1767
Amazon ECS サービスのパラメータの更新 1776
サービスの削除 1806
サービスの短い ARN を移行する 1807
サービスの調整ロジック 1813
サービス定義パラメータ 1814
サービス定義テンプレート 1845
リソースのタグ付け 1853
リソースのタグ付け方法 1854
作成時のリソースのタグ付け 1857
制限事項 1858
Amazon ECS マネージドのタグ 1858
請求にタグを使用する 1859
リソースに タグを追加する 1860

Amazon ECS マネージドインスタンスのタグ 1863
AWS によって追加されたタグ 1863
カスタムタグ 1864
EC2 コンテナインスタンスにタグを追加する 1865
外部コンテナインスタンスにタグを追加する 1867
使用状況レポート 1867
タスクレベルのコストと使用状況 1868
モニタリング 1870
Amazon ECS モニタリングのベストプラクティス 1871
モニタリングツール 1871
自動化ツール 1871
手動ツール 1873
CloudWatch を使用して Amazon ECS をモニタリングする 1874
考慮事項 1875
推奨メトリクス 1875
Amazon ECS メトリクスの表示 1876
Amazon ECS CloudWatch メトリクス 1877
AWS Fargate 使用状況メトリクス 1910
Amazon ECS クラスターの予約率メトリクス 1912
Amazon ECS クラスターの使用率メトリクス 1914
Amazon ECS サービスの使用率メトリクス 1916
EventBridge を使用して Amazon ECS エラーへの対応を自動化する 1920
Amazon ECS イベント 1921
イベントの処理 1951
オブザーバビリティが強化された Container Insights を使用し、Amazon ECS コンテナを監視する 1955
Amazon ECS マネージドインスタンスの追加メトリクス 1956
Amazon ECS マネージドインスタンスのモニタリング 1961
Container Insights のモニタリング 1961
GPU モニタリング 1962
インスタンスのモニタリング 1962
コンテナのヘルスチェックを使用してタスク状態を判定する 1964
タスクの正常性を判別する方法 1966
ヘルスチェックとエージェントの切断 1967
コンテナの正常性を表示する 1967
Amazon ECS コンテナインスタンスの正常性をモニタリングする 1967

コンテナインスタンスの異常 1969
アプリケーショントレースデータを使用して Amazon ECS 最適化の機会を特定する 1969
AWS X-Ray を使用した AWS Distro for OpenTelemetry の統合に必要な IAM 権限 1970
タスク定義で AWS X-Ray 統合向け OpenTelemetry サイドカー用 AWS Distro の指定 1971
アプリケーションメトリクスを使用して Amazon ECS アプリケーションのパフォーマンスを相関させる 1972
アプリケーションメトリクスを Amazon CloudWatch にエクスポートする 1973
アプリケーションメトリクスを Amazon Managed Service for Prometheus にエクスポートする 1977
AWS CloudTrail を使用して Amazon ECS API コールをログに記録する 1980
CloudTrail の Amazon ECS 管理イベント 1982
CloudTrail の Amazon ECS データイベント 1982
Amazon ECS イベントの例 1983
Amazon ECS サービスとタスクに関する CloudWatch Logs Live Tail の表示 1985
必要なアクセス許可 1986
手順 1986
AWS CLI のリファレンス 1987
アクションログを使用して Amazon ECS オペレーションをモニタリングする 1987
Amazon ECS アクションログの概要 1987
アクションログの利点 1988
ログで記録される内容 1988
アクションログの仕組み 1988
ログスキーマの概要 1989
ログの送信先と整理 1989
料金 1990
Amazon ECS アクションログの導入 1990
Amazon ECS アクションログを使用したトラブルシューティング 1994
メタデータを使用したワークロードのモニタリング 1998
環境変数 1999
コンテナメタデータファイル 2000
Amazon ECS マネージドインスタンスのタスクで使用できるタスクメタデータ 2007
EC2 のタスクで使用できる Amazon ECS タスクメタデータ 2019
Fargate のタスクで使用できるタスクメタデータ 2063
コンテナの詳細分析 2086
ランタイムモニタリングを使用して不正な動作を特定する 2089
Amazon ECS でのランタイムモニタリングの仕組み 2090

考慮事項 2091
リソース使用率 2092
Fargate ワークロードのランタイムモニタリング 2092
EC2 ワークロードのランタイムモニタリング 2096
ランタイムモニタリングのトラブルシューティング 2101
ECS Exec を使用して Amazon ECS コンテナをモニタリングする 2105
考慮事項 2105
アーキテクチャ 2108
ECS Exec の設定 2109
ECS Exec を使用したログ記録 2111
IAM ポリシーを使用して ECS Exec へのアクセスを制限する 2115
ECS Exec のトラブルシューティング 2119
ECS Exec を使用してコマンド実行 2119
Compute Optimizer 推奨事項 2122
Fargate のタスクサイズに関する推奨事項 2122
Amazon ECS MCP サーバー 2123
概要 2123
開始方法 2124
開始方法 2124
前提条件 2124
セットアップ 2125
一般的な設定とベストプラクティス 2131
ツール設定 2132
ツール設定 2132
環境変数 2133
引数 2133
ツール 2134
トラブルシューティング 2141
停止したタスクのエラーを解決する 2145
停止したタスクのエラーメッセージの更新 2146
停止したタスクのエラーを表示する 2150
停止したタスクのエラーメッセージ 2152
タスクの接続を検証する 2172
IAM ロールリクエストの表示 2177
サービスイベントメッセージを表示する 2178
Amazon ECS のサービスイベントメッセージ 2179

Amazon ECS サービス異常イベントメッセージ 2192
Amazon ECS アベイラビリティーゾーンサービスの再調整サービスイベントメッセージ . 2193
Amazon ECS のサービスロードバランサーのトラブルシューティング 2195
Amazon ECS のサービス自動スケーリングのトラブルシューティング 2197
イベントキャプチャ 2198
仕組み 2198
イベントタイプ 2199
ロググループ設定 2199
考慮事項 2199
イベントベースのトラブルシューティング 2200
既存のクラスターのイベントキャプチャを有効にする 2201
サービスおよびタスクの状態変更イベントを表示する 2202
タスク定義の無効な CPU またはメモリエラーのトラブルシューティングする 2203
コンテナエージェントログの表示 2206
Amazon ECS ログコレクターを使用したコンテナログの収集 2208
エージェントのイントロスペクション 2210
Amazon ECS の Docker 診断 2212
Amazon ECS の Docker コンテナを一覧表示する 2213
Amazon ECS で Docker ログを表示する 2214
Amazon ECS で Docker コンテナを検査する 2215
Amazon ECS の Docker デーモンからの詳細な出力の設定 2216
Amazon ECS の Docker API error (500): devmapper のトラブルシューティング 2217
ECS Exec に関する問題のトラブルシューティング 2219
Execチェッカーを使用して検証する 2219
execute-command 呼び出し時のエラー 2219
Amazon ECS Anywhere に関する問題のトラブルシューティング 2219
外部インスタンス登録の問題 2220
外部インスタンスネットワークの問題 2221
実行中のタスクに関する問題 2221
Fargate での Java クラス読み込みに関する問題のトラブルシューティング 2222
症状 2222
原因 2222
解決策 2222
防止 2224
AWS Fargate スロットリングのクォータ 2224
Fargate での RunTask API のスロットリング 2225

Fargate でのレートクォータの調整 2225
Amazon ECS マネージドインスタンスのトラブルシューティング 2226
前提条件 2226
一般的なトラブルシューティングのシナリオ 2226
Amazon ECS マネージドインスタンスのログ分析 2233
EC2 AWS CLI を使用して Amazon ECS マネージドインスタンスからコンソール出力を取得する 2233
クリーンアップ 2234
その他のリソース 2152
Amazon ECS マネージドインスタンスのトラブルシューティング 2234
Amazon ECS マネージドインスタンスエラー 2239
API の失敗の理由 2243
Amazon Q Developer のトラブルシューティング 2253
必要なアクセス許可 2253
手順 2253
考慮事項 2254
詳細情報 2255
セキュリティ 2256
Identity and Access Management 2257
対象者 2257
アイデンティティによる認証 2258
ポリシーを使用したアクセス権の管理 2259
IAM を使用するAmazon Elastic Container Service 2261
アイデンティティベースのポリシーの例 2272
AWSAmazon ECS の 管理ポリシー 2284
サービスにリンクされたロールの使用 2308
Amazon ECS の IAM ロール 2315
Amazon ECS コンソールに必要なアクセス許可 2386
Amazon ECS ブルー/グリーンデプロイの Lambda 関数に必要となるアクセス許可 2397
Amazon ECS のサービス自動スケーリングに必要な IAM アクセス許可 2398
リソース作成時のタグ付け 2399
トラブルシューティング 2403
IAM ベストプラクティス 2406
ログ記録とモニタリング 2409
コンプライアンス検証 2411
コンプライアンスとセキュリティのベストプラクティス 2411

AWS Fargate FIPS-140 コンプライアンス 2413
AWS Fargate FIPS-140 に関する考慮事項 2413
Fargate で FIPS を使用する 2414
Fargate FIPS-140 監査に CloudTrail を使用する 2414
インフラストラクチャセキュリティ 2416
インターフェイス VPC エンドポイント (AWS PrivateLink) 2416
責任共有モデル 2424
Fargate 2424
EC2 2425
Amazon ECS マネージドインスタンスの責任共有 2426
AWS の責任 2426
お客様の責任 2427
ネットワークセキュリティのベストプラクティス 2428
転送中の暗号化 2428
タスクネットワーク 2429
AWS PrivateLink および Amazon ECS 2430
コンテナエージェントの設定 2431
ネットワークセキュリティに関する推奨事項 2431
タスクとコンテナのセキュリティのベストプラクティス 2433
最小限のイメージを作成するか、distroless のイメージを使用する 2433
イメージをスキャンして脆弱性がないか調べる 2435
イメージから特別な権限を削除する 2436
キュレーションされたイメージのセットを作成する 2436
アプリケーションパッケージとライブラリをスキャンして脆弱性がないか調べる 2435
静的コード分析を行う 2437
非ルートユーザーとしてコンテナを実行する 2437
読み取り専用のルートファイルシステムを使用する 2438
CPU とメモリの制限を設定してタスクを設定する (Amazon EC2) 2438
Amazon ECR でイミュータブルタグを使用する 2439
コンテナを特権として実行することは避けてください (Amazon EC2) 2439
不要な Linux 機能をコンテナから削除する 2439
カスタマーマネージドキー (CMK) を使用して Amazon ECR にプッシュされるイメージを暗号化する 2440
チュートリアル 2441
AWS CLI を使用して、Fargate 用の Linux タスクを作成する 2443
前提条件 2444

ステップ 1: クラスターを作成する 2445
ステップ 2: Linux タスク定義を登録 2446
ステップ 3: タスク定義をリスト表示する 2447
ステップ 4: サービスを作成する 2448
ステップ 5: サービスをリスト表示する 2448
ステップ 6: 実行中のサービスを記述する 2449
ステップ 7: テスト 2451
ステップ 8: クリーンアップ 2455
AWS CLI を使用して、Fargate 用の Windows タスクを作成する 2455
前提条件 2456
ステップ 1: クラスターを作成する 2457
ステップ 2: Windows タスク定義を登録する 2457
ステップ 3: タスク定義をリスト表示する 2459
ステップ 4: サービスを作成する 2459
ステップ 5: サービスをリスト表示する 2460
ステップ 6: 実行中のサービスを記述する 2460
ステップ 7: クリーンアップする 2462
AWS CLI を使用して、EC2 用のタスクを作成する 2463
前提条件 2464
クラスターを作成する 2464
Amazon ECS AMI 向けコンテナインスタンスの起動 2465
コンテナインスタンスを一覧表示する 2467
コンテナインスタンスを説明する 2468
タスク定義を登録する 2471
タスク定義をリスト表示する 2473
サービスを作成する 2473
リストサービス 2474
サービスを記述する 2474
実行中のタスクを記述する 2475
ウェブサーバーをテストする。 2476
リソースをクリーンアップする 2477
CloudWatch Events イベントをリッスンするように Amazon ECS を設定する 2479
前提条件: テストクラスターを設定する 2479
ステップ 1: Lambda 関数を作成する 2479
ステップ 2: イベントルールを登録する 2480
ステップ 3: タスク定義を作成する 2481

ステップ 4: ルールをテストする 2482
タスク停止イベントに Amazon Simple Notification Service アラートを送信 2483
前提条件: テストクラスターを設定する 2483
前提条件: Amazon SNS 用のアクセス許可の設定 2483
ステップ 1: Amazon SNS トピックを作成してサブスクライブする 2484
ステップ 2: イベントルールを登録する 2484
ステップ 3: ルールをテストする 2485
複数行またはスタックトレースのログメッセージの連結 2487
必要な IAM 許可 2487
複数行ログの設定を使用する場合を決定する 2488
オプションを解析して連結する 2489
Windows コンテナに Fluent Bit をデプロイする 2509
前提条件 2511
ステップ 1: IAM アクセスロールを作成する 2512
ステップ 2: Amazon ECS Windows コンテナインスタンスを作成する 2513
ステップ 3: Fluent Bit の設定 2514
ステップ 4: ログを CloudWatch にルーティングする Windows Fluent Bit タスク定義を登録する 2516
ステップ 5: デーモンスケジューリング戦略を使用して ecs-windows-fluent-bit タスク定義を Amazon ECS サービスとして実行する 2518
ステップ 6: ログを生成する Windows タスク定義を登録する 2519
ステップ 7: windows-app-task タスク定義を実行する 2520
ステップ 8: CloudWatch でログを確認する 2521
ステップ 9: クリーンアップする。 2522
EC2 Linux コンテナ用の gMSA を使用する 2523
考慮事項 2523
前提条件 2525
セットアップ 2526
CredSpec file 2533
Fargate で Linux コンテナに gMSA を使用する 2534
考慮事項 2534
前提条件 2535
セットアップ 2535
CredSpec file 2538
AWS CLI を使用するドメインレス gMSA で Windows コンテナを使用する 2540
前提条件 2540

ステップ 1: Active Directory ドメイン サービス (AD DS) で gMSA アカウントを作成して設定する 2542
ステップ 2: Secrets Manager に認証情報をアップロードする 2544
ステップ 3: CredSpec JSON を変更してドメインレス gMSA 情報を含める 2545
ステップ 4： Amazon S3 に CredSpec をアップロードする 2546
ステップ 5: (オプション) Amazon ECS クラスターを作成する 2547
ステップ 6: コンテナインスタンスの IAM ロールを作成する 2548
ステップ 7: カスタムのタスク実行ロールを作成する 2548
ステップ 8: Amazon ECS Exec のタスクロールを作成する 2549
ステップ 9: タスク定義を登録する 2551
ステップ 10: Windows コンテナインスタンスを登録する 2552
ステップ 11: コンテナインスタンスを検証する 2553
ステップ 12: Windows タスクを実行する 2554
ステップ 13: コンテナに gMSA 認証情報があることを検証する 2555
ステップ 14: クリーンアップする 2556
デバッグ 2557
EC2 Windows コンテナで gMSAs を使用する方法について説明します。 2558
考慮事項 2559
前提条件 2560
セットアップ 2561
Image Builder を使用して、Amazon ECS に最適化されたカスタム AMI を構築する 2566
Amazon ECS に最適化された AMI のクリーンアップ 2568
イメージ ARN と Infrastructure as Code (IaC) の使用 2569
CloudFormation でのイメージ ARN の使用 2571
Terraform でのイメージ ARN の使用 2572
AWS 深層学習コンテナを使用する 2573
チュートリアル: Blue/Green デプロイを使用するサービスの作成 2573
前提条件 2573
ステップ 1: Application Load Balancer の作成 2574
ステップ 2: Amazon ECS クラスターを作成する 2575
ステップ 3: タスク定義を登録する 2576
ステップ 4: Amazon ECS サービスを作成する 2577
ステップ 5: AWS CodeDeploy リソースを作成する 2578
ステップ 6: CodeDeploy デプロイを作成してする 2581
ステップ 7: クリーンアップする 2584
Service Quotas 2587

AWS マネジメントコンソール でサービスクォータを管理する 2587
Service Quotas と API スロットリング制限を処理する 2589
Elastic Load Balancing 261
Elastic Network Interface 262
AWS Cloud Map 264
Amazon ECS の API の使用 2595
Amazon ECS リソースによるアクション 2595
アカウント設定のアクション 2596
クラスターのアクション 2596
クラスターキャパシティプロバイダーのアクション 2596
コンテナのアクション 2597
コンテナエージェントのアクション 2597
コンテナインスタンスのアクション 2597
名前空間のアクション 2597
サービスアクション 2598
サービスデプロイのアクション 2598
サービスリビジョンのアクション 2598
タスク定義のアクション 2598
タスクのアクション 2599
タスクセットのアクション 2599
タグのアクション 2599
API リクエストのスロットリング 2600
スロットリングの適用方法 2600
スロットリングのクォータ 2601
API スロットリングのクォータの調整 2605
API スロットリングの処理 2605
べき等性の確保 2605
Amazon ECS のべき等性 2605
RunTask のべき等性 2606
例 2608
べき等リクエストの再試行の推奨事項 2611
ドキュメント履歴 2613

Amazon Elastic Container Service とは

> **Tip**
>
> 今後開催予定のコンテナワークショップシリーズに参加して、Amazon ECS と AWS Fargate のベストプラクティスを学びましょう。サインアップするにはこちらをクリックしてください。

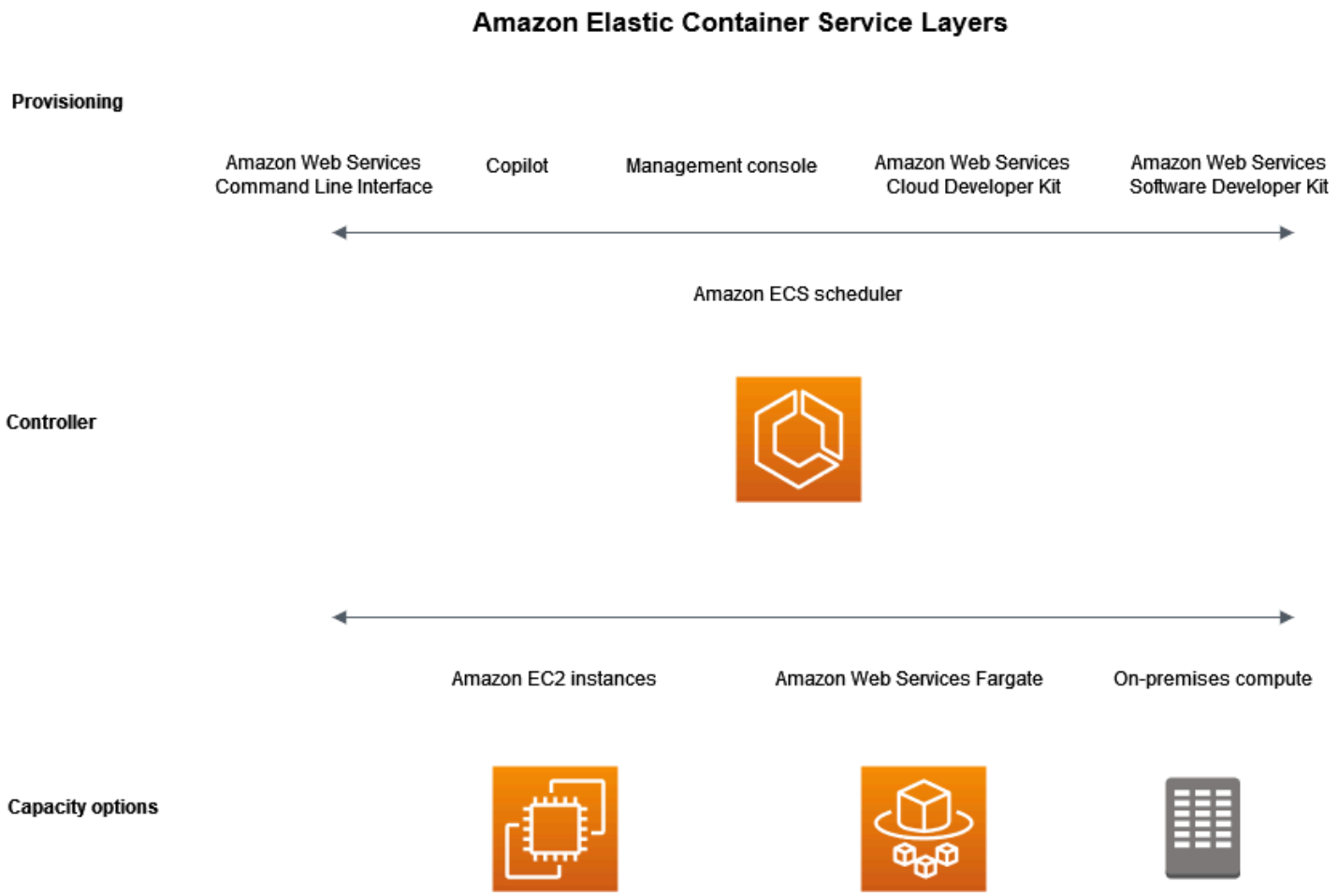
Amazon Elastic Container Service (Amazon ECS) は、コンテナ化されたアプリケーションを簡単にデプロイ、管理、スケーリングできる、完全マネージド型のコンテナオーケストレーションサービスです。フルマネージドサービスである Amazon ECS には、AWS の設定と運用に関するベストプラクティスが組み込まれています。Amazon Elastic Container Registry などの AWS ツールと Docker などのサードパーティーツールの両方に統合されています。この統合により、チームは環境ではなくアプリケーションの構築に集中しやすくなります。コントロールプレーンの複雑な管理は必要なく、クラウドの AWS リージョン 間またはオンプレミスで、コンテナワークロードを実行およびスケーリングできます。

用語とコンポーネント

Amazon ECS には次の 3 つのレイヤーがあります。

- キャパシティ - コンテナが稼働するインフラストラクチャ
- コントローラー - コンテナ上で実行されるアプリケーションをデプロイして管理します
- プロビジョニング - スケジューラーと連携してアプリケーションやコンテナをデプロイおよび管理するために使用できるツール

Amazon のレイヤーは以下の図のようになります。



容量は、コンテナが実行されるインフラストラクチャです。以下は、キャパシティオプションの概要です。

- Amazon ECS マネージドインスタンスは、Amazon ECS のコンピューティングオプションであり、インフラストラクチャ管理を AWS にオフロードしながら、コンテナ化されたワークロードをさまざまな Amazon EC2 インスタンスタイプで実行できます。Amazon ECS マネージドインスタンスでは、AWS が基盤となるインフラストラクチャの割り当て、パッチ適用、スケーリング、メンテナンスを処理している間に、GPU アクセラレーション、特定の CPU アーキテクチャ、高ネットワークパフォーマンス、および特殊なインスタンスタイプなどの特定のコンピューティング機能にアクセスできます。
- クラウド内 AWS の Amazon EC2 instances

  インスタンスタイプ、インスタンス数を選択し、キャパシティを管理します。
- AWS クラウドでのサーバーレス

Fargate はサーバーレスの従量制料金計算エンジンです。Fargate を使用すると、サーバーを管理したり、キャパシティ計画に対処したり、セキュリティのためにコンテナワークロードを分離したりする必要がなくなります。

- オンプレミス仮想マシン (VM) またはサーバー

  Amazon ECS Anywhere は、オンプレミスサーバーや仮想マシン (VM) などの外部インスタンスを Amazon ECS クラスターに登録するためのサポートを提供します。

Amazon ECS スケジューラは、アプリケーションを管理するソフトウェアです。

機能

Amazon ECS には次の高度な機能があります。

タスク定義

アプリケーションのブループリント。

クラスター

アプリケーションが実行されるインフラストラクチャ。

タスク

作業を実行してから停止するバッチジョブなどのアプリケーション。

サービス

長時間実行されるステートレスアプリケーション。

アカウント設定

機能へのアクセスを許可します。

クラスターの自動スケーリング

Amazon ECS では、クラスターに登録された Amazon EC2 インスタンスのスケーリングを管理します。

サービスのオートスケーリング

Amazon ECS は、サービスで必要なタスク数を自動的に増減します。

プロビジョニング

Amazon ECS のプロビジョニングには複数のオプションがあります。

- AWS マネジメントコンソール — Amazon ECSリソースへのアクセスに使用するウェブインターフェイスを提供します。
- AWS Command Line Interface (AWS CLI) — Amazon ECSを含む一連のさまざまな AWS サービス用のコマンドを提供します。Windows、Mac、Linux でサポートされています。詳細については、「AWS Command Line Interface」を参照してください。
- AWS SDK — 言語固有の API を提供し、接続の詳細の多くを処理します。これらには、署名の計算、リクエストの再試行処理、エラー処理などを含みます。詳細については、AWS SDK を参照してください。
- AWS CDK - 使い慣れたプログラミング言語を使用して、クラウドアプリケーションリソースをモデル化およびプロビジョニングするために使用できるオープンソースのソフトウェア開発フレームワークを提供します。AWS CDK は、AWS CloudFormation を通じて安全かつ繰り返し可能な方法でリソースをプロビジョニングします。

料金

Amazon ECS の料金は、コンテナに選択した容量オプションによって異なります。

- Amazon ECS の料金 - Amazon ECS の料金情報です。
- AWS Fargate の料金 – Fargate の料金に関する情報。

関連サービス

Amazon ECS で使用できるサービス

他の AWS のサービス を使用して、タスクとサービスを Amazon ECS にデプロイできます。

Amazon EC2 Auto Scaling

アプリケーションの負荷を処理するために適切な数の Amazon EC2 インスタンスがあることを確認できます。

Amazon CloudWatch

サービスとタスクをモニタリングします。

Amazon Elastic Container Registry

コンテナイメージを保存して管理します。

Elastic Load Balancing

受信サービストラフィックを自動的に分散します。

Amazon GuardDuty

不正な、または悪意のある可能性がある コンテナインスタンスとワークロードの使用を検出します。

Amazon ECS のリソースの作成および使用方法について説明します。

以下のガイドでは、Amazon ECS にアクセスするために利用できるツールの概要と、コンテナを実行するための初歩的な手順について説明します。Docker の基本では、Docker コンテナイメージを作成し、Amazon ECR プライベート リポジトリにアップロードする基本的な手順を説明します。入門ガイドでは、AWS Copilotのコマンドラインインターフェースと AWS マネジメントコンソールを使用して、Amazon ECS と AWS Fargate でコンテナを実行するための一般的なタスクを完了する手順について説明します。

内容

- Amazon ECS を使用するようにセットアップする
- Amazon ECS で使用するコンテナイメージの作成
- Amazon ECS マネージドインスタンスのタスクの作成方法について説明します。
- AWS CLI を使用した Amazon ECS マネージドインスタンスのタスクの作成方法について説明します。
- Fargate 用の Amazon ECS Linux タスクを作成する方法について説明します。
- Fargate 用の Amazon ECS Windows タスクを作成する方法について説明します。
- EC2 用の Amazon ECS Windows タスクを作成する方法について説明します。
- AWS CDK を使用した Amazon ECS リソースの作成
- AWS Copilot コマンドラインインターフェイスを使用した Amazon ECS リソースの作成

Amazon ECS を使用するようにセットアップする

すでに Amazon Web Services (AWS) にサインアップしていて、Amazon Elastic Compute Cloud (Amazon EC2) を使用している場合は、すぐに Amazon ECS を使用し始めることができます。2 つのサービスのセットアッププロセスは似ています。次のガイドでは、最初の Amazon ECS のクラスターを起動する準備をします。

Amazon ECS のセットアップを行うには、以下のタスクを完了します。

AWS マネジメントコンソール

AWS マネジメントコンソール は、Amazon ECS リソースを管理するためのブラウザベースのインターフェイスです。コンソールには、サービスの概要が視覚的に表示されるため、追加のツールを使用しなくても、Amazon ECS の特徴と機能を簡単に探索できます。コンソールの使用について説明した、関連するチュートリアルが多数用意されています。

コンソールに関するチュートリアルについては、「Amazon ECS のリソースの作成および使用方法について説明します。」を参照してください。

アクションが成功するかどうかについてすぐに視覚的なフィードバックがもたらされるため、多くのお客様はコンソールを使用することを好みます。AWS マネジメントコンソールに慣れているAWS お客様は、ロードバランサーや Amazon EC2 インスタンスなどの関連リソースを簡単に管理できます。

AWS マネジメントコンソール から始めます。

AWS アカウント へのサインアップ

AWS を使い始めるには、AWS アカウントが必要です。AWS アカウントの作成の詳細については、「AWS アカウント管理 リファレンスガイド」の「AWS アカウントの開始方法」を参照してください。

仮想プライベートクラウドを作成する

Amazon Virtual Private Cloud (Amazon VPC) を使用すると、定義した仮想ネットワーク内で AWS リソースを起動できます。コンテナインスタンスは、VPC で起動することを強くお勧めします。

デフォルトの VPC がある場合は、このセクションをスキップして次のタスク「セキュリティグループの作成」に移動できます。デフォルトの VPC があるかどうかを確認するには、「Amazon VPC ユーザーガイド」の「デフォルトの VPC とデフォルトのサブネットを使って作業する」を参照してください。また、次の手順を使用して、アカウントにデフォルト以外の VPC を作成することもできます。

VPC の作成方法の詳細については、「Amazon VPC ユーザーガイド」の「VPC を作成する」を参照し、次の表を使用して、選択するオプションを決定します。

オプション	値	
作成するためのリソース	VPC 専用	

オプション	値
名前	オプションで、VPC の名前を指定します。
IPv4 CIDR ブロック	IPv4 CIDR 手動入力 CIDR ブロックサイズは /16 から /28 の間である必要があります。
IPv6 CIDR ブロック	IPv6 CIDR ブロックなし
テナンシー	デフォルト

Amazon VPC の詳細については、Amazon VPC ユーザーガイドのAmazon VPC とはを参照してください。

セキュリティグループの作成

セキュリティグループは、関連付けられたコンテナインスタンスのファイアウォールとして動作し、インバウンドトラフィックとアウトバウンドトラフィックの両方をコンテナインスタンスレベルで制御します。SSH を使用して IP アドレスからコンテナインスタンスに接続するためのルールをセキュリティグループに追加できます。さらに、任意の場所からのインバウンドおよびアウトバウンドの HTTP アクセスおよび HTTPS アクセスを可能にするルールを追加できます。タスクで使用するポートを開くためのルールを追加します。コンテナインスタンスには、Amazon ECS サービスエンドポイントと通信するために外部ネットワークアクセスが必要です。

複数のリージョンでコンテナインスタンスを起動する予定がある場合は、各リージョンでセキュリティグループを作成する必要があります。詳細については、Amazon EC2 ユーザーガイド の「リージョンとアベイラビリティゾーン」を参照してください。

> **Tip**
>
> ローカルコンピュータのパブリック IP アドレスが必要になります。このアドレスはサービスを使って取得できます。例えば、次のサービスが提供されています。http://checkip.amazonaws.com/ または https://checkip.amazonaws.com/。IP アドレスを提供する別のサービスを検索するには、検索フレーズ「what is my IP address」を使用します。インターネットサービスプロバイダー (ISP) 経由で、またはファイアウォールの内側から静的 IP

アドレスなしで接続する場合は、クライアントコンピュータで使用されている IP アドレスの範囲を見つける必要があります。

セキュリティグループの作成方法の詳細については、「Amazon EC2 ユーザーガイド」の「Amazon EC2 インスタンスのセキュリティ グループの作成」を参照し、次の表を使用して選択するオプションを決定してください。

オプション	値
リージョン	キーペアを作成したリージョンと同じリージョン。
名前	「ecs–instances–default–cluster」などの覚えやすい名前。
VPC	デフォルトの VPC (「(デフォルト)」の印が付いています)。 **Note** アカウントが Amazon EC2 Classic をサポートしている場合は、前のタスクで作成した VPC を選択します。

ユースケースに追加するアウトバウンドルールの詳細については、「Amazon EC2 ユーザーガイド」の「さまざまなユースケースのセキュリティグループのルール」を参照してください。

Amazon ECS コンテナインスタンスでは、インバウンドポートが開いている必要はありません。ただし、コンテナインスタンスにログインして Docker コマンドでタスクを確認できるように、SSH ルールを追加できます。また、ウェブサーバーを実行するタスクをコンテナインスタンスでホストする場合は、HTTP と HTTPS のルールを追加できます。コンテナインスタンスには、Amazon ECS サービスエンドポイントとの通信に外部ネットワークアクセスが必要です。これらのオプションのセキュリティグループルールを追加するには、以下のステップを実行します。

セキュリティグループに次の 3 つのインバウンドルールを追加します。セキュリティグループの作成方法の詳細については、「Amazon EC2 ユーザーガイド」の「セキュリティグループのルールを設定する」を参照してください。

オプション	値	
HTTP ルール	Type: HTTP ソース: 任意の場所 (`0.0.0.0/0`) このオプションでは、送信元として IPv4 の CIDR ブロック 0.0.0.0/0 が自動的に追加されます。この設定は、テスト環境での短時間の使用であれば許容できますが、実稼働環境においては安全ではありません。実稼働環境では、特定の IP アドレスまたは特定のアドレス範囲にのみ、インスタンスへのアクセスを限定します。	
HTTPS ルール	タイプ: HTTPS ソース: 任意の場所 (`0.0.0.0/0`) この設定は、テスト環境での短時間の使用であれば許容できますが、実稼働環境においては安全ではありません。実稼働環境では、特定の IP アドレスまたは特定のアドレス範囲にのみ、インスタンスへのアクセスを限定します。	
SSH ルール	タイプ: SSH	

オプション	値	
	ソース: [Custom] で、コンピュータまたはネットワークのパブリック IP アドレスを CIDR 表記で指定します。CIDR 表記で個々の IP アドレスを指定するには、ルーティングプレフィックスを追加します。/32例えば、IP アドレスが 203.0.113.25 の場合は、203.0.113.25/32 を指定します。会社が特定の範囲からアドレスを割り当てている場合、 203.0.113.0/24 などの範囲全体を指定します。 **Important** セキュリティ上の理由で、すべての IP アドレス (0.0.0.0/0) からインスタンスへの SSH アクセスを許可することはお勧めしません。ただし、それがテスト目的で短期間の場合は例外です。	

EC2 インスタンスに接続するための認証情報を作成します。

Amazon ECS では、EC2 を使用する場合にのみキーペアが必要です。

AWS では公開キー暗号化を使用して、お客様のインスタンスのログイン情報の安全性を保護します。Amazon ECS コンテナインスタンスなどの Linux インスタンスでは、SSH アクセスにパスワー

ドを使いません。キーペアを使用してインスタンスに安全にログインします。コンテナインスタンスを起動するときにキーペアの名前を指定し、プライベートキーを指定して、SSH でログインします。

キーペアをまだ作成していない場合は、Amazon EC2 コンソールを使用して作成できます。複数のリージョンでインスタンスを起動する予定がある場合は、各リージョンでキーペアを作成する必要があります。リージョンの詳細については、「Amazon EC2 ユーザーガイド」の「Regions and Availability Zones」を参照してください。

キーペアを作成するには

- Amazon EC2 コンソールを使用して、キーペアを作成します。キーペアの作成の詳細については、「Amazon EC2 ユーザーガイド」の「Create a key pair」を参照してください。

Linux インスタンスに接続する方法については、「Amazon EC2 ユーザーガイド」の「Linux インスタンスへの接続」を参照してください。

AWS CLI のインストール

Amazon ECS を使用すると、AWS マネジメントコンソール でのすべてのオペレーションを手動で管理できます。ただし、ローカルのデスクトップまたはデベロッパーボックスに AWS CLI をインストールすることは可能です。この場合は、Amazon ECS 内の共通の管理タスクを自動化するための、スクリプトを作成できます。

Amazon ECS で AWS CLI を使用するには、最新バージョンの AWS CLI をインストールします。AWS CLI のインストールまたは最新バージョンへのアップグレードについては、「AWS Command Line Interface ユーザーガイド」の「AWS CLI の最新バージョンをインストールまたは更新」を参照してください。

AWS Command Line Interface (AWS CLI) は、AWS サービスを管理するために使用できる統合ツールです。この 1 つのツールだけでは、複数の AWS サービスを制御し、スクリプトを使用してこれらのサービスを自動化することができます。AWS CLIのAmazon ECS コマンドは、Amazon ECS API を反映しています。

AWS CLI は、スクリプト作成やコマンドラインツールとのインターフェイスを好みかつ使い慣れており、Amazon ECS リソースに対して実行するアクションを正確に把握しているお客様に適しています。AWS CLI は、Amazon ECS API に詳しくなりたいお客様にも役立ちます。お客様は、AWS CLI を使用して、Amazon ECS リソースでコマンドラインインターフェイスから直接作成、読み取り、更新、削除などの多数の操作を実行できます。

Amazon ECS API および対応する CLI コマンドに詳しくなり、自動スクリプトを作成し、Amazon ECS リソースに対して特定のアクションを実行したい場合は、AWS CLI を使用します。

AWS では、AWS Tools for Windows PowerShell コマンドラインツールも提供されています。詳細については、AWS Tools for Windows PowerShellユーザーガイドを参照してください。

Amazon ECS を使用するための次のステップ

AWS CLI をインストールした後は、さまざまなツールを使用して Amazon ECS を引き続き使用できます。以下のリンクでは、これらのツールのいくつかについて説明し、Amazon ECS で使用する方法の例を示します。

- Docker で最初のコンテナイメージを作成し、Amazon ECS タスク定義で使用するために Amazon ECR にプッシュします。
- Fargate 用の Amazon ECS Linux タスクを作成する方法について説明します。.
- Fargate 用の Amazon ECS Windows タスクを作成する方法について説明します。.
- EC2 用の Amazon ECS Windows タスクを作成する方法について説明します。.
- 希望するプログラミング言語を使用して、AWS CDK を使用した Amazon ECS リソースの作成 でインフラストラクチャまたはアーキテクチャをコードとして定義します。
- AWS を使用して、自動デプロイにより、環境内のすべての AWS CloudFormation を用いた Amazon ECS の使用 リソースを定義および管理します。
- 包括的な AWS Copilot コマンドラインインターフェイスを使用した Amazon ECS リソースの作成 エンドツーエンドのデベロッパーワークフローを使用して、インフラストラクチャの AWS ベストプラクティスに準拠したコンテナアプリケーションを作成、リリース、運用します。

Amazon ECS で使用するコンテナイメージの作成

Amazon ECS は、タスク定義で Docker イメージを使用して、コンテナを起動します。Docker は、コンテナ内の分散アプリケーションの構築、実行、テスト、デプロイするためのツールを提供するテクノロジーです。

Amazon ECS は、コンテナ化されたアプリケーションをコンテナインスタンスまたは AWS Fargate にスケジュールします。コンテナ化されたアプリケーションは、コンテナイメージとしてパッケージ化されます。この例では、ウェブサーバーのコンテナイメージを作成します。

最初の Docker イメージを作成し、そのイメージを Amazon ECS タスク定義で使用するために、コンテナレジストリである Amazon ECR にプッシュできます。このチュートリアルは、Docker の概

念と機能を基本的に理解していることを前提としています。Docker の詳細については、「Docker とは」および「Docker ドキュメント」を参照してください。

前提条件

開始する前に、以下の前提条件を満たしていることを確認します。

- Amazon ECR のセットアップステップを完了したことを確認します。詳細については、「Amazon Elastic Container Registry ユーザーガイド」の「Amazon でのライフサイクルを通じたイメージの移動 ECR」を参照してください。
- ユーザーは、Amazon ECR サービスにアクセスし、使用するために必要な IAM 権限を持ちます。詳細については、「Amazon ECRマネージドポリシー」を参照してください。
- Docker をインストールします。Amazon Linux 2023 の Docker インストールステップについては、「AL2023 に Docker をインストールする」を参照してください。他のすべてのオペレーティングシステムについては、[Docker Desktop overview] (Docker デスクトップの概要) の Docker ドキュメントを参照してください。
- AWS CLI がインストールされ、設定されている。詳細については、「AWS Command Line Interfaceユーザーガイド」の「AWS CLI の最新バージョンのインストールまたは更新」を参照してください。

ローカル開発環境がない、または不要で、Amazon EC2 インスタンスを使って Docker を使用したい場合は、Amazon Linux 2023 を使用して Amazon EC2 インスタンスを起動し、Docker Engine と Docker CLI をインストールするためのステップを以下に示します。

AL2023 に Docker をインストールする

Docker は、Ubuntu のような最新の Linux ディストリビューションから macOS や Windows まで、さまざまなオペレーティングシステムで使用できます。特定のオペレーティングシステムに Docker をインストールする方法の詳細については、Docker インストールガイド を参照してください。

Docker を使用するには、ローカルの開発システムは必要ありません。Amazon EC2 をすでに使用している場合は、Amazon Linux 2023 インスタンスを起動し、Docker をインストールして開始できます。

Docker をインストール済みの場合は、この手順をスキップして「Docker イメージの作成」に進んでください。

Amazon Linux 2023 AMI を使用して Amazon EC2 インスタンスに Docker をインストールするには

1. 最新の Amazon Linux 2023 AMI を使用してインスタンスを起動します。詳細については、「Amazon EC2 ユーザーガイド」の「コンソールのインスタンス起動ウィザードを使用して EC2 インスタンスを起動する」を参照してください。
2. インスタンスに接続します。詳細については、「Amazon EC2 ユーザーガイド」の「EC2 インスタンスへの接続」を参照してください。
3. インスタンスでインストールされているパッケージとパッケージキャッシュを更新します。

```
sudo yum update -y
```

4. 最新の Docker Community Edition パッケージをインストールします。

```
sudo yum install docker
```

5. Docker サービスを開始します。

```
sudo service docker start
```

6. ec2-user を docker グループに追加すると、sudo を使用せずに Docker コマンドを実行できます。

```
sudo usermod -a -G docker ec2-user
```

7. ログアウトし、再びログインして、新しい docker グループアクセス権限を取得します。これは、現在の SSH ターミナルウィンドウを閉じて、新しいウィンドウでインスタンスに再接続することで達成できます。新しい SSH セッションには適切な docker グループ権限があります。
8. ec2-user が sudo を使用せずに Docker コマンドを実行できることを確認します。

```
docker info
```

> **Note**
>
> 場合によっては、Docker デーモンにアクセスするための ec2-user に対するアクセス権限を提供するため、インスタンスを再起動する必要があります。次のエラーが表示された場合は、インスタンスを再起動してください。

```
Cannot connect to the Docker daemon. Is the docker daemon running on this
 host?
```

Docker イメージの作成

Amazon ECS のタスク定義では、コンテナイメージを使用してクラスター内のコンテナインスタンスでコンテナを起動します。このセクションでは、シンプルなウェブアプリケーションの Docker イメージを作成し、ローカルシステムまたは Amazon EC2 インスタンスでテストしてから、そのイメージを Amazon ECR コンテナレジストリにプッシュして、Amazon ECS タスク定義で使用できるようにします。

シンプルなウェブアプリケーションの Docker イメージを作成するには

1. `Dockerfile` という名前のファイルを作成します。Dockerfile は、Docker イメージに使用する基本イメージと、そのイメージにインストールして実行するものを記述するマニフェストです。Dockerfile の詳細については、「Dockerfile リファレンス」を参照してください。

   ```
   touch Dockerfile
   ```

2. 前の手順で作成した `Dockerfile` を編集し、以下のコンテンツを追加します。

   ```
   FROM public.ecr.aws/amazonlinux/amazonlinux:latest

   # Update installed packages and install Apache
   RUN yum update -y && \
    yum install -y httpd

   # Write hello world message
   RUN echo 'Hello World!' > /var/www/html/index.html

   # Configure Apache
   RUN echo 'mkdir -p /var/run/httpd' >> /root/run_apache.sh && \
    echo 'mkdir -p /var/lock/httpd' >> /root/run_apache.sh && \
    echo '/usr/sbin/httpd -D FOREGROUND' >> /root/run_apache.sh && \
    chmod 755 /root/run_apache.sh

   EXPOSE 80
   ```

```
CMD /root/run_apache.sh
```

この Dockerfile は、Amazon ECR パブリックでホストされているパブリック Amazon Linux 2023 イメージを使用します。RUN の手順により、パッケージキャッシュが更新され、ウェブサーバー用のいくつかのソフトウェアがインストールされてから、「Hello World!」 のコンテンツがウェブサーバーのドキュメントルートに書き込みされます。EXPOSE の命令でコンテナ上のポート 80 がリッスン中のものになり、CMD の命令でウェブサーバーが起動します。

3. Dockerfile から Docker イメージを作成します。

> **Note**
>
> Docker の一部のバージョンでは、次の例に示す相対パスの代わりに、次のコマンドで Dockerfile への完全パスが必要になる場合があります。
> Apple Silicon などの ARM ベースのシステムでコマンドを実行する場合は、--platform オプション「--platform linux/amd64」を使用します。

```
docker build -t hello-world .
```

4. コンテナイメージを一覧表示します。

```
docker images --filter reference=hello-world
```

出力:

```
REPOSITORY          TAG                 IMAGE ID            CREATED
 SIZE
hello-world         latest              e9ffedc8c286        4 minutes ago
 194MB
```

5. 新しく構築されたイメージを実行します。-p 80:80 オプションは、コンテナ上の公開されたポート 80 をホストシステム上のポート 80 にマッピングします。

```
docker run -t -i -p 80:80 hello-world
```

> **Note**
>
> Apache ウェブサーバーからの出力はターミナルウィンドウに表示されます。"Could not reliably determine the fully qualified domain name" メッセージは無視できます。

6. ブラウザーを開き、Docker を実行している、コンテナのホストサーバーを参照します。

   - EC2 インスタンスを使用している場合、これはサーバーの [Public DNS] 値であり、SSH でインスタンスに接続するときに使用するアドレスと同じです。インスタンスのセキュリティグループでポート 80 上の受信トラフィックを許可していることを確認します。
   - Docker をローカルに実行している場合は、ブラウザで http://localhost/ を参照します。

   ウェブページに「Hello, World!」が確認できます。表示されます。

7. [Ctrl + C] キーを押して、Docker コンテナを停止します。

Amazon Elastic Container Registry にイメージをプッシュします

Amazon ECR は、AWS マネージドイメージレジストリサービスです。Docker CLI を使用して、Amazon ECR リポジトリ内のイメージをプッシュ、プル、および管理できます。Amazon ECR 製品の詳細、主なお客様導入事例、FAQ については、Amazon Elastic コンテナレジストリ 製品の詳細ページを参照してください。

イメージにタグを付け、Amazon ECR にプッシュするには

1. `hello-world` イメージを保存する Amazon ECR リポジトリを作成します。出力の `repositoryUri` に注目してください。

   `region` を、AWS リージョン に置き換えます (例えば `us-east-1`)。

   ```
   aws ecr create-repository --repository-name hello-repository --region region
   ```

   出力:

   ```
   {
       "repository": {
           "registryId": "aws_account_id",
   ```

```
        "repositoryName": "hello-repository",
        "repositoryArn": "arn:aws:ecr:region:aws_account_id:repository/hello-
repository",
        "createdAt": 1505337806.0,
        "repositoryUri": "aws_account_id.dkr.ecr.region.amazonaws.com/hello-
repository"
    }
}
```

2. 前のステップの hello-world の値で repositoryUri イメージにタグを付けます。

```
docker tag hello-world aws_account_id.dkr.ecr.region.amazonaws.com/hello-repository
```

3. aws ecr get-login-password コマンドを実行します。認証先のレジストリ URI を指定します。詳細については、Amazon Elastic Container Registry ユーザーガイドの「レジストリの認証」を参照してください。

```
aws ecr get-login-password --region region | docker login --username AWS --
password-stdin aws_account_id.dkr.ecr.region.amazonaws.com
```

出力:

```
Login Succeeded
```

> **Important**
>
> エラーが発生した場合は、AWS CLI の最新バージョンをインストールまたはアップグレードします。詳細については、「AWS Command Line Interface ユーザーガイド」の「AWS CLI の最新バージョンをインストールまたは更新」を参照してください。

4. repositoryUri前のステップの値 を使用して、Amazon ECR にイメージをプッシュします。

```
docker push aws_account_id.dkr.ecr.region.amazonaws.com/hello-repository
```

クリーンアップ

Amazon ECS タスク定義の作成、およびコンテナイメージを使用したタスク起動を続行するには、次のステップ に進んでください。Amazon ECR イメージの試用を終了したら、レポジトリを削除し、イメージストレージに対して課金されないようにします。

```
aws ecr delete-repository --repository-name hello-repository --region region --force
```

次のステップ

タスク定義にはタスク実行ロールが必要です。詳細については、「Amazon ECS タスク実行IAM ロール」を参照してください。

コンテナイメージを作成して Amazon ECR にプッシュしたら、そのイメージをタスク定義で使用できます。詳細については、以下のいずれかを参照してください。

- the section called “Fargate 用のタスクを作成する方法について説明します。”
- the section called “Fargate 用の Windows タスクを作成する方法について説明します。”
- AWS CLI を使用して、Fargate 用の Amazon ECS Linux タスクを作成する

Amazon ECS マネージドインスタンスのタスクの作成方法について説明します。

Amazon ECS マネージドインスタンスで、Amazon ECS を使用してコンテナ化されたアプリケーションを実行する方法について説明します。

前提条件

このチュートリアルを開始する前に以下を完了してください。

- 「Amazon ECS を使用するようにセットアップする」のステップを完了していること。
- 「the section called “セットアップ”」のステップを完了していること。
- Amazon ECS マネージドインスタンスに必要な IAM ロールがあること。これには、以下が含まれます。
  - インフラストラクチャロール – Amazon ECS がユーザーに代わって AWS サービスを呼び出し、Amazon ECS マネージドインスタンスインフラストラクチャを管理できるようにします。

詳細については、「Amazon ECS インフラストラクチャ IAM ロール」を参照してください。

- インスタンスプロファイル – マネージドインスタンスで実行されている Amazon ECS コンテナエージェントと Docker デーモンへのアクセス許可を提供します。

  インスタンスロール名には、インフラストラクチャロールの `iam:PassRole` アクションと一致するプレフィックスとして `ecsInstanceRole` を含める必要があります。

  詳細については、「Amazon ECS マネージドインスタンスのインスタンスプロファイル」を参照してください。

- VPC およびセキュリティグループが使用できるように作成されていること。このチュートリアルは、Amazon ECR パブリックにホストされているコンテナイメージを使用するため、インスタンスではインターネットアクセスが必要になります。インスタンスでインターネットへのルートを設定するには、以下のいずれかのオプションを使用します。
  - Elastic IP アドレスが割り当てられた NAT ゲートウェイでプライベートサブネットを使用する。
  - パブリックサブネットを使用してインスタンスにパブリック IP アドレスを割り当てる。

  詳細については、「the section called “仮想プライベートクラウドを作成する”」を参照してください。

  セキュリティグループとルールの詳細については、「Amazon Virtual Private Cloud ユーザーガイド」の「VPC のデフォルトセキュリティグループ」および「ルールの例」を参照してください。

- (オプション) AWS CloudShell は、お客様が独自の EC2 インスタンスを作成することなくコマンドラインを提供するツールです。詳細については、「AWS CloudShell ユーザーガイド」の「AWS CloudShell とは」を参照してください。

ステップ 1: クラスターを作成する

1. Amazon ECS コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションバーから、使用するリージョンを選択します。
3. ナビゲーションペインで [Clusters] (クラスター) を選択します。
4. [Clusters] (クラスター) ページで、[Create Cluster] (クラスターの作成) を選択します。
5. [クラスター設定] で、[クラスター名] にクラスターの一意の名前を入力します。
6. [インフラストラクチャ] で、[Fargate とマネージド EC2] を選択します。
7. マネージドインスタンスの設定を次のように行います。

a. [インフラストラクチャロール] で、マネージドインスタンスのインフラストラクチャ管理用に作成した IAM ロールを選択します。
b. [IAM インスタンスプロファイル] で、作成した `ecsInstanceRole` を選択します。
c. [インスタンス属性] で、[ECS デフォルトを使用する] を選択します。

8. [ネットワーキング]で、マネージドインスタンスの VPC とサブネットを設定します。

a. [VPC] で、ターゲットインスタンスをホストする VPC を選択してください。
b. [サブネット]で、マネージドインスタンスを起動するサブネットを 1 つ以上選択します。
c. [Security groups] で、1 つまたは複数のセキュリティグループを選択してください。

9. (オプション) クラスターにタグを追加するには、[タグ] を展開して設定します。
10. [作成] を選択します。

ステップ 2: タスク定義を作成する

タスク定義はアプリケーションのブループリントです。Amazon ECS でタスクを起動するたびに、タスク定義を指定します。サービスは、コンテナに使用する Docker イメージ、タスクで使用するコンテナの数、各コンテナのリソース割り当てを認識します。タスク定義を作成するには、次の手順に従います。

1. ナビゲーションペインで、[タスク定義] を選択します。
2. [Create new task definition] (新しいタスク定義の作成)、[Create new task definition with JSON] (JSON で新しいタスク定義を作成) の順に選択します。
3. 次の JSON をコピーしてエディタに貼り付け、事前入力された JSON を置き換えます。

   `account-id` を AWS アカウント ID に、**`region`** を使用中のリージョンに置き換えます。

```
{
  "family": "managed-instance-tutorial",
  "networkMode": "awsvpc",
  "executionRoleArn": "arn:aws:iam::account-id:role/ecsTaskExecutionRole",
  "containerDefinitions": [
    {
      "name": "sample-app",
      "image": "public.ecr.aws/docker/library/httpd:latest",
      "essential": true,
      "portMappings": [
        {
```

```
          "containerPort": 80,
          "hostPort": 80,
          "protocol": "tcp"
        }
      ],
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "/ecs/managed-instance-tutorial",
          "awslogs-region": "region",
          "awslogs-stream-prefix": "ecs"
        }
      }
    }
  ],
  "requiresCompatibilities": [
    "MANAGED_INSTANCES"
  ],
  "cpu": "1024",
  "memory": "2048"
}
```

4. [作成] を選択します。

ステップ 3: サービスを作成する

Amazon ECS サービスを使用すると、 Amazon ECS クラスター内で、タスク定義の指定した数のインスタンスを同時に実行して維持できます。サービスを作成するには、次の手順に従います。

1. ナビゲーションペインで、[クラスター] を選択し、次に **managed-instance-tutorial** クラスターを選択します。
2. [Services] (サービス) タブから、[Create] (作成) を選択します。
3. [タスク定義ファミリー] で、[managed-instance-tutorial] を選択します。
4. [サービス名] に **managed-instance-tutorial-service** と入力します。
5. [環境] で、[キャパシティプロバイダー戦略]を選択します。
6. [ネットワーク] で、以下を設定します。
   a. 既存の VPC を選択するか、新しい VPC を作成します。
   b. サブネット では、使用するサブネットを選択します。

c. [セキュリティグループ] では、既存のセキュリティグループを選択することも、新しいセキュリティグループ (ポート 80 でインバウンドトラフィックを許可) を作成することもできます。

7. [作成] を選択します。

ステップ 4: サービスを表示する

サービスが起動したら表示して、詳細の確認やテストができます。

1. [**managed-instance-tutorial-service** サービス] を選択します。
2. [タスク] タブで、実行中のタスクのタスク ID を選択します。
3. [ネットワーク] の [パブリック IP] で、[アドレスを開く] を選択します。
4. ウェブサーバーが適切に実行されていることを示す Apache HTTP サーバーのテストページが表示されます。

ステップ 5: クリーンアップ

このチュートリアルが終了したら、使用していないリソースに対する料金が発生しないように、関連付けられたリソースをクリーンアップする必要があります。

1. ナビゲーションペインで [Clusters] (クラスター) を選択します。
2. [クラスター] ページで、**managed-instance-tutorial** クラスターを選択します。
3. [サービス] タブを選択します。
4. [**managed-instance-tutorial-service** サービス]、[削除] の順に選択します。
5. 確認プロンプトで **delete** と入力し、[削除] を選択します。
6. サービスが削除されたら、ナビゲーションペインで [クラスター] を選択します。
7. [クラスター] ページで **managed-instance-tutorial** クラスターを選択し、[クラスターの削除] を選択します。
8. 確認プロンプトで **delete managed-instance-tutorial** と入力し、[削除] を選択します。

AWS CLI を使用した Amazon ECS マネージドインスタンスのタスクの作成方法について説明します。

次のステップは、Amazon ECS (Amazon ECS マネージドインスタンスあり) における AWS CLI を使ったクラスターのセットアップ、キャパシティプロバイダーの作成、タスク定義の登録、Linux タスクの実行、その他の一般的なシナリオを実行するために役立ちます。AWS CLI の最新バージョンを使用する。最新のバージョンにアップグレードする方法については、「AWS CLI の最新バージョンをインストールまたは更新」を参照してください。

> **Note**
>
> デュアルスタックサービスエンドポイントを使用することで、AWS AWS CLI、SDK、Amazon ECS API から、IPv4 と IPv6 の両方を経由して Amazon ECS とやり取りができます。詳細については、「the section called “デュアルスタックのエンドポイントの使用”」を参照してください。

トピック

- the section called “前提条件”
- the section called “ステップ 1: クラスターを作成する”
- the section called “ステップ 2: Amazon ECS マネージドインスタンスキャパシティプロバイダーを作成する”
- the section called “ステップ 3: クラスターのデフォルトのキャパシティプロバイダー戦略を設定する”
- the section called “ステップ 4: Linux タスク定義を登録する”
- the section called “ステップ 5: タスク定義をリスト表示する”
- the section called “ステップ 6: サービスを作成する”
- the section called “ステップ 7: サービスをリスト表示する”
- the section called “ステップ 8: 実行中のサービスを記述する”
- the section called “ステップ 9: テスト”
- the section called “ステップ 10: クリーンアップする”

前提条件

このチュートリアルを開始する前に以下を完了してください。

- 「Amazon ECS を使用するようにセットアップする」のステップを完了していること。
- AWS CLI の最新バージョンがインストールされ、設定されていること。AWS CLI のインストールまたはアップグレードについては、「AWS Command Line Interfaceユーザーガイド」の「AWS CLI の最新バージョンのインストールまたは更新」を参照してください。
- 「the section called “セットアップ”」のステップを完了していること。
- Amazon ECS マネージドインスタンスに必要な IAM ロールがあること。これには、以下が含まれます。
  - インフラストラクチャロール – Amazon ECS がユーザーに代わって AWS サービスを呼び出し、Amazon ECS マネージドインスタンスインフラストラクチャを管理できるようにします。

    詳細については、「Amazon ECS インフラストラクチャ IAM ロール」を参照してください。
  - インスタンスプロファイル – マネージドインスタンスで実行されている Amazon ECS コンテナエージェントと Docker デーモンへのアクセス許可を提供します。

    インスタンスロール名には、インフラストラクチャロールの `iam:PassRole` アクションと一致するプレフィックスとして `ecsInstanceRole` を含める必要があります。

    詳細については、「Amazon ECS マネージドインスタンスのインスタンスプロファイル」を参照してください。
- VPC およびセキュリティグループが使用できるように作成されていること。このチュートリアルは、Amazon ECR パブリックにホストされているコンテナイメージを使用するため、インスタンスではインターネットアクセスが必要になります。インスタンスでインターネットへのルートを設定するには、以下のいずれかのオプションを使用します。
  - Elastic IP アドレスが割り当てられた NAT ゲートウェイでプライベートサブネットを使用する。
  - パブリックサブネットを使用してインスタンスにパブリック IP アドレスを割り当てる。

  詳細については、「the section called “仮想プライベートクラウドを作成する”」を参照してください。

  セキュリティグループとルールの詳細については、「Amazon Virtual Private Cloud ユーザーガイド」の「VPC のデフォルトセキュリティグループ」および「ルールの例」を参照してください。

- (オプション) AWS CloudShell は、お客様が独自の EC2 インスタンスを作成することなくコマンドラインを提供するツールです。詳細については、「AWS CloudShell ユーザーガイド」の「AWS CloudShell とは」を参照してください。

ステップ 1: クラスターを作成する

デフォルトでは、アカウントは default クラスターを受け取ります。

> **Note**
>
> お客様に提供される default クラスターを使用する利点は、後続のコマンドで --cluster *cluster_name* オプションを指定する必要がないことです。デフォルト以外の独自のクラスターを作成する場合は、そのクラスターで使用する予定のコマンドごとに --cluster *cluster_name* を指定する必要があります。

以下のコマンドを使用して、一意の名前を付けた独自のクラスターを作成します。

```
aws ecs create-cluster --cluster-name managed-instances-cluster
```

出力:

```
{
    "cluster": {
        "status": "ACTIVE",
        "defaultCapacityProviderStrategy": [],
        "statistics": [],
        "capacityProviders": [],
        "tags": [],
        "clusterName": "managed-instances-cluster",
        "settings": [
            {
                "name": "containerInsights",
                "value": "disabled"
            }
        ],
        "registeredContainerInstancesCount": 0,
        "pendingTasksCount": 0,
        "runningTasksCount": 0,
        "activeServicesCount": 0,
```

```
        "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/managed-instances-
cluster"
    }
}
```

ステップ 2: Amazon ECS マネージドインスタンスキャパシティプロバイダーを作成する

Amazon ECS マネージドインスタンスを使用してタスクを実行する前に、インフラストラクチャ設定を定義するキャパシティプロバイダーを作成する必要があります。キャパシティプロバイダーは、マネージドインスタンスの IAM ロール、ネットワーク設定、その他の設定を指定します。

キャパシティプロバイダー設定を使用して JSON ファイルを作成します。プレースホルダー値を実際のリソース識別子に置き換えます。

```
{
    "name": "managed-instances-cp",
    "cluster": "managed-instances-cluster",
    "managedInstancesProvider": {
        "infrastructureRoleArn":
 "arn:aws:iam::aws_account_id:role/ecsInfrastructureRole",
        "instanceLaunchTemplate": {
            "ec2InstanceProfileArn": "arn:aws:iam::aws_account_id:instance-
profile/ecsInstanceRole",
            "networkConfiguration": {
                "subnets": [
                    "subnet-abcdef01234567890",
                    "subnet-1234567890abcdef0"
                ],
                "securityGroups": [
                    "sg-0123456789abcdef0"
                ]
            },
            "storageConfiguration": {
                "storageSizeGiB": 100
            },
            "monitoring": "basic"
        }
    }
}
```

この設定を managed-instances-cp.json として保存し、キャパシティプロバイダーを作成します。

```
aws ecs create-capacity-provider --cli-input-json file://managed-instances-cp.json
```

コマンドは、作成完了後にキャパシティプロバイダーの説明を返します。

ステップ 3: クラスターのデフォルトのキャパシティプロバイダー戦略を設定する

Amazon ECS マネージドインスタンスキャパシティプロバイダーを、デフォルトのキャパシティプロバイダー戦略として使用するようにクラスターを更新します。こうすることで、タスクとサービスは、キャパシティプロバイダーを明示的に指定することなく、Amazon ECS マネージドインスタンスを自動的に使用できるようになります。

クラスターキャパシティプロバイダー設定を使用して JSON ファイルを作成します。

```
{
    "cluster": "managed-instances-cluster",
    "capacityProviders": [
        "managed-instances-cp"
    ],
    "defaultCapacityProviderStrategy": [
        {
            "capacityProvider": "managed-instances-cp",
            "weight": 1
        }
    ]
}
```

この設定ファイルを cluster-cp-strategy.json という名前で保存して、クラスターを更新します。

```
aws ecs put-cluster-capacity-providers --cli-input-json file://cluster-cp-strategy.json
```

ステップ 4: Linux タスク定義を登録する

クラスターでタスクを実行する前に、タスク定義を登録する必要があります。タスク定義とは、1 つにグループ化されたコンテナのリストです。以下の例は、Docker Hub でホストされている httpd コンテナイメージを使用して PHP ウェブアプリケーションを作成するシンプルなタスク定義です。使

用できるタスク定義パラメータの詳細については、「the section called “Fargate のタスク定義パラメータ”」を参照してください。

```
{
    "family": "sample-managed-instances",
    "networkMode": "awsvpc",
    "containerDefinitions": [
        {
            "name": "managed-instances-app",
            "image": "public.ecr.aws/docker/library/httpd:latest",
            "portMappings": [
                {
                    "containerPort": 80,
                    "hostPort": 80,
                    "protocol": "tcp"
                }
            ],
            "essential": true,
            "entryPoint": [
                "sh",
                "-c"
            ],
            "command": [
                "/bin/sh -c \"echo '<html><head><title>Amazon ECS Sample App</
title><style>body {margin-top: 40px; background-color: #333;} </style></head><body><div
 style=color:white;text-align:center><h1>Amazon ECS Sample App</h1><h2>Congratulations!
</h2><p>Your application is now running on a container in Amazon ECS using Amazon ECS
 Managed Instances.</p></div></body></html>' >  /usr/local/apache2/htdocs/index.html &&
 httpd-foreground\""
            ]
        }
    ],
    "requiresCompatibilities": [
        "MANAGED_INSTANCES"
    ],
    "cpu": "256",
    "memory": "512"
}
```

タスク定義の JSON をファイルとして保存し、`--cli-input-json file://path_to_file.json` オプションで渡します。

コンテナの定義に JSON ファイルを使用するには。

```
aws ecs register-task-definition --cli-input-json file://$HOME/tasks/managed-instances-task.json
```

register-task-definition コマンドによって、タスク定義の登録が完了した後、その定義の説明が返されます。

ステップ 5: タスク定義をリスト表示する

list-task-definitions コマンドを使用して、いつでもアカウントのタスク定義をリスト表示できます。このコマンドの出力は、`family` または `revision` をコールするときに一緒に使用する、run-task 値および start-task 値を表示します。

```
aws ecs list-task-definitions
```

出力:

```
{
    "taskDefinitionArns": [
        "arn:aws:ecs:region:aws_account_id:task-definition/sample-managed-instances:1"
    ]
}
```

ステップ 6: サービスを作成する

アカウントのタスクを登録したら、クラスターに登録されたタスク用のサービスを作成することができます。この例では、`sample-managed-instances:1` タスク定義の 1 つのインスタンスを使用して、クラスターで実行されるサービスを作成します。このタスクにはインターネットへのルートが必要であり、そのために 2 つの方法のいずれかを使用します。1 つの方法は、パブリックサブネットに Elastic IP アドレスを持つ NAT ゲートウェイで構成されたプライベートサブネットを使用することです。もう 1 つの方法は、パブリックサブネットを使用してパブリック IP アドレスをタスクに割り当てることです。両方の例を以下に示します。

プライベートサブネットを使用した例:

```
aws ecs create-service --cluster managed-instances-cluster --service-name managed-instances-service --task-definition sample-managed-instances:1 --desired-count 1 --network-configuration "awsvpcConfiguration={subnets=[subnet-abcd1234],securityGroups=[sg-abcd1234]}"
```

パブリックサブネットを使用した例:

```
aws ecs create-service --cluster managed-instances-cluster --service-name managed-
instances-service --task-definition sample-managed-instances:1 --desired-
count 1 --network-configuration "awsvpcConfiguration={subnets=[subnet-
abcd1234],securityGroups=[sg-abcd1234],assignPublicIp=ENABLED}"
```

create-service コマンドは、作成完了後にサービスの説明を返します。

ステップ 7: サービスをリスト表示する

クラスターのサービスをリスト表示します。前のセクションで作成したサービスが表示されます。このコマンドでサービス名または完全 ARN を取得できます。これを後で説明するサービスに使用します。

```
aws ecs list-services --cluster managed-instances-cluster
```

出力:

```
{
    "serviceArns": [
        "arn:aws:ecs:region:aws_account_id:service/managed-instances-cluster/managed-
instances-service"
    ]
}
```

ステップ 8: 実行中のサービスを記述する

先に取得したサービス名を使用してタスクを記述し、サービスに関する詳細情報を取得します。

```
aws ecs describe-services --cluster managed-instances-cluster --services managed-
instances-service
```

このコマンドが正常に実行されると、サービスの障害とサービスの説明が返されます。例えば、services セクションでは、タスクの実行中または保留中のステータスなど、デプロイに関する情報が見つかります。また、タスク定義、ネットワーク設定、タイムスタンプ付きのイベントに関する情報も見つかります。障害セクションでは、呼び出しに関連する障害がある場合にその情報が見つかります。

出力では、サービスが Amazon ECS マネージドインスタンスキャパシティプロバイダーを使用していることが示されます。

```
{
    "services": [
        {
            "capacityProviderStrategy": [
                {
                    "capacityProvider": "managed-instances-cp",
                    "weight": 1,
                    "base": 0
                }
            ],
            "networkConfiguration": {
                "awsvpcConfiguration": {
                    "subnets": [
                        "subnet-abcd1234"
                    ],
                    "securityGroups": [
                        "sg-abcd1234"
                    ],
                    "assignPublicIp": "ENABLED"
                }
            },
            "enableECSManagedTags": false,
            "loadBalancers": [],
            "deploymentController": {
                "type": "ECS"
            },
            "desiredCount": 1,
            "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/managed-instances-
cluster",
            "serviceArn": "arn:aws:ecs:region:aws_account_id:service/managed-instances-
cluster/managed-instances-service",
            "serviceName": "managed-instances-service",
            "taskDefinition": "arn:aws:ecs:region:aws_account_id:task-definition/
sample-managed-instances:1"
        }
    ],
    "failures": []
}
```

ステップ 9: テスト

デプロイをテストするには、タスクを実行しているマネージドインスタンスのパブリック IP アドレスを見つける必要があります。

パブリックサブネットを使用してデプロイされたタスクのテスト

まず、サービスからタスク ARN を取得します。

```
aws ecs list-tasks --cluster managed-instances-cluster --service managed-instances-
service
```

出力にはタスク ARN が含まれます。

```
{
    "taskArns": [
        "arn:aws:ecs:region:aws_account_id:task/managed-instances-cluster/EXAMPLE"
    ]
}
```

コンテナインスタンス ARN を取得するタスクを記述します。tasks パラメータにはタスク ARN を使用します。

```
aws ecs describe-tasks --cluster managed-instances-cluster --tasks
 arn:aws:ecs:region:aws_account_id:task/managed-instances-cluster/EXAMPLE
```

出力には、タスクが Amazon ECS マネージドインスタンスで実行されており、コンテナインスタンス ARN が含まれていることが示されます。

```
{
    "tasks": [
        {
            "launchType": "MANAGED_INSTANCES",
            "capacityProviderName": "managed-instances-cp",
            "containerInstanceArn": "arn:aws:ecs:region:aws_account_id:container-
instance/managed-instances-cluster/CONTAINER_INSTANCE_ID",
            "taskArn": "arn:aws:ecs:region:aws_account_id:task/managed-instances-
cluster/EXAMPLE",
            "taskDefinitionArn": "arn:aws:ecs:region:aws_account_id:task-definition/
sample-managed-instances:1"
```

```
        }
    ]
}
```

EC2 インスタンス ID を取得するコンテナインスタンスを記述します。

```
aws ecs describe-container-instances --cluster managed-instances-cluster --container-
instances CONTAINER_INSTANCE_ID
```

出力には EC2 インスタンス ID が含まれます。

```
{
    "containerInstances": [
        {
            "ec2InstanceId": "i-1234567890abcdef0",
            "capacityProviderName": "managed-instances-cp",
            "containerInstanceArn": "arn:aws:ecs:region:aws_account_id:container-
instance/managed-instances-cluster/CONTAINER_INSTANCE_ID"
        }
    ]
}
```

パブリック IP アドレスを取得するには EC2 インスタンスを記述します。

```
aws ec2 describe-instances --instance-ids i-1234567890abcdef0
```

パブリック IP アドレスが出力に表示されます。

```
{
    "Reservations": [
        {
            "Instances": [
                {
                    "PublicIpAddress": "198.51.100.2",
                    "InstanceId": "i-1234567890abcdef0"
                }
            ]
        }
    ]
}
```

ウェブブラウザにパブリック IP アドレスを入力すると、Amazon ECS マネージドインスタンスで実行中の Amazon ECS のサンプルアプリケーションを表示するウェブページが確認できます。

プライベートサブネットを使用してデプロイされたタスクのテスト

プライベートサブネットにデプロイされたタスクの場合、Amazon ECS Exec を使用してコンテナに接続し、インスタンス内からデプロイをテストできます。前述の手順と同じ手順に従ってタスク ARN を取得し、ECS Exec を使用します。

```
aws ecs execute-command --cluster managed-instances-cluster \
    --task arn:aws:ecs:region:aws_account_id:task/managed-instances-cluster/EXAMPLE \
    --container managed-instances-app \
    --interactive \
    --command "/bin/sh"
```

インタラクティブシェルが実行されたら、ウェブサーバーをテストできます。

```
curl localhost
```

Amazon ECS サンプルアプリケーションのウェブページと同等の HTML が表示されます。

ステップ 10: クリーンアップする

このチュートリアルが終了したら、未使用のリソースに対する料金が発生しないように、それに関連付けられたリソースをクリーンアップする必要があります。

サービスを削除します。

```
aws ecs delete-service --cluster managed-instances-cluster --service managed-instances-service --force
```

サービスが削除され、すべてのタスクが停止するまで待機してから、キャパシティプロバイダーを削除します。

```
aws ecs delete-capacity-provider --capacity-provider managed-instances-cp
```

クラスターを削除します。

```
aws ecs delete-cluster --cluster managed-instances-cluster
```

> **Note**
>
> マネージドインスタンスは、キャパシティプロバイダーが削除されると自動的に終了します。EC2 インスタンスを手動で終了する必要はありません。

Fargate 用の Amazon ECS Linux タスクを作成する方法について説明します。

Amazon Elastic Container Service (Amazon ECS) は、非常にスケーラブルで高速なコンテナ管理サービスで、クラスターでコンテナの実行、停止、管理を簡単に行うことができます。AWS Fargate でサービスやタスクを起動して、Amazon ECS 管理のサーバーレスインフラストラクチャにあるコンテナをホストすることができます。Fargate の詳細については、Amazon ECS の AWS Fargate 用のアーキテクト を参照してください。

Amazon ECS が AWS Fargate をサポートするリージョンで、タスクに Fargate を使用して、Amazon ECS on AWS Fargate の利用を開始します。

AWS Fargateで Amazon ECSの使用を開始するには、以下のステップを完了します。

前提条件

開始する前に、Amazon ECS を使用するようにセットアップする のステップを完了し、IAM ユーザーに `AdministratorAccess` IAM ポリシー例で指定されているアクセス許可があることを確認します。

コンソールは、Fargate タスクに必要なタスク実行 IAM ロールを自動で作成しようとします。コンソールがこの IAM ロールを作成できるようにするには、次のいずれかがtrueである必要があります。

- ユーザーが管理者権限を持っていることが必要です。詳細については、「Amazon ECS を使用するようにセットアップする」を参照してください。
- ユーザーがサービスロールを作成するための IAM 権限を持っていることが必要です。詳細については、「AWS サービスにアクセス許可を委任するロールの作成」を参照してください。
- 管理者権限を持つユーザーは、タスク実行ロールを手動で作成することにより、使用するアカウントで有効にできます。詳細については、「Amazon ECS タスク実行IAM ロール」を参照してください。

> **Important**
>
> タスク定義を使用してサービスを作成するときに選択するセキュリティグループには、インバウンドトラフィック用にポート 80 が開いている必要があります。セキュリティグループに次のインバウンドルールを追加します。セキュリティグループの作成方法については、「Amazon EC2 ユーザーガイド」の「Amazon EC2 インスタンス用のセキュリティグループの作成」を参照してください。
>
> - [Type]: HTTP
> - [Protocol]: TCP
> - ポート範囲: 80
> - ソース: 任意の場所 (`0.0.0.0/0`)

ステップ 1: クラスターを作成する

デフォルトの VPC を使用するクラスターを作成します。

開始する前に、適切な IAM アクセス許可を割り当ててください。詳細については、「the section called “Amazon ECS クラスターの例”」を参照してください。

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションバーから、使用するリージョンを選択します。
3. ナビゲーションペインで [Clusters] (クラスター) を選択します。
4. [Clusters] (クラスター) ページで、[Create Cluster] (クラスターの作成) を選択します。
5. [Cluster configuration] (クラスター設定) で、[Cluster name] (クラスター名) に一意の名前を入力します。

   名前には、最大 255 文字 (大文字と小文字)、数字、およびハイフンを含めることができます。
6. (オプション) Container Insights を有効にするには、[Monitoring] (モニタリング) を展開し、[Use Container Insights] (Container Insights を使用する) をオンにします。
7. (オプション) クラスターを識別しやすくするには、[Tags] (タグ) を展開し、タグを設定します。

   [タグの追加] [タグの追加] を選択して、以下を実行します。

   - [キー] にはキー名を入力します。
   - [値] にキー値を入力します。

[タグを削除] タグのキーと値の右側にある [削除] を選択します。

8. [作成] を選択します。

ステップ 2: タスク定義を作成する

タスク定義はアプリケーションの設計図のようなものです。Amazon ECS でタスクを起動するたびに、タスク定義を指定します。サービスは、コンテナに使用する Docker イメージ、タスクで使用するコンテナの数、各コンテナのリソース割り当てを認識します。

1. ナビゲーションペインで、[タスク定義] を選択します。
2. [Create new Task Definition] (新しいタスク定義の作成)、[Create new revision with JSON] (JSON で新しいリビジョンを作成) の順に選択します。
3. 以下のタスク定義の例をコピーしてボックスに貼り付け、[Save (保存)] を選択します。

```
{
    "family": "sample-fargate",
    "networkMode": "awsvpc",
    "containerDefinitions": [
        {
            "name": "fargate-app",
            "image": "public.ecr.aws/docker/library/httpd:latest",
            "portMappings": [
                {
                    "containerPort": 80,
                    "hostPort": 80,
                    "protocol": "tcp"
                }
            ],
            "essential": true,
            "entryPoint": [
                "sh",
  "-c"
            ],
            "command": [
                "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample
 App</title> <style>body {margin-top: 40px; background-color: #333;} </style> </
head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</
h1> <h2>Congratulations!</h2> <p>Your application is now running on a container in
```

```
 Amazon ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html &&
 httpd-foreground\""
            ]
        }
    ],
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "cpu": "256",
    "memory": "512"
}
```

4. [作成] を選択します。

ステップ 3: サービスを作成する

タスク定義を使用してサービスを作成します。

1. ナビゲーションペインで、[Clusters] (クラスター) を選択してから、ステップ 1: クラスターを作成する で作成したクラスターを選択します。
2. [Services] (サービス) タブから、[Create] (作成) を選択します。
3. [サービスの詳細] で、アプリケーションのデプロイ方法を指定します。
   a. [タスク定義ファミリー] で、ステップ 2: タスク定義を作成する で作成したタスク定義を選択します。
   b. [Service name] (サービス名) でサービスの名前を入力します。
4. [環境] で [起動タイプ] を選択し、FARGATE を選択します。
5. [デプロイ設定] で、[必要なタスク]に 1 と入力します。
6. [ネットワーク] で、タスク用に新しいセキュリティグループを作成するか、既存のセキュリティグループを選択できます。使用するセキュリティグループに、前提条件 に一覧表示されているインバウンドルールがあることを確認してください。
7. [作成] を選択します。

ステップ 4: サービスを表示する

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。

3. サービスを実行したクラスターを選択します。
4. [サービス] タブの [サービス名] で、ステップ 3: サービスを作成する に作成したサービスを選択します。
5. [タスク] タブ > サービス内のタスクの順に選択します。
6. [タスク] ページの [設定] セクションの [パブリック IP] で、[オープンアドレス] を選択します。

ステップ 5: クリーンアップ

Amazon ECS クラスターの使用が終了したら、使用していないリソースに対する料金が発生しないよう、それに関連付けられたリソースをクリーンアップする必要があります。

タスク、サービス、クラスター、コンテナインスタンスなど、一部の Amazon ECS リソースは、Amazon ECS コンソールを使用してクリーンアップします。Amazon EC2 インスタンス、Elastic Load Balancing ロードバランサー、Auto Scaling グループなど他のリソースは、Amazon EC2 コンソールで手動でクリーンアップするか、それを作成した CloudFormation スタックを削除することでクリーンアップする必要があります。

1. ナビゲーションペインで [Clusters] (クラスター) を選択します。
2. [クラスター] ページで、このチュートリアル用に作成したクラスターを選択します。
3. [サービス] タブを選択します。
4. サービス > [削除] の順に選択します。
5. 確認プロンプトで、delete (削除) と入力し、[Delete] (削除) を選択します。または、サービスを削除する前に、ユーザーに代わって Amazon ECS がサービスをスケールダウンする `Force delete` オプションを使用することもできます。

   サービスが削除されるまでお待ちください。
6. [Delete Cluster] を選択します。確認プロンプトで、delete ***`cluster-name`***と入力し、[Delete] (削除) を選択します。クラスターを削除すると、Auto Scaling グループ、VPC、ロードバランサーなどを含むクラスターで作成された関連リソースがクリーンアップされます。

Fargate 用の Amazon ECS Windows タスクを作成する方法について説明します。

Amazon ECS が AWS Fargate をサポートするリージョンで、タスクに Fargate を使用して、Amazon ECS on AWS Fargate の利用を開始します。

AWS Fargateで Amazon ECSの使用を開始するには、以下のステップを完了します。

前提条件

開始する前に、Amazon ECS を使用するようにセットアップする のステップを完了し、IAM ユーザーに AdministratorAccess IAM ポリシー例で指定されているアクセス許可があることを確認します。

コンソールは、Fargate タスクに必要なタスク実行 IAM ロールを自動で作成しようとします。コンソールがこの IAM ロールを作成できるようにするには、次のいずれかがtrueである必要があります。

- ユーザーが管理者権限を持っていることが必要です。詳細については、「Amazon ECS を使用するようにセットアップする」を参照してください。
- ユーザーがサービスロールを作成するための IAM 権限を持っていることが必要です。詳細については、「AWS サービスにアクセス許可を委任するロールの作成」を参照してください。
- 管理者権限を持つユーザーは、タスク実行ロールを手動で作成することにより、使用するアカウントで有効にできます。詳細については、「Amazon ECS タスク実行IAM ロール」を参照してください。

> **Important**
>
> タスク定義を使用してサービスを作成するときに選択するセキュリティグループには、インバウンドトラフィック用にポート 80 が開いている必要があります。セキュリティグループに次のインバウンドルールを追加します。セキュリティグループの作成方法については、「Amazon EC2 ユーザーガイド」の「Amazon EC2 インスタンス用のセキュリティグループの作成」を参照してください。
>
> - [Type]: HTTP
> - [Protocol]: TCP
> - ポート範囲: 80
> - ソース: 任意の場所 (0.0.0.0/0)

ステップ 1: クラスターを作成する

デフォルトの VPCを使用する [ウィンドウズ] という名前の新しいクラスターを作成できます。

AWS マネジメントコンソール を使用してクラスターを作成するには

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションバーから、使用するリージョンを選択します。
3. ナビゲーションペインで [Clusters] (クラスター) を選択します。
4. [Clusters] (クラスター) ページで、[Create Cluster] (クラスターの作成) を選択します。
5. [Cluster configuration] (クラスター設定) の [Cluster name] (クラスター名) に「windows」と入力します。
6. (オプション) Container Insights を有効にするには、[Monitoring] (モニタリング) を展開し、[Use Container Insights] (Container Insights を使用する) をオンにします。
7. (オプション) クラスターを識別しやすくするには、[Tags] (タグ) を展開し、タグを設定します。

   [タグの追加] [タグの追加] を選択して、以下を実行します。

   - [キー] にはキー名を入力します。
   - [値] にキー値を入力します。

   [タグを削除] タグのキーと値の右側にある [削除] を選択します。
8. [作成] を選択します。

ステップ 2: Windows タスク定義を登録する

Amazon ECS クラスターで Windows コンテナを実行する前に、タスク定義を登録する必要があります。次のタスク定義の例では、`mcr.microsoft.com/windows/servercore/iis` コンテナイメージを使用してコンテナインスタンスのポート 8080 でシンプルなウェブページを表示します。

AWS マネジメントコンソール を使用してサンプルタスク定義を登録するには

1. ナビゲーションペインで、タスク定義 を選択します。
2. [Create new task definition] (新しいタスク定義の作成)、[Create new task definition with JSON] (JSON で新しいタスク定義を作成) の順に選択します。
3. 以下のタスク定義の例をコピーしてボックスに貼り付け、[Save (保存)] を選択します。

```
{
    "containerDefinitions": [
        {
```

```
            "command": ["New-Item -Path C:\\inetpub\\wwwroot\\index.html
 -Type file -Value '<html> <head> <title>Amazon ECS Sample App</title>
 <style>body {margin-top: 40px; background-color: #333;} </style> </head><body>
 <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1>
 <h2>Congratulations!</h2> <p>Your application is now running on a container in
 Amazon ECS.</p>'; C:\\ServiceMonitor.exe w3svc"],
            "entryPoint": [
                "powershell",
                "-Command"
            ],
            "essential": true,
            "cpu": 2048,
            "memory": 4096,
            "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-
ltsc2019",
            "name": "sample_windows_app",
            "portMappings": [
                {
                    "hostPort": 80,
                    "containerPort": 80,
                    "protocol": "tcp"
                }
            ]
        }
    ],
    "memory": "4096",
    "cpu": "2048",
    "networkMode": "awsvpc",
    "family": "windows-simple-iis-2019-core",
    "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
    "runtimePlatform": {"operatingSystemFamily": "WINDOWS_SERVER_2019_CORE"},
    "requiresCompatibilities": ["FARGATE"]
}
```

4. 情報を確認し、[Create] (作成) を選択します。

ステップ 3: タスク定義を使用してサービスを作成する

タスク定義を登録したら、それを使用してクラスターにタスクを配置できます。次の手順では、タスク定義を使用してサービスを作成し、クラスターに 1 つのタスクを配置します。

コンソールを使用してタスク定義からサービスを作成するには

1. ナビゲーションペインで、[Clusters] (クラスター) を選択してから、ステップ 1: クラスターを作成する で作成したクラスターを選択します。
2. [Services] (サービス) タブから、[Create] (作成) を選択します。
3. [Deployment configure] (デプロイ設定) で、アプリケーションのデプロイ方法を指定します。
   a. [Task definition] (タスク定義) で、ステップ 2: Windows タスク定義を登録する で作成したタスク定義を選択します。
   b. [Service name] (サービス名) でサービスの名前を入力します。
   c. [Desired tasks] (必要なタスク) に、1と入力します。
4. [ネットワーク] で、セキュリティグループを作成するか、既存のグループを選択できます。使用するセキュリティグループに、前提条件 に一覧表示されているインバウンドルールがあることを確認してください。
5. [作成] を選択します。

ステップ 4: サービスを表示する

サービスによってクラスターでタスクが起動されたら、サービスを表示し、ブラウザで IIS テストページを開いてコンテナが実行中であることを確認できます。

> **Note**
>
> コンテナインスタンスで Windows コンテナベースレイヤーをダウンロードして抽出するまでに最大 15 分かかる場合があります。

サービスを表示するには

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。
3. サービスを実行したクラスターを選択します。
4. [サービス] タブの [サービス名] で、ステップ 3: タスク定義を使用してサービスを作成する に作成したサービスを選択します。
5. [タスク] タブ > サービス内のタスクの順に選択します。
6. [タスク] ページの [設定] セクションの [パブリック IP] で、[オープンアドレス] を選択します。

ステップ 5: クリーンアップ

Amazon ECS クラスターの使用が終了したら、使用していないリソースに対する料金が発生しないよう、それに関連付けられたリソースをクリーンアップする必要があります。

タスク、サービス、クラスター、コンテナインスタンスなど、一部の Amazon ECS リソースは、Amazon ECS コンソールを使用してクリーンアップします。Amazon EC2 インスタンス、Elastic Load Balancing ロードバランサー、Auto Scaling グループなど他のリソースは、Amazon EC2 コンソールで手動でクリーンアップするか、それを作成した CloudFormation スタックを削除することでクリーンアップする必要があります。

1. ナビゲーションペインで [Clusters] (クラスター) を選択します。
2. [クラスター] ページで、このチュートリアル用に作成したクラスターを選択します。
3. [サービス] タブを選択します。
4. サービス > [削除] の順に選択します。
5. 確認プロンプトで、delete (削除) と入力し、[Delete] (削除) を選択します。

   サービスが削除されるまでお待ちください。
6. [Delete Cluster] を選択します。確認プロンプトで、delete ***cluster-name***と入力し、[Delete] (削除) を選択します。クラスターを削除すると、Auto Scaling グループ、VPC、ロードバランサーなどを含むクラスターで作成された関連リソースがクリーンアップされます。

EC2 用の Amazon ECS Windows タスクを作成する方法について説明します。

EC2 を使用して Amazon ECS を開始するには、タスク定義を登録し、クラスターを作成し、コンソールにサービスを作成します。

EC2 起動タイプを使用して、Amazon ECS を開始するには、以下の手順を完了します。

前提条件

開始する前に、Amazon ECS を使用するようにセットアップする のステップを完了し、IAM ユーザーに `AdministratorAccess` IAM ポリシー例で指定されているアクセス許可があることを確認します。

コンソールは、Fargate タスクに必要なタスク実行 IAM ロールを自動で作成しようとします。コンソールがこの IAM ロールを作成できるようにするには、次のいずれかがtrueである必要があります。

- ユーザーが管理者権限を持っていることが必要です。詳細については、「Amazon ECS を使用するようにセットアップする」を参照してください。
- ユーザーがサービスロールを作成するための IAM 権限を持っていることが必要です。詳細については、「AWS サービスにアクセス許可を委任するロールの作成」を参照してください。
- 管理者権限を持つユーザーは、タスク実行ロールを手動で作成することにより、使用するアカウントで有効にできます。詳細については、「Amazon ECS タスク実行IAM ロール」を参照してください。

> **Important**
>
> タスク定義を使用してサービスを作成するときに選択するセキュリティグループには、インバウンドトラフィック用にポート 80 が開いている必要があります。セキュリティグループに次のインバウンドルールを追加します。セキュリティグループの作成方法については、「Amazon EC2 ユーザーガイド」の「Amazon EC2 インスタンス用のセキュリティグループの作成」を参照してください。
>
> - [Type]: HTTP
> - [Protocol]: TCP
> - ポート範囲: 80
> - ソース: 任意の場所 (`0.0.0.0/0`)

ステップ 1: クラスターを作成する

Amazon ECS クラスターはタスク、サービス、およびコンテナインスタンスの論理グループです。

以下では、1 つの Amazon EC2 インスタンスが登録されたクラスターを作成し、そこでタスクを実行できるようにするステップについて説明します。特定のフィールドが示されていない場合は、コンソールのデフォルト値のままにします。

新しいクラスターを作成するには (Amazon ECS コンソール)

開始する前に、適切な IAM アクセス許可を割り当ててください。詳細については、「the section called “Amazon ECS クラスターの例”」を参照してください。

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションバーから、使用するリージョンを選択します。
3. ナビゲーションペインで [Clusters] (クラスター) を選択します。
4. [Clusters] (クラスター) ページで、[Create Cluster] (クラスターの作成) を選択します。
5. [Cluster configuration] (クラスター設定) で、[Cluster name] (クラスター名) に一意の名前を入力します。

   名前には、最大 255 文字 (大文字と小文字)、数字、およびハイフンを含めることができます。
6. (オプション) タスクとサービスが起動する VPC とサブネットを変更するには、[Networking] (ネットワーク) で、次のいずれかのオペレーションを実行します。

   - サブネットを削除するには、[Subnets] (サブネット) で、削除するサブネットごとに [X] を選択します。
   - [デフォルト] VPC 以外の VPC に変更するには、[VPC] で既存の [VPC] を選択し、[サブネット] で各サブネットを選択します。
7. クラスターに Amazon EC2 インスタンスを追加するには、[インフラストラクチャー] を展開して [Amazon EC2 インスタンス] を選択します。次に、キャパシティープロバイダーとして機能する Auto Scaling グループを設定します。

   a. 既存の Auto Scaling グループを使用するには、[Auto Scaling group (ASG)] (Auto Scaling グループ) からグループを選択します。

   b. Auto Scaling グループを作成するには、Auto Scaling group(ASG) (Auto Scaling グループ) から、[Create new group] (新しいグループの作成) を選択し、グループに関する以下の詳細情報を入力します。

      - Operating system/Architecture (オペレーティングシステム/アーキテクチャ) を使用する場合、Auto Scaling グループインスタンスの Amazon ECS 最適化 AMI を選択します。
      - [EC2 instance type] (EC2 インスタンスタイプ) を使用する場合、ワークロードのインスタンスタイプを選択します。さまざまなインスタンスタイプに関する詳細については、Amazon EC2 インスタンスを参照してください。

マネージドスケーリングは、Auto Scaling グループが同じインスタンスタイプまたは類似のインスタンスタイプを使用している場合に最適です。

- [SSH key pair] (SSH キーペア) を使用する場合、インスタンスに接続する際に ID を証明するペアを選択してください。
- [Capacity] (キャパシティー) には、Auto Scaling グループで起動するインスタンスの最小数と最大数を入力します。Amazon EC2 instances が AWS リソースに存在する間、コストが発生します。詳細については、Amazon EC2 の料金表 を参照してください。

8. (オプション) Container Insights を有効にするには、[Monitoring] (モニタリング) を展開し、[Use Container Insights] (Container Insights を使用する) をオンにします。
9. (オプション) クラスタータグを管理するには、[Tags] (タグ) を展開し、次のいずれかのオペレーションを実行します。

   [タグの追加] [タグの追加] を選択して、以下を実行します。

   - [キー] にはキー名を入力します。
   - [値] にキー値を入力します。

   [タグを削除] タグのキーと値の右側にある [削除] を選択します。
10. [作成] を選択します。

ステップ 2: タスク定義を登録する

AWS マネジメントコンソール を使用してサンプルタスク定義を登録するには

1. ナビゲーションペインで、[タスク定義] を選択します。
2. [Create new task definition] (新しいタスク定義の作成)、[Create new task definition with JSON] (JSON で新しいタスク定義を作成) の順に選択します。
3. 以下のタスク定義の例をコピーしてボックスに貼り付け、[保存] を選択します。

```
{
    "containerDefinitions": [
        {
            "command": ["New-Item -Path C:\\inetpub\\wwwroot\\index.html
 -Type file -Value '<html> <head> <title>Amazon ECS Sample App</title>
 <style>body {margin-top: 40px; background-color: #333;} </style> </head><body>
 <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1>
```

```
 <h2>Congratulations!</h2> <p>Your application is now running on a container in
 Amazon ECS.</p>'; C:\\ServiceMonitor.exe w3svc"],
            "entryPoint": [
                "powershell",
                "-Command"
            ],
            "essential": true,
            "cpu": 2048,
            "memory": 4096,
            "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-
ltsc2019",
            "name": "sample_windows_app",
            "portMappings": [
                {
                    "hostPort": 443,
                    "containerPort": 80,
                    "protocol": "tcp"
                }
            ]
        }
    ],
    "memory": "4096",
    "cpu": "2048",
    "family": "windows-simple-iis-2019-core",
    "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
    "runtimePlatform": {"operatingSystemFamily": "WINDOWS_SERVER_2019_CORE"},
    "requiresCompatibilities": ["EC2"]
}
```

4. 情報を確認し、[Create] (作成) を選択します。

ステップ 3: サービスを作成する

Amazon ECS サービスを使用すると、Amazon ECS クラスター内で、タスク定義の指定した数のインスタンスを同時に実行して維持するのに役立ちます。タスクが何らかの理由で失敗または停止した場合、 Amazon ECS サービススケジューラは、タスク定義の別のインスタンスを起動してそれに置き換え、サービスで必要な数のタスクを維持します。サービスの詳細については、「Amazon ECS サービス」を参照してください。

サービスを作成するには

1. ナビゲーションペインで [Clusters] (クラスター) を選択します。

2. ステップ 1: クラスターを作成する で作成したクラスターを選択します。
3. [Services (サービス)] タブで、[Create (作成)] を選択します。
4. [Environment] (環境) セクションで、以下を実行します。
   a. [Compute options] (コンピューティングオプション) では、[Launch type] (起動タイプ) を選択します。
   b. [起動タイプ] で [EC2] をクリックします。
5. [Deployment configuration] (デプロイ設定) セクションで、以下を実行します。
   a. [Family] (ファミリー) で、ステップ 2: タスク定義を登録する で作成したタスク定義を選択します。
   b. [Service name] (サービス名) でサービスの名前を入力します。
   c. [Desired tasks] (必要なタスク) に、1と入力します。
6. オプションを確認し [作成] を選択します。
7. [View service (サービスの表示)] を選択して、サービスを確認します。

ステップ 4: サービスを表示する

このサービスはウェブベースのアプリケーションであるため、ウェブブラウザでコンテナを表示できます。

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。
3. サービスを実行したクラスターを選択します。
4. [サービス] タブの [サービス名] で、ステップ 3: サービスを作成する に作成したサービスを選択します。
5. [タスク] タブ > サービス内のタスクの順に選択します。
6. [タスク] ページの [設定] セクションの [パブリック IP] で、[オープンアドレス] を選択します。以下のスクリーンショットは想定される出力です。



ステップ 5: クリーンアップ

Amazon ECS クラスターの使用が終了したら、使用していないリソースに対する料金が発生しないよう、それに関連付けられたリソースをクリーンアップする必要があります。

タスク、サービス、クラスター、コンテナインスタンスなど、一部の Amazon ECS リソースは、Amazon ECS コンソールを使用してクリーンアップします。Amazon EC2 インスタンス、Elastic Load Balancing ロードバランサー、Auto Scaling グループなど他のリソースは、Amazon EC2 コンソールで手動でクリーンアップするか、それを作成した CloudFormation スタックを削除することでクリーンアップする必要があります。

1. ナビゲーションペインで [Clusters] (クラスター) を選択します。
2. [クラスター] ページで、このチュートリアル用に作成したクラスターを選択します。
3. [サービス] タブを選択します。
4. サービス > [削除] の順に選択します。
5. 確認プロンプトで、delete (削除) と入力し、[Delete] (削除) を選択します。

   サービスが削除されるまでお待ちください。
6. [Delete Cluster] を選択します。確認プロンプトで、delete ***cluster-name***と入力し、[Delete] (削除) を選択します。クラスターを削除すると、Auto Scaling グループ、VPC、ロードバランサーなどを含むクラスターで作成された関連リソースがクリーンアップされます。

AWS CDK を使用した Amazon ECS リソースの作成

AWS Cloud Development Kit (AWS CDK) は、選択したプログラミング言語を使用して、AWS クラウドインフラストラクチャの定義に使用できる、Infrastructure as Code (IAC) フレームワークです。独自のクラウドインフラストラクチャを定義するには、最初に、(CDK でサポートされている言語のいずれかで) 1 つ以上のスタックが含まれるアプリケーションを記述します。次に、それを CloudFormation テンプレートに合成して、AWS アカウント にリソースをデプロイします。このトピックのステップに従って、Amazon Elastic Container Service (Amazon ECS) と Fargate 上の AWS CDK で、コンテナ化されたウェブサーバーをデプロイしてください。

CDK に含まれる AWS 構成ライブラリは、AWS のサービス が提供するリソースのモデル化に使用できるモジュールを提供します。一般的なサービスでは、ライブラリが、スマートなデフォルトとベストプラクティスを持つ厳選された構成を提供します。これらのモジュールの 1 つである `aws-ecs-patterns` は、コンテナ化されたサービスと必要なすべてのサポートリソースをほんの数行のコードで定義するために使用できる、高レベルの抽象化を提供します。

このトピックでは、`ApplicationLoadBalancedFargateService` 構成を使用します。この構成は、Application Load Balancer の背後にある Fargate に Amazon ECS サービスをデプロイします。この `aws-ecs-patterns` モジュールには、Network Load Balancer を使用したり、Amazon EC2 で実行したりする構成も含まれています。

このタスクを開始する前に、AWS CDK 開発環境をセットアップし、次のコマンドを実行して AWS CDK をインストールします。AWS CDK 開発環境のセットアップ方法については、「Getting Started With the AWS CDK - Prerequisites」(の使用を開始する - 前提条件) を参照してください。

```
npm install -g aws-cdk
```

> **Note**
>
> これらの手順は、AWS CDK v2 を使用していることを前提としています。

トピック

- ステップ 1:AWS CDK プロジェクトを設定する
- ステップ 2: AWS CDK を使用して、Fargate 上のコンテナ化されたウェブサーバーを定義する
- ステップ 3: ウェブサーバーをテストする
- ステップ 4: クリーンアップする

- 次のステップ

ステップ 1:AWS CDK プロジェクトを設定する

新しいAWS CDKアプリケーションのディレクトリを作成し、プロジェクトを初期化します。

TypeScript

```
mkdir hello-ecs
cd hello-ecs
cdk init --language typescript
```

JavaScript

```
mkdir hello-ecs
cd hello-ecs
cdk init --language javascript
```

Python

```
mkdir hello-ecs
cd hello-ecs
cdk init --language python
```

プロジェクトが開始されたら、プロジェクトの仮想環境をアクティブにし、AWS CDK のベースラインの依存関係をインストールします。

```
source .venv/bin/activate
python -m pip install -r requirements.txt
```

Java

```
mkdir hello-ecs
cd hello-ecs
cdk init --language java
```

この Maven プロジェクトを Java IDE にインポートします。例えば Eclipse では、[File] (ファイル) > [Import] (インポート) > [Maven] > [Existing Maven Projects] (既存の Maven プロジェクト) を使用します。

C#

```
mkdir hello-ecs
cd hello-ecs
cdk init --language csharp
```

Go

```
mkdir hello-ecs
cd hello-ecs
cdk init --language go
```

> **Note**
>
> AWS CDKアプリケーションテンプレートは、プロジェクトディレクトリの名前を使用し、ソースファイルとクラスの名前を生成します。この例では、ディレクトリ名は `hello-ecs` です。別のプロジェクトディレクトリ名を使用する場合、アプリケーションはこれらの指示と一致しません。

AWS CDK v2 は、`aws-cdk-lib` と呼ばれる単一パッケージですべての AWS のサービス の安定した構成を含んでいます。このパッケージは、プロジェクト開始時に依存関係としてインストールされます。特定のプログラミング言語で作業する場合、パッケージはプロジェクトを初めて構築するときにインストールされます。このトピックでは、Amazon ECS の使用時に高レベルの抽象化を提供する、Amazon ECS Patterns 構成の使用方法について説明します。このモジュールは、Amazon ECS 構成およびその他の構成に依存し、Amazon ECS アプリケーションに必要なリソースをプロビジョニングします。

これらのライブラリを CDK アプリケーションにインポートする際に使用する名前は、使用するプログラミング言語によって若干異なります。参考として、サポートされている各 CDK プログラミング言語で使用される名前を以下に示します。

TypeScript

```
aws-cdk-lib/aws-ecs
aws-cdk-lib/aws-ecs-patterns
```

JavaScript

```
aws-cdk-lib/aws-ecs
aws-cdk-lib/aws-ecs-patterns
```

Python

```
aws_cdk.aws_ecs
aws_cdk.aws_ecs_patterns
```

Java

```
software.amazon.awscdk.services.ecs
software.amazon.awscdk.services.ecs.patterns
```

C#

```
Amazon.CDK.AWS.ECS
Amazon.CDK.AWS.ECS.Patterns
```

Go

```
github.com/aws/aws-cdk-go/awscdk/v2/awsecs
github.com/aws/aws-cdk-go/awscdk/v2/awsecspatterns
```

ステップ 2: AWS CDK を使用して、Fargate 上のコンテナ化されたウェブサーバーを定義する

コンテナイメージ amazon-ecs-sample を使用します。このイメージには、nginx で実行される PHP ウェブアプリケーションが含まれています。

作成した AWS CDK プロジェクトで、スタック定義を含むファイルを次の例のいずれかと同様に編集します。

> **Note**
>
> スタックはデプロイの単位です。すべてのリソースがスタック内にある必要があり、スタック内のすべてのリソースは同時にデプロイされます。リソースのデプロイに失敗すると、既にデプロイされている他のリソースはロールバックされます。AWS CDK アプリケーション

には複数のスタックを含めることができ、あるスタック内のリソースは別のスタック内のリソースを参照できます。

TypeScript

lib/hello-ecs-stack.ts を更新すると次のようになります。

```
import * as cdk from 'aws-cdk-lib';
import { Construct } from 'constructs';
import * as ecs from 'aws-cdk-lib/aws-ecs';
import * as ecsp from 'aws-cdk-lib/aws-ecs-patterns';

export class HelloEcsStack extends cdk.Stack {
  constructor(scope: Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    new ecsp.ApplicationLoadBalancedFargateService(this, 'MyWebServer', {
      taskImageOptions: {
        image: ecs.ContainerImage.fromRegistry('public.ecr.aws/ecs-sample-image/
amazon-ecs-sample:latest'),
      },
      publicLoadBalancer: true
    });
  }
}
```

JavaScript

lib/hello-ecs-stack.js を更新すると次のようになります。

```
const cdk = require('aws-cdk-lib');
const { Construct } = require('constructs');
const ecs = require('aws-cdk-lib/aws-ecs');
const ecsp = require('aws-cdk-lib/aws-ecs-patterns');

class HelloEcsStack extends cdk.Stack {
  constructor(scope = Construct, id = string, props = cdk.StackProps) {
    super(scope, id, props);

    new ecsp.ApplicationLoadBalancedFargateService(this, 'MyWebServer', {
      taskImageOptions: {
```

```
        image: ecs.ContainerImage.fromRegistry('amazon/amazon-ecs-sample'),
      },
      publicLoadBalancer: true
    });
  }
}

module.exports = { HelloEcsStack }
```

Python

hello-ecs/hello_ecs_stack.py を更新すると次のようになります。

```
import aws_cdk as cdk
from constructs import Construct

import aws_cdk.aws_ecs as ecs
import aws_cdk.aws_ecs_patterns as ecsp

class HelloEcsStack(cdk.Stack):

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        ecsp.ApplicationLoadBalancedFargateService(self, "MyWebServer",
            task_image_options=ecsp.ApplicationLoadBalancedTaskImageOptions(
                image=ecs.ContainerImage.from_registry("amazon/amazon-ecs-sample")),
            public_load_balancer=True
        )
```

Java

src/main/java/com.myorg/HelloEcsStack.java を更新すると次のようになります。

```
package com.myorg;

import software.constructs.Construct;
import software.amazon.awscdk.Stack;
import software.amazon.awscdk.StackProps;

import software.amazon.awscdk.services.ecs.ContainerImage;
import
 software.amazon.awscdk.services.ecs.patterns.ApplicationLoadBalancedFargateService;
```

```
import
 software.amazon.awscdk.services.ecs.patterns.ApplicationLoadBalancedTaskImageOptions;

public class HelloEcsStack extends Stack {
    public HelloEcsStack(final Construct scope, final String id) {
        this(scope, id, null);
    }

    public HelloEcsStack(final Construct scope, final String id, final StackProps
 props) {
        super(scope, id, props);

        ApplicationLoadBalancedFargateService.Builder.create(this, "MyWebServer")
         .taskImageOptions(ApplicationLoadBalancedTaskImageOptions.builder()
           .image(ContainerImage.fromRegistry("amazon/amazon-ecs-sample"))
           .build())
         .publicLoadBalancer(true)
         .build();
    }
}
```

C#

src/HelloEcs/HelloEcsStack.cs を更新すると次のようになります。

```
using Amazon.CDK;
using Constructs;
using Amazon.CDK.AWS.ECS;
using Amazon.CDK.AWS.ECS.Patterns;
namespace HelloEcs
{
    public class HelloEcsStack : Stack
    {
        internal HelloEcsStack(Construct scope, string id, IStackProps props =
 null) : base(scope, id, props)
        {
            new ApplicationLoadBalancedFargateService(this, "MyWebServer",
                new ApplicationLoadBalancedFargateServiceProps
                {
                    TaskImageOptions = new ApplicationLoadBalancedTaskImageOptions
                    {
                        Image = ContainerImage.FromRegistry("amazon/amazon-ecs-
sample")
                    },
```

```
                    PublicLoadBalancer = true
                });
        }
    }
}
```

Go

hello-ecs.go を更新すると次のようになります。

```
package main

import (
 "github.com/aws/aws-cdk-go/awscdk/v2"
 // "github.com/aws/aws-cdk-go/awscdk/v2/awssqs"
 "github.com/aws/aws-cdk-go/awscdk/v2/awsecs"
 "github.com/aws/aws-cdk-go/awscdk/v2/awsecspatterns"
 "github.com/aws/constructs-go/constructs/v10"
 "github.com/aws/jsii-runtime-go"
)

type HelloEcsStackProps struct {
 awscdk.StackProps
}

func NewHelloEcsStack(scope constructs.Construct, id string, props
 *HelloEcsStackProps) awscdk.Stack {
 var sprops awscdk.StackProps
 if props != nil {
  sprops = props.StackProps
 }
 stack := awscdk.NewStack(scope, &id, &sprops)

 // The code that defines your stack goes here

 // example resource
 // queue := awssqs.NewQueue(stack, jsii.String("HelloEcsQueue"),
 &awssqs.QueueProps{
 //  VisibilityTimeout: awscdk.Duration_Seconds(jsii.Number(300)),
 // })
 res := awsecspatterns.NewApplicationLoadBalancedFargateService(stack,
 jsii.String("MyWebServer"),
  &awsecspatterns.ApplicationLoadBalancedFargateServiceProps{
   TaskImageOptions: &awsecspatterns.ApplicationLoadBalancedTaskImageOptions{
```

```
    Image: awsecs.ContainerImage_FromRegistry(jsii.String("amazon/amazon-ecs-
sample"), &awsecs.RepositoryImageProps{}),
   },
  },
 )
 awscdk.NewCfnOutput(stack, jsii.String("LoadBalancerDNS"),
 &awscdk.CfnOutputProps{Value: res.LoadBalancer().LoadBalancerDnsName()})

 return stack
}

func main() {
 defer jsii.Close()

 app := awscdk.NewApp(nil)

 NewHelloEcsStack(app, "HelloEcsStack", &HelloEcsStackProps{
  awscdk.StackProps{
   Env: env(),
  },
 })

 app.Synth(nil)
}

// env determines the AWS environment (account+region) in which our stack is to
// be deployed. For more information see: https://docs.aws.amazon.com/cdk/latest/
guide/environments.html
func env() *awscdk.Environment {
 // If unspecified, this stack will be "environment-agnostic".
 // Account/Region-dependent features and context lookups will not work, but a
 // single synthesized template can be deployed anywhere.
 //---------------------------------------------------------------------------
 return nil

 // Uncomment if you know exactly what account and region you want to deploy
 // the stack to. This is the recommendation for production stacks.
 //---------------------------------------------------------------------------
 // return &awscdk.Environment{
 //  Account: jsii.String("123456789012"),
 //  Region:  jsii.String("us-east-1"),
 // }

 // Uncomment to specialize this stack for the AWS Account and Region that are
```

```
 // implied by the current CLI configuration. This is recommended for dev
 // stacks.
 //---------------------------------------------------------------------------
 // return &awscdk.Environment{
 //  Account: jsii.String(os.Getenv("CDK_DEFAULT_ACCOUNT")),
 //  Region:  jsii.String(os.Getenv("CDK_DEFAULT_REGION")),
 // }
}
```

前述の短いスニペットには次のものが含まれます。

- サービスの論理名: `MyWebServer`。
- Amazon ECR Public Gallery から取得したコンテナイメージ: `amazon/amazon-ecs-sample`。
- ロードバランサーが公開アドレスを持ち、インターネットからアクセスできる事実といった、その他の関連情報。

AWS CDK は、次のリソースを含む、ウェブサーバーのデプロイに必要なすべてのリソースを作成します。この例では、これらのリソースは省略されています。

- Amazon ECS クラスター
- Amazon VPC と Amazon EC2 インスタンス
- Auto Scaling グループ
- Application Load Balancer
- IAM ロールとポリシー

自動プロビジョニングされたリソースの一部は、スタックで定義されているすべての Amazon ECS サービスにより共有されます。

ソースファイルを保存し、アプリケーションのメインディレクトリで `cdk synth` コマンドを実行します。AWS CDK はアプリケーションを実行し、そのアプリケーションから CloudFormation テンプレートを合成して、テンプレートを表示します。テンプレートは約 600 行の YAML ファイルです。ファイルの先頭は以下のとおりです。テンプレートはこの例と異なる場合があります。

```
Resources:
  MyWebServerLB3B5FD3AB:
    Type: AWS::ElasticLoadBalancingV2::LoadBalancer
```

```
    Properties:
      LoadBalancerAttributes:
        - Key: deletion_protection.enabled
          Value: "false"
      Scheme: internet-facing
      SecurityGroups:
        - Fn::GetAtt:
            - MyWebServerLBSecurityGroup01B285AA
            - GroupId
      Subnets:
        - Ref: EcsDefaultClusterMnL3mNNYNVpcPublicSubnet1Subnet3C273B99
        - Ref: EcsDefaultClusterMnL3mNNYNVpcPublicSubnet2Subnet95FF715A
      Type: application
    DependsOn:
      - EcsDefaultClusterMnL3mNNYNVpcPublicSubnet1DefaultRouteFF4E2178
      - EcsDefaultClusterMnL3mNNYNVpcPublicSubnet2DefaultRouteB1375520
    Metadata:
      aws:cdk:path: HelloEcsStack/MyWebServer/LB/Resource
  MyWebServerLBSecurityGroup01B285AA:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Automatically created Security Group for ELB
 HelloEcsStackMyWebServerLB06757F57
      SecurityGroupIngress:
        - CidrIp: 0.0.0.0/0
          Description: Allow from anyone on port 80
          FromPort: 80
          IpProtocol: tcp
          ToPort: 80
      VpcId:
        Ref: EcsDefaultClusterMnL3mNNYNVpc7788A521
    Metadata:
      aws:cdk:path: HelloEcsStack/MyWebServer/LB/SecurityGroup/Resource
# and so on for another few hundred lines
```

AWS アカウント でサービスをデプロイするには、cdk deploy コマンドをアプリケーションのメインディレクトリで実行します。AWS CDK が生成した IAM ポリシーを、承認するよう求められます。

デプロイには数分かかり、その間に AWS CDK は複数のリソースを作成します。デプロイからの出力の末尾数行に、ロードバランサーの公開ホスト名と、新しいウェブサーバーの URL が含まれます。その内容は次のとおりです。

```
Outputs:
HelloEcsStack.MyWebServerLoadBalancerDNSXXXXXXX = Hello-MyWeb-ZZZZZZZZZZZZZ-
ZZZZZZZZZZ.us-west-2.elb.amazonaws.com
HelloEcsStack.MyWebServerServiceURLYYYYYYYY = http://Hello-MyWeb-ZZZZZZZZZZZZZ-
ZZZZZZZZZZ.us-west-2.elb.amazonaws.com
```

ステップ 3: ウェブサーバーをテストする

デプロイ出力から URL をコピーし、ウェブブラウザに貼り付けます。ウェブサーバーからの次のウェルカムメッセージが表示されます。



ステップ 4: クリーンアップする

ウェブサーバーの使用が終了したら、CDK を使用して `cdk destroy` コマンドをアプリケーションのメインディレクトリで実行し、サービスを終了します。これにより、将来的に想定外の請求が発生するのを防げます。

次のステップ

AWS CDK を使用して AWS インフラストラクチャを開発する方法の詳細については、「AWS CDK デベロッパーガイド」を参照してください。

選択した言語で AWS CDK アプリケーションを記述する方法については、以下を参照してください。

TypeScript

- TypeScript で AWS CDK を操作

JavaScript

JavaScript で AWS CDK を操作

Python

Python で AWS CDK を捜査

Java

Java で AWS CDK を操作

C#

C# で AWS CDK を操作

Go

Go で AWS CDK の使用

このトピックで使用する AWS 構成ライブラリモジュールの詳細は、以下の AWS CDK API リファレンスの概要を参照してください。

- aws-ecs
- aws-ecs-patterns

AWS Copilot コマンドラインインターフェイスを使用した Amazon ECS リソースの作成

⚠ AWS Copilot CLI のサポート終了通知

AWS Copilot CLI のサポートは、2026 年 6 月 12 日付けで終了します。この終了日以降、ツールには更新、セキュリティパッチ、またはテクニカルサポートが提供されなくなります。

- 2026 年 6 月 12 日以降に新機能や機能強化は提供されません
- セキュリティ更新やバグ修正は提供されません
- テクニカルサポートの対象外になります
- 既存のデプロイは引き続き機能しますが、継続的なメンテナンスは行われません

サポートが必要ですか? 詳しい移行ガイダンスについては、AWS サポートにお問い合わせください。または、こちらのブログ投稿を参照してください。

AWS Copilot コマンドラインインターフェイス (CLI) コマンドは、ローカル開発環境から、Amazon ECS での本番稼働対応のコンテナ化されたアプリケーションの構築、リリース、および運用を簡素化します。AWS Copilot CLI は、Infrastructure as Code 使用することから、ユーザーの代わりにプロビジョニングされた CI/CD パイプラインを作成することまで、最新のアプリケーションのベストプラクティスをサポートするデベロッパーワークフローと連携 Infrastructure as CodeInfrastructure as Code します。AWS Copilot CLI を毎日の開発の一部として使用し、AWS マネジメントコンソールの代替としてテストのサイクルを使用します。

AWS Copilot は現在 Linux、macOS、Windows システムをサポートしています。AWS Copilot CLI の最新バージョンの詳細については、「リリース」を参照してください。

> **Note**
>
> AWS Copilot CLI のソースコードは GitHub で入手できます。含める変更について、問題とプルリクエストを送信することをお勧めします。ただし、現在、Amazon Web Services では、AWS Copilot コードの変更されたコピーの実行をサポートしていません。AWS Copilot の問題は、「Gitter」 または「GitHub」 でご連絡ください。問題を開いたり、フィードバックを提供したり、バグを報告したりすることができます。
> Copilot 開発の詳細については、「What's the future of Copilot?」を参照してください。

トピック

- AWS Copilot CLI のインストール
- AWS Copilot CLI を使用してサンプル Amazon ECS アプリケーションをデプロイする

AWS Copilot CLI の追加ドキュメントは、AWS Copilot ウェブサイトで入手できます。

AWS Copilot CLI のインストール

⚠ AWS Copilot CLI のサポート終了通知

AWS Copilot CLI のサポートは、2026 年 6 月 12 日付けで終了します。この終了日以降、ツールには更新、セキュリティパッチ、またはテクニカルサポートが提供されなくなります。

- 2026 年 6 月 12 日以降に新機能や機能強化は提供されません
- セキュリティ更新やバグ修正は提供されません
- テクニカルサポートの対象外になります
- 既存のデプロイは引き続き機能しますが、継続的なメンテナンスは行われません

サポートが必要ですか? 詳しい移行ガイダンスについては、AWS サポートにお問い合わせください。または、こちらのブログ投稿を参照してください。

以下の手順を使用して、Homebrew を使用するか、バイナリを手動でダウンロードして、AWS Copilot CLI をインストールできます。

Homebrew の使用

次のコマンドは、Homebrew を使用して macOS または Linux システムに AWS Copilot CLIをインストールするために使用されます。インストールする前に、Homebrew をインストールする必要があります。詳細については、「Homebrew」を参照してください。

```
brew install aws/tap/copilot-cli
```

バイナリのダウンロード

Homebrew の代わりに、macOS、Windows、または Linux システムに手動で AWS Copilot CLI をインストールできます。ご使用のオペレーティングシステムに合わせて次のコマンドを使用して、バイナリをダウンロードします。macOS と Linux の例には、バイナリに実行アクセス許可を適用するコマンドが含まれているほか、インストールが機能していることを検証するためのヘルプメニューがリストされています。

macOS

macOS の場合:

```
sudo curl -Lo /usr/local/bin/copilot https://github.com/aws/copilot-cli/releases/
latest/download/copilot-darwin \
   && sudo chmod +x /usr/local/bin/copilot \
   && copilot --help
```

macOS ARM システムの場合:

```
sudo curl -Lo /usr/local/bin/copilot https://github.com/aws/copilot-cli/releases/
latest/download/copilot-darwin-arm64 \
   && sudo chmod +x /usr/local/bin/copilot \
   && copilot --help
```

Linux

Linux x86 (64 ビット) システムの場合:

```
sudo curl -Lo /usr/local/bin/copilot https://github.com/aws/copilot-cli/releases/
latest/download/copilot-linux \
   && sudo chmod +x /usr/local/bin/copilot \
   && copilot --help
```

Linux ARM システムの場合:

```
sudo curl -Lo /usr/local/bin/copilot https://github.com/aws/copilot-cli/releases/
latest/download/copilot-linux-arm64 \
   && sudo chmod +x /usr/local/bin/copilot \
   && copilot --help
```

Windows

PowerShell を使用して、次のコマンドを実行します。

```
New-Item -Path 'C:\copilot' -ItemType directory; `
  Invoke-WebRequest -OutFile 'C:\copilot\copilot.exe' https://github.com/aws/
copilot-cli/releases/latest/download/copilot-windows.exe
```

(任意) PGP 署名を使用した手動でインストールした AWS Copilot CLI の検証

AWS Copilot CLI の実行可能ファイルは、PGP 署名を使用して暗号で署名されます。PGP 署名を使用して、AWS Copilot CLI 実行可能ファイルの妥当性を検証できます。GnuPG ツールを使用してこの署名を検証するには、次のステップを行います。

1. GnuPG をダウンロードし、インストールします。詳細については、GnuPG のウェブサイトを参照してください。

   macOS

   Homebrew の使用をお勧めします。製品ウェブサイトの手順に従って、Homebrew をインストールします。詳細については、「Homebrew」を参照してください。Homebrew がインストールされたら、macOS 端末から次のコマンドを使用します。

   ```
   brew install gnupg
   ```

   Linux

   任意の Linux のパッケージマネージャーを使用して gpg をインストールします。

   Windows

   GnuPG ウェブサイトから Windows の簡易インストーラをダウンロードし、管理者としてインストールします。GnuPG をインストールしたら、管理者向け PowerShell を閉じてから再度開きます。

   詳細については、「GnuPG のダウンロード」を参照してください。
2. GnuPG パスが環境パスに追加されていることを確認します。

   macOS

   ```
   echo $PATH
   ```

   出力に GnuPG パスが表示されない場合は、次のコマンドを実行してパスに追加します。

   ```
   PATH=$PATH:<path to GnuPG executable files>
   ```

Linux

```
echo $PATH
```

出力に GnuPG パスが表示されない場合は、次のコマンドを実行してパスに追加します。

```
export PATH=$PATH:<path to GnuPG executable files>
```

Windows

```
Write-Output $Env:PATH
```

出力に GnuPG パスが表示されない場合は、次のコマンドを実行してパスに追加します。

```
$Env:PATH += ";<path to GnuPG executable files>"
```

3. ローカルプレーンテキストファイルを作成します。

macOS

ターミナルで、次のように入力します。

```
touch <public_key_filename.txt>
```

TextEdit でファイルを開きます。

Linux

gedit など、テキストエディタでテキストファイルを作成します。`public_key_filename.txt` として保存します。

Windows

Notepad など、テキストエディタでテキストファイルを作成します。`public_key_filename.txt` として保存します。

4. 次の Amazon ECS PGP パブリックキーの内容を追加し、ファイルを保存します。

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2
```

```
mQINBFq1SasBEADliGcT1NVJ1ydfN8DqebYYe9ne3dt6jqKFmKowLmm6LLGJe7HU
jGtqhCWRDkN+qPpHqdArRgDZAtn2pXY5fEipHgar4CP8QgRnRMO2fl74lmavr4Vg
7K/KH8VHlq2uRw32/B94XLEgRbGTMdWFdKuxoPCttBQaMj3LGn6Pe+6xVWRkChQu
BoQAhjBQ+bEm0kNy0LjNgjNlnL3UMAG56t8E3LANIgGgEnpNsB1UwfWluPoGZoTx
N+6pHBJrKIL/1v/ETU4FXpYw2zvhWNahxeNRnoYj3uycHkeliCrw4kj0+skizBgO
2K7oVX8Oc3j5+ZilhL/qDLXmUCb2az5cMM1mOoF8EKX5HaNuq1KfwJxqXE6NNIcO
lFTrT7QwD5fMNld3FanLgv/ZnIrsSaqJOL6zRSq8O4LN1OWBVbndExk2Kr+5kFxn
5lBPgfPgRj5hQ+KTHMa9Y8Z7yUc64BJiN6F9Nl7FJuSsfqbdkvRLsQRbcBG9qxX3
rJAEhieJzVMEUNl+EgeCkxj5xuSkNU7zw2c3hQZqEcrADLV+hvFJktOz9Gm6xzbq
lTnWWCz4xrIWtuEBA2qE+MlDheVd78a3gIsEaSTfQq0osYXaQbvlnSWOoc1y/5Zb
zizHTJIhLtUyls9WisP2s0emeHZicVMfW61EgPrJAiupgc7kyZvFt4YwfwARAQAB
tCRBbWF6b24gRUNTIDxlY3Mtc2VjdXJpdHlAYW1hem9uLmNvbT6JAhwEEAECAAYF
AlrjL0YACgkQHivRXs0TaQrg1g/+JppwPqHnlVPmv7lessB8I5UqZeD6p6uVpHd7
Bs3pcPp8BV7BdRbs3sPLt5bV1+rkqOlw+0gZ4Q/ue/YbWtOAt4qY0OcEo0HgcnaX
lsB827QIfZIVtGWMhuh94xzm/SJkvngml6KB3YJNnWP61A9qJ37/VbVVLzvcmazA
McWB4HUMNrhd0JgBCo0gIpqCbpJEvUc02Bjn23eEJsS9kC7OUAHyQkVnx4d9UzXF
4OoISF6hmQKIBoLnRrAlj5Qvs3GhvHQ0ThYq0Grk/KMJJX2CSqt7tWJ8gk1n3H3Y
SReRXJRnv7DsDDBwFgT6r5Q2HW1TBUvaoZy5hF6maD09nHcNnvBjqADzeT8Tr/Qu
bBCLzkNSYqqkpgtwv7seoD2P4n1giRvDAOEfMZpVkUr+C252IaH1HZFEz+TvBVQM
Y8OWWxmIJW+J6evjo3N1eO19UHv71jvoF8zljbI4bsL2c+QTJmOv7nRqzDQgCWyp
Id/v2dUVVTk1j9omuLBBwNJzQCB+72LcIzJhYmaP1HC4LcKQG+/f41exuItenatK
lEJQhYtyVXcBlh6Yn/wzNg2NWOwb3vqY/F7m6u9ixAwgtIMgPCDE4aJ86zrrXYFz
N2HqkTSQh77Z8KPKmyGopsmN/reMuilPdINb249nA0dzoN+nj+tTFOYCIaLaFyjs
Z0r1QAOJAjkEEwECACMFAlq1SasCGwMHCwkIBwMCAQYVCAIJCgsEFgIDAQIeAQIX
gAAKCRC86dmkLVF4T9iFEACEnkm1dNXsWUx34R3c0vamHrPxvfkyI1FlEUen8D1h
uX9xy6jCEROHWEp0rjGK4QDPgM93sWJ+s1UAKg214QRVzft0y9/DdR+twApA0fzy
uavIthGd6+03jAAo6udYDE+cZC3P7XBbDiYEWk4XAF9I1JjB8hTZUgvXBL046JhG
eM17+crgUyQeetkiOQemLbsbXQ40Bd9V7zf7XJraFd8VrwNUwNb+9KFtgAsc9rk+
YIT/PEf+YOPysgcxI4sTWghtyCulVnuGoskgDv4v73PALU0ieUrvvQVqWMRvhVx1
0X90J7cC1KOyhlEQQ1aFTgmQjmXexVTwIBm8LvysFK6YXM41KjOrlz3+6xBIm/qe
bFyLUnf4WoiuOplAaJhK9pRY+XEnGNxdtN4D26Kd0F+PLkm3Tr3Hy3b1Ok34FlGr
KVHUq1TZD7cvMnnNKEELTUcKX+1mV3an16nmAg/my1JSUt6BNK2rJpY1s/kkSGSE
XQ4zuF2IGCpvBFhYAlt5Un5zwqkwwQR3/n2kwAoDzonJcehDw/C/cGos5D0aIU7I
K2X2aTD3+pA7Mx3IMe2hqmYqRt9X42yF1PIEVRneBRJ3HDezAgJrNh0GQWRQkhIx
gz6/cTR+ekr5TptVszS9few2GpI5bCgBKBisZIssT89aw7mAKWut0Gcm4qM9/yK6
1bkCDQRatUmrARAAxNPvVwreJ2yAiFcUpdRlVhsuOgnxvs1QgsIw3H7+Pacr9Hpe
8uftYZqdC82KeSKhpHq7c8gMTMucIINtH25x9BCc73E33EjCL9Lqov1TL7+QkgHe
T+JIhZwdD8Mx2K+LVVVu/aWkNrfMuNwyDUciSI4D5QHa8T+F8fgN4OTpwYjirzel
5yoICMr9hVcbzDNv/ozKCxjx+XKgnFc3wrnDfJfntfDAT7ecwbUTL+viQKJ646s+
psiqXRYtVvYInEhLVrJ0aV6zHFoigE/Bils6/g7ru1Q6CEHqEw++APs5CcE8VzJu
WAGSVHZgun5Y9N4quR/M9Vm+IPMhTxrAg7rOvyRN9cAXfeSMf77I+XTifigNna8x
t/MOdjXr1fjF4pThEi5u6WsuRdFwjY2azEv3vevodTi4HoJReH6dFRa6y8c+UDgl
2iHiOKIpQqLbHEfQmHcDd2fix+AaJKMnPGNku9qCFEMbgSRJpXz6BfwnY1QuKE+I
R6jA0frUNt2jhiGG/F8RceXzohaaC/Cx7LUCUFWc0n7z32C9/Dtj7I1PMOacdZzz
```

```
bjJzRKO/ZDv+UN/c9dwAkllzAyPMwGBkUaY68EBstnIliW34aWm6IiHhxioVPKSp
VJfyiXPO0EXqujtHLAeChfjcns3I12YshT1dv2PafG53fp33ZdzeUgsBo+EAEQEA
AYkCHwQYAQIACQUCWrVJqwIbDAAKCRC86dmkLVF4T+ZdD/9x/8APzgNJF3o3STrF
jvnV1ycyhWYGAeBJiu7wjsNWwzMFOv15tLjB7AqeVxZn+WKDD/mIOQ45OZvnYZuy
X7DR0JszaH9wrYTxZLVruAu+t6UL0y/XQ4L1GZ9QR6+r+7t1Mvbfy7BlHbvX/gYt
Rwe/uwdibI0CagEzyX+2D3kTOlHO5XThbXaNf8AN8zha91Jt2Q2UR2X5T6JcwtMz
FBvZnl3LSmZyE0EQehS2iUurU4uWOpGppuqVnbi0jbCvCHKgDGrqZ0smKNAQng54
F365W3g8AfY48s8XQwzmcliowYX9bT8PZiEi0J4QmQh0aXkpqZyFefuWeOL2R94S
XKzr+gRh3BAULoqF+qK+IUMxTip9KTPNvYDpiC66yBiT6gFDji5Ca9pGpJXrC3xe
TXiKQ8DBWDhBPVPrruLIaenTtZEOsPc4I85yt5U9RoPTStcOr34s3w5yEaJagt6S
Gc5r9ysjkfH6+6rbi1ujxMgROSqtqr+RyB+V9A5/OgtNZc8llK6u4UoOCde8jUUW
vqWKvjJB/Kz3u4zaeNu2ZyyHaOqOuH+TETcW+jsY9IhbEzqN5yQYGi4pVmDkY5vu
lXbJnbqPKpRXgM9BecV9AMbPgbDq/5LnHJJXg+G8YQOgp4lR/hC1TEFdIp5wM8AK
CWsENyt2o1rjgMXiZOMF8A5oBLkCDQRatUuSARAAr77kj7j2QR2SZeOSlFBvV7oS
mFeSNnz9xZssqrsm6bTwSHM6YLDwc7Sdf2esDdyzONETwqrVCg+FxgL8hmo9hS4c
rR6tmrP0mOmptr+xLLsKcaP7ogIXsyZnrEAEsvW8PnfayoiPCdc3cMCR/lTnHFGA
7EuR/XLBmi7Qg9tByVYQ5Yj5wB9V4B2yeCt3XtzPqeLKvaxl7PNelaHGJQY/xo+m
V0bndxf9IY+4oFJ4blD32WqvyxESo7vW6WBh7oqv3Zbm0yQrr8a6mDBpqLkvWwNI
3kpJR974tg5o5LfDu1BeeyHWPSGm4U/G4JB+JIG1ADy+RmoWEt4BqTCZ/knnoGvw
D5sTCxbKdmuOmhGyTssoG+3OOcGYHV7pWYPhazKHMPm201xKCjH1RfzRULzGKjD+
yMLT1I3AXFmLmZJXikAOlvE3/wgMqCXscbycbLjLD/bXIuFWo3rzoezeXjgi/DJx
jKBAyBTYO5nMcth1O9oaFd9d0HbsOUDkIMnsgGBE766Piro6MHo0T0rXl07Tp4pI
rwuSOsc6XzCzdImj0Wc6axS/HeUKRXWdXJwno5awTwXKRJMXGfhCvSvbcbc2Wx+L
IKvmB7EB4K3fmjFFE67yolmiw2qRcUBfygtH3eL5XZU28MiCpue8Y8GKJoBAUyvf
KeM1rO8Jm3iRAc5a/D0AEQEAAYkEPgQYAQIACQUCWrVLkgIbAgIpCRC86dmkLVF4
T8FdIAQZAQIABgUCWrVLkgAKCRDePL1hra+LjtHYD/9MucxdFe6bXO1dQR4tKhhQ
P0LRqy6zlBY9ILCLowNdGZdqorogUiUymgn3VhEhVtxTOoHcN7qOuM01PNsRnOeS
EYjf8Xrb1clzkD6xULwmOclTb9bBxnBc/4PFvHAbZW3QzusaZniNgkuxt6BTfloS
Of4inq71kjmGK+TlzQ6mUMQUg228NUQC+a84EPqYyAeY1sgvgB7hJBhYL0QAxhcW
6m20Rd8iEc6HyzJ3yCOCsKip/nRWAbf0OvfHfRBp0+m0ZwnJM8cPRFjOqqzFpKH9
HpDmTrC4wKP1+TL52LyEqNh4yZitXmZNV7giSRIkk0eDSko+bFy6VbMzKUMkUJK3
D3eHFAMkujmbfJmSMTJOPGn5SB1HyjCZNx6bhIIbQyEUB9gKCmUFaqXKwKpF6rj0
iQXAJxLR/shZ5Rk96VxzOphUl7T90m/PnUEEPwq8KsBhnMRgxa0RFidDP+n9fgtv
HLmrOqX9zBCVXh0mdWYLrWvmzQFWzG7AoE55fkf8nAEPsalrCdtaNUBHRXA0OQxG
AHMOdJQQvBsmqMvuAdjkDWpFu5y0My5ddU+hiUzUyQLjL5Hhd5LOUDdewlZgIw1j
xrEAUzDKetnemM8GkHxDgg8koev5frmShJuce7vSjKpCNg3EIJSgqMOPFjJuLWtZ
vjHeDNbJy6uNL65ckJy6WhGjEADS2WAW1D6Tfekkc21SsIXk/LqEpLMR/0g5OUif
wcEN1rS9IJXBwIy8MelN9qr5KcKQLmfdfBNEyyceBhyVl0MDyHOKC+7PofMtkGBq
13QieRHv5GJ8LB3fclqHV8pwTTo3Bc8z2g0TjmUYAN/ixETdReDoKavWJYSE9yoM
aaJu279ioVTrwpECse0XkiRyKToTjwOb73CGkBZZpJyqux/rmCV/fp4ALdSW8zbz
FJVORaivhoWwzjpfQKhwcU9lABXi2UvVm14v0AfeI7oiJPSU1zM4fEny4oiIBXlR
zhFNih1UjIu82X16mTm3BwbIga/s1fnQRGzyhqUIMii+mWra23EwjChaxpvjjcUH
5ilLc5Zq781aCYRygYQw+hu5nFkOH1R+Z50Ubxjd/aqUfnGIAX7kPMD3Lof4KldD
Q8ppQriUvxVo+4nPV6rpTy/PyqCLWDjkguHpJsEFsMkwajrAz0QNSAU5CJ0G2Zu4
```

```
yxvYlumHCEl7nbFrm0vIiA75Sa8KnywTDsyZsu3XcOcf3g+g1xWTpjJqy2bYXlqz
9uDOWtArWHOis6bq8l9RE6xr1RBVXS6uqgQIZFBGyq66b0dIq4D2JdsUvgEMaHbc
e7tBfeB1CMBdA64e9Rq7bFR7Tvt8gasCZYlNr3lydh+dFHIEkH53HzQe6l88HEic
+0jVnLkCDQRa55wJARAAyLya2Lx6gyoWoJN1a6740q3o8e9d4KggQOfGMTCflmeq
ivuzgN+3DZHN+9ty2KxXMtn0mhHBerZdbNJyjMNT1gAgrhPNB4HtXBXum2wS57WK
DNmade914L7FWTPAWBG2Wn448OEHTqsClICXXWy9IICgclAEyIq0Yq5mAdTEgRJS
Z8t4GpwtDL9gNQyFXaWQmDmkAsCygQMvhAlmu9xOIzQG5CxSnZFk7zcuL60k14Z3
Cmt49k4T/7ZU8goWi8tt+rU78/IL3J/fF9+1civ1OwuUidgfPCSvOUW1JojsdCQA
L+RZJcoXq7lfOFj/eNjeOSstCTDPfTCL+kThE6E5neDtbQHBYkEX1BRiTedsV4+M
ucgiTrdQFWKf89G72xdv8ut9AYYQ2BbEYU+JAYhUH8rYYui2dHKJIgjNvJscuUWb
+QEqJIRleJRhrO+/CHgMs4fZAkWF1VFhKBkcKmEjLn1f7EJJUUW84ZhKXjO/AUPX
1CHsNjziRceuJCJYox1cwsoq6jTE50GiNzcIxTn9xUc0UMKFeggNAFys1K+TDTm3
Bzo8H5ucjCUEmUm9lhkGwqTZgOlRX5eqPX+JBoSaObqhgqCa5IPinKRa6MgoFPHK
6sYKqroYwBGgZm6Js5chpNchvJMs/3WXNOEVg0J3z3vP0DMhxqWm+r+n9zlW8qsA
EQEAAYkEPgQYAQgACQUCWuecCQIbAgIpCRC86dmkLVF4T8FdIAQZAQgABgUCWuec
CQAKCRBQ3szEcQ5hr+ykD/4tOLRHFHXuKUcxgGaubUcVtsFrwBKma1cYjqaPms8u
6Sk0wfGRI32G/GhOrp0Ts/MOkbObq6VLTh8N5Yc/53MEl8zQFw9Y5AmRoW4PZXER
ujs5s7p4oR7xHMihMjCCBn1bvrR+34YPfgzTcgLiOEFHYT8UTxwnGmXOvNkMM7md
xD3CV5q6VAte8WKBo/220II3fcQlc9r/oWX4kXXkb0v9hoGwKbDJ1tzqTPrp/xFt
yohqnvImpnlz+Q9zXmbrWYL9/g8VCmW/NN2gju2G3Lu/TlFUWIT4v/5OPK6TdeNb
VKJO4+S8bTayqSG9CML1S57KSgCo5HUhQWeSNHI+fpe5oX6FALPT9JLDce8OZz1i
cZZ0MELP37mOOQun0AlmHm/hVzf0f311PtbzcqWaE51tJvgUR/nZFo6Ta3O5Ezhs
3VlEJNQ1Ijf/6DH87SxvAoRIARCuZd0qxBcDK0avpFzUtbJd24lRA3WJpkEiMqKv
RDVZkE4b6TW61f0o+LaVfK6E8oLpixegS4fiqC16mFrOdyRk+RJJfIUyz0WTDVmt
g0U1CO1ezokMSqkJ7724pyjr2xf/r9/sC6aOJwB/lKgZkJfC6NqL7TlxVA31dUga
LEOvEJTTE4gl+tYtfsCDvALCtqL0jduSkUo+RXcBItmXhA+tShW0pbS2Rtx/ixua
KohVD/0R4QxiSwQmICNtm9mw9ydIl1yjYXX5a9x4wMJracNY/LBybJPFnZnT4dYR
z4XjqysDwvvYZByaWoIe3QxjX84V6MlI2IdAT/xImu8gbaCI8tmyfpIrLnPKiR9D
VFYfGBXuAX7+HgPPSFtrHQONCALxxzlbNpS+zxt9r0MiLgcLyspWxSdmoYGZ6nQP
RO5Nm/ZVS+u2imPCRzNUZEMa+dlE6kHx0rS0dPiuJ4O7NtPeYDKkoQtNagspsDvh
cK7CSqAiKMq06UBTxqlTSRkm62eOCtcs3p3OeHu5GRZF1uzTET0ZxYkaPgdrQknx
ozjP5mC7X+45lcCfmcVt94TFNL5HwEUVJpmOgmzILCI8yoDTWzloo+i+fPFsXX4f
kynhE83mSEcr5VHFYrTY3mQXGmNJ3bCLuc/jq7ysGq69xiKmTlUeXFm+aojcRO5i
zyShIRJZ0GZfuzDYFDbMV9amA/YQGygLw//zP5ju5SW26dNxlf3MdFQE5JJ86rn9
MgZ4gcpazHEVUsbZsgkLizRp9imUiH8ymLqAXnfRGlU/LpNSefnvDFTtEIRcpOHc
bhayG0bk51Bd4mioOXnIsKy4j63nJXA27x5EVVHQ1sYRN8Ny4Fdr2tMAmj2O+X+J
qX2yy/UX5nSPU492e2CdZ1UhoU0SRFY3bxKHKB7SDbVeav+K5g==
=Gi5D
-----END PGP PUBLIC KEY BLOCK-----
```

Amazon ECS PGP パブリックキーの詳細を参照用として以下に示します。

```
Key ID: BCE9D9A42D51784F
```

```
Type: RSA
Size: 4096/4096
Expires: Never
User ID: Amazon ECS
Key fingerprint: F34C 3DDA E729 26B0 79BE AEC6 BCE9 D9A4 2D51 784F
```

5. ターミナルで次のコマンドを使用して、Amazon ECS PGP パブリックキーを持ったファイルをインポートします。

```
gpg --import <public_key_filename.txt>
```

6. AWS Copilot CLI 署名をダウンロードします。署名は、ASCII でデタッチ済みの PGP 署名で、拡張子が .asc のファイルに保存されています。この署名ファイルには、対応する実行可能ファイルと同じ名前が付けられており、拡張子は .asc です。

   macOS

   macOS システムでは、次のコマンドを実行します。

   ```
   sudo curl -Lo copilot.asc https://github.com/aws/copilot-cli/releases/latest/download/copilot-darwin.asc
   ```

   Linux

   Linux システム x86 (64 ビット) の場合は、次のコマンドを実行します。

   ```
   sudo curl -Lo copilot.asc https://github.com/aws/copilot-cli/releases/latest/download/copilot-linux.asc
   ```

   Linux ARM システムの場合は、次のコマンドを実行します。

   ```
   sudo curl -Lo copilot.asc https://github.com/aws/copilot-cli/releases/latest/download/copilot-linux-arm64.asc
   ```

   Windows

   PowerShell を使用して、次のコマンドを実行します。

   ```
   Invoke-WebRequest -OutFile 'C:\copilot\copilot.asc' https://github.com/aws/copilot-cli/releases/latest/download/copilot-windows.exe.asc
   ```

7. 次のコマンドを使用して、署名を検証します。

   - macOS および Linux システムの場合:

     ```
     gpg --verify copilot.asc /usr/local/bin/copilot
     ```

   - Windows システムの場合:

     ```
     gpg --verify 'C:\copilot\copilot.asc' 'C:\copilot\copilot.exe'
     ```

   正常な出力:

   ```
   gpg: Signature made Tue Apr  3 13:29:30 2018 PDT
   gpg:                using RSA key DE3CBD61ADAF8B8E
   gpg: Good signature from "Amazon ECS <ecs-security@amazon.com>" [unknown]
   gpg: WARNING: This key is not certified with a trusted signature!
   gpg:          There is no indication that the signature belongs to the owner.
   Primary key fingerprint: F34C 3DDA E729 26B0 79BE  AEC6 BCE9 D9A4 2D51 784F
        Subkey fingerprint: EB3D F841 E2C9 212A 2BD4  2232 DE3C BD61 ADAF 8B8E
   ```

   > **Important**
   >
   > 出力に警告が表示されることがありますが、問題ありません。これは、個人 PGP キー (持っている場合) と Amazon ECS PGP キーの間に信頼チェーンがないために表示されます。詳細については、「信用の輪 (Web of Trust)」を参照してください。

8. Windows のインストールの場合は、PowerShell で次のコマンドを実行して、AWS Copilot ディレクトリをパスに追加します。

   ```
   $Env:PATH += ";<path to Copilot executable files>"
   ```

AWS Copilot CLI を使用してサンプル Amazon ECS アプリケーションをデプロイする

⚠ AWS Copilot CLI のサポート終了通知

AWS Copilot CLI のサポートは、2026 年 6 月 12 日付けで終了します。この終了日以降、ツールには更新、セキュリティパッチ、またはテクニカルサポートが提供されなくなります。

- 2026 年 6 月 12 日以降に新機能や機能強化は提供されません
- セキュリティ更新やバグ修正は提供されません
- テクニカルサポートの対象外になります
- 既存のデプロイは引き続き機能しますが、継続的なメンテナンスは行われません

サポートが必要ですか? 詳しい移行ガイダンスについては、AWS サポートにお問い合わせください。または、こちらのブログ投稿を参照してください。

AWS Copilot CLI をインストールしたら、以下の手順に従ってサンプルアプリケーションをデプロイし、デプロイを検証し、リソースをクリーンアップできます。

前提条件

開始する前に、以下の前提条件を満たしていることを確認してください。

- をインストールして設定しますAWS CLI 詳細については、「AWS コマンドラインインターフェイス」を参照してください。
- `aws configure` を実行して、AWSコパイロット CLI がアプリケーションとサービスの管理に使用するデフォルトプロファイルを設定します。
- Docker のインストールと実行 詳細については、「Docker の開始方法」を参照してください。

単一コマンドを使用したサンプル Amazon ECS アプリケーションのデプロイ

1. 次のコマンドを使用して、GitHub リポジトリからクローンされたサンプルウェブアプリケーションをデプロイします。AWS Copilot `init` とそのフラグの詳細については、AWS Copilot ドキュメントを参照してください。

```
git clone https://github.com/aws-samples/aws-copilot-sample-service.git demo-app &&
 \
cd demo-app &&                                  \
copilot init --app demo                         \
  --name api                                    \
  --type 'Load Balanced Web Service'            \
  --dockerfile './Dockerfile'                   \
  --port 80                                     \
  --tag  latest                                 \
  --deploy
```

2. デプロイが完了すると、AWS Copilot CLI はデプロイの検証に使用できる URL を返します。次のコマンドを使用して、アプリケーションのステータスを確認することもできます。

   - AWS コパイロットのアプリケーションを一覧を表示します。

     ```
     copilot app ls
     ```

   - アプリケーション内の環境およびサービスに関する情報を表示します。

     ```
     copilot app show
     ```

   - 環境に関する情報を表示します。

     ```
     copilot env ls
     ```

   - エンドポイント、キャパシティー、関連リソースなど、サービスに関する情報を表示します。

     ```
     copilot svc show
     ```

   - アプリケーション内のすべてのサービスのリスト。

     ```
     copilot svc ls
     ```

   - デプロイされたサービスのログを表示します。

     ```
     copilot svc logs
     ```

   - サービスのステータスを表示する

```
copilot svc status
```

3. このデモが終了したら、次のコマンドを実行して関連付けられたリソースをクリーンアップし、未使用のリソースに対する料金が発生しないようにしてください。

```
copilot app delete
```

AWS CloudFormation を用いた Amazon ECS の使用

Amazon ECS は、ユーザー定義テンプレートで AWS リソースをモデル化してセットアップするサービスである AWS CloudFormation と統合されています。CloudFormation が使用するテンプレートは、YAML もしくは JSON 形式のテキストファイルです。テンプレートとは、作成する AWS リソースの設計図のようなものです。テンプレートを作成して送信すると、CloudFormation はスタックを作成します。テンプレートで定義したリソースは、このスタックで管理します。リソースを作成、更新、または削除する場合は、そのリソースから作成されたスタックを作成、更新、または削除します。スタックの更新には、まず変更セットを作成する必要があります。変更セットは、変更を行う前にその変更に影響を受ける対象を示します。これにより、データベース名を変更するなどして、データベースを誤って削除することを防ぐことができます。テンプレート、スタック、変更セットの詳細については、「AWS CloudFormation ユーザーガイド」の「CloudFormation の仕組み」を参照してください。

CloudFormation を使用することで、リソースやインフラストラクチャの作成、管理に費やす時間を短縮できます。Amazon ECS クラスター、タスク定義、サービスなど、必要なすべての AWS リソースを説明するテンプレートを作成できます。次に、CloudFormation はプロビジョニングと必要なリソースの設定を行います。

さらに、CloudFormation を使用することで、テンプレートを再利用して Amazon ECS リソースを何度でも一貫した形でセットアップできます。リソースを一度記述すれば、複数の AWS アカウントおよび AWS リージョン で同じリソースを繰り返しプロビジョニングできるようになります。

CloudFormation テンプレートは、AWS マネジメントコンソール と AWS Command Line Interface の両方でリソースの作成に使用できます。

CloudFormation の詳細については、以下のリソースを参照してください。

- AWS CloudFormation
- AWS CloudFormation ユーザーガイド
- AWS CloudFormation コマンドラインインターフェイスユーザーガイド

トピック

- CloudFormation コンソールを使用した Amazon ECS リソースの作成
- AWS CLI コマンドを使用した CloudFormation の Amazon ECS リソースの作成
- Amazon ECS の CloudFormation テンプレートの例

CloudFormation コンソールを使用した Amazon ECS リソースの作成

CloudFormation を用いてAmazon ECS を使用する方法の一つが、AWS マネジメントコンソール を用いたやり方です。ここでは、タスク定義、クラスター、サービスなどの Amazon ECS コンポーネントの CloudFormation スタックを作成し、コンソールから直接デプロイできます。次のチュートリアルでは、CloudFormation コンソールを使用して Amazon ECS リソースを作成する方法を学習します。

前提条件

このチュートリアルでは、以下の前提条件をすでに満たしているものとします。

- 「Amazon ECS を使用するようにセットアップする」のステップを完了していること。
- IAM ユーザーに AmazonECS_FullAccess IAM ポリシー例で指定されている必要なアクセス権限があること。

ステップ 1: スタックテンプレートを作成する

次の手順を使用して、Amazon ECS サービスおよびその他の関連リソース向けの CloudFormation スタックテンプレートを作成します。

1. お好きなテキストエディタを使用して、`ecs-tutorial-template.yaml` という名前のファイルを作成します。
2. `ecs-tutorial-template.yaml` ファイルに次のテンプレートを貼り付けて、変更を保存します。

```
AWSTemplateFormatVersion: '2010-09-09'
Description: '[AWSDocs] ECS: load-balanced-web-application'

Parameters:
  VpcCidr:
    Type: String
    Default: '10.0.0.0/16'
    Description: CIDR block for the VPC
  ContainerImage:
    Type: String
    Default: 'public.ecr.aws/ecs-sample-image/amazon-ecs-sample:latest'
    Description: Container image to use in task definition
```

```
  PublicSubnet1Cidr:
    Type: String
    Default: '10.0.1.0/24'
    Description: CIDR block for public subnet 1

  PublicSubnet2Cidr:
    Type: String
    Default: '10.0.2.0/24'
    Description: CIDR block for public subnet 2

  PrivateSubnet1Cidr:
    Type: String
    Default: '10.0.3.0/24'
    Description: CIDR block for private subnet 1

  PrivateSubnet2Cidr:
    Type: String
    Default: '10.0.4.0/24'
    Description: CIDR block for private subnet 2

  ServiceName:
    Type: String
    Default: 'tutorial-app'
    Description: Name of the ECS service

  ContainerPort:
    Type: Number
    Default: 80
    Description: Port on which the container listens

  DesiredCount:
    Type: Number
    Default: 2
    Description: Desired number of tasks

  MinCapacity:
    Type: Number
    Default: 1
    Description: Minimum number of tasks for auto scaling

  MaxCapacity:
    Type: Number
    Default: 10
```

```
    Description: Maximum number of tasks for auto scaling

Resources:
  # VPC and Networking
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: !Ref VpcCidr
      EnableDnsHostnames: true
      EnableDnsSupport: true
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-vpc'

  # Internet Gateway
  InternetGateway:
    Type: AWS::EC2::InternetGateway
    Properties:
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-igw'

  InternetGatewayAttachment:
    Type: AWS::EC2::VPCGatewayAttachment
    Properties:
      InternetGatewayId: !Ref InternetGateway
      VpcId: !Ref VPC

  # Public Subnets for ALB
  PublicSubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [0, !GetAZs '']
      CidrBlock: !Ref PublicSubnet1Cidr
      MapPublicIpOnLaunch: true
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-public-subnet-1'

  PublicSubnet2:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
```

```
      AvailabilityZone: !Select [1, !GetAZs '']
      CidrBlock: !Ref PublicSubnet2Cidr
      MapPublicIpOnLaunch: true
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-public-subnet-2'

  # Private Subnets for ECS Tasks
  PrivateSubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [0, !GetAZs '']
      CidrBlock: !Ref PrivateSubnet1Cidr
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-private-subnet-1'

  PrivateSubnet2:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [1, !GetAZs '']
      CidrBlock: !Ref PrivateSubnet2Cidr
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-private-subnet-2'

  # NAT Gateways for private subnet internet access
  NatGateway1EIP:
    Type: AWS::EC2::EIP
    DependsOn: InternetGatewayAttachment
    Properties:
      Domain: vpc
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-nat-eip-1'

  NatGateway2EIP:
    Type: AWS::EC2::EIP
    DependsOn: InternetGatewayAttachment
    Properties:
      Domain: vpc
      Tags:
```

```
        - Key: Name
          Value: !Sub '${AWS::StackName}-nat-eip-2'

  NatGateway1:
    Type: AWS::EC2::NatGateway
    Properties:
      AllocationId: !GetAtt NatGateway1EIP.AllocationId
      SubnetId: !Ref PublicSubnet1
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-nat-1'

  NatGateway2:
    Type: AWS::EC2::NatGateway
    Properties:
      AllocationId: !GetAtt NatGateway2EIP.AllocationId
      SubnetId: !Ref PublicSubnet2
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-nat-2'

  # Route Tables
  PublicRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-public-routes'

  DefaultPublicRoute:
    Type: AWS::EC2::Route
    DependsOn: InternetGatewayAttachment
    Properties:
      RouteTableId: !Ref PublicRouteTable
      DestinationCidrBlock: 0.0.0.0/0
      GatewayId: !Ref InternetGateway

  PublicSubnet1RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet1
```

```
  PublicSubnet2RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet2

  PrivateRouteTable1:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-private-routes-1'

  DefaultPrivateRoute1:
    Type: AWS::EC2::Route
    Properties:
      RouteTableId: !Ref PrivateRouteTable1
      DestinationCidrBlock: 0.0.0.0/0
      NatGatewayId: !Ref NatGateway1

  PrivateSubnet1RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PrivateRouteTable1
      SubnetId: !Ref PrivateSubnet1

  PrivateRouteTable2:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-private-routes-2'

  DefaultPrivateRoute2:
    Type: AWS::EC2::Route
    Properties:
      RouteTableId: !Ref PrivateRouteTable2
      DestinationCidrBlock: 0.0.0.0/0
      NatGatewayId: !Ref NatGateway2

  PrivateSubnet2RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
```

```
    Properties:
      RouteTableId: !Ref PrivateRouteTable2
      SubnetId: !Ref PrivateSubnet2

  # Security Groups
  ALBSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupName: !Sub '${AWS::StackName}-alb-sg'
      GroupDescription: Security group for Application Load Balancer
      VpcId: !Ref VPC
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
          CidrIp: 0.0.0.0/0
          Description: Allow HTTP traffic from internet
      SecurityGroupEgress:
        - IpProtocol: -1
          CidrIp: 0.0.0.0/0
          Description: Allow all outbound traffic
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-alb-sg'

  ECSSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupName: !Sub '${AWS::StackName}-ecs-sg'
      GroupDescription: Security group for ECS tasks
      VpcId: !Ref VPC
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: !Ref ContainerPort
          ToPort: !Ref ContainerPort
          SourceSecurityGroupId: !Ref ALBSecurityGroup
          Description: Allow traffic from ALB
      SecurityGroupEgress:
        - IpProtocol: -1
          CidrIp: 0.0.0.0/0
          Description: Allow all outbound traffic
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-ecs-sg'
```

```
# Application Load Balancer
ApplicationLoadBalancer:
  Type: AWS::ElasticLoadBalancingV2::LoadBalancer
  Properties:
    Name: !Sub '${AWS::StackName}-alb'
    Scheme: internet-facing
    Type: application
    Subnets:
      - !Ref PublicSubnet1
      - !Ref PublicSubnet2
    SecurityGroups:
      - !Ref ALBSecurityGroup
    Tags:
      - Key: Name
        Value: !Sub '${AWS::StackName}-alb'

ALBTargetGroup:
  Type: AWS::ElasticLoadBalancingV2::TargetGroup
  Properties:
    Name: !Sub '${AWS::StackName}-tg'
    Port: !Ref ContainerPort
    Protocol: HTTP
    VpcId: !Ref VPC
    TargetType: ip
    HealthCheckIntervalSeconds: 30
    HealthCheckPath: /
    HealthCheckProtocol: HTTP
    HealthCheckTimeoutSeconds: 5
    HealthyThresholdCount: 2
    UnhealthyThresholdCount: 5
    Tags:
      - Key: Name
        Value: !Sub '${AWS::StackName}-tg'

ALBListener:
  Type: AWS::ElasticLoadBalancingV2::Listener
  Properties:
    DefaultActions:
      - Type: forward
        TargetGroupArn: !Ref ALBTargetGroup
    LoadBalancerArn: !Ref ApplicationLoadBalancer
    Port: 80
    Protocol: HTTP
```

```
# ECS Cluster
ECSCluster:
  Type: AWS::ECS::Cluster
  Properties:
    ClusterName: !Sub '${AWS::StackName}-cluster'
    CapacityProviders:
      - FARGATE
      - FARGATE_SPOT
    DefaultCapacityProviderStrategy:
      - CapacityProvider: FARGATE
        Weight: 1
      - CapacityProvider: FARGATE_SPOT
        Weight: 4
    ClusterSettings:
      - Name: containerInsights
        Value: enabled
    Tags:
      - Key: Name
        Value: !Sub '${AWS::StackName}-cluster'

# IAM Roles
ECSTaskExecutionRole:
  Type: AWS::IAM::Role
  Properties:
    RoleName: !Sub '${AWS::StackName}-task-execution-role'
    AssumeRolePolicyDocument:
      Version: '2012-10-17       '
      Statement:
        - Effect: Allow
          Principal:
            Service: ecs-tasks.amazonaws.com
          Action: sts:AssumeRole
    ManagedPolicyArns:
      - arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy
    Tags:
      - Key: Name
        Value: !Sub '${AWS::StackName}-task-execution-role'

ECSTaskRole:
  Type: AWS::IAM::Role
  Properties:
    RoleName: !Sub '${AWS::StackName}-task-role'
    AssumeRolePolicyDocument:
```

```
        Version: '2012-10-17       '
        Statement:
          - Effect: Allow
            Principal:
              Service: ecs-tasks.amazonaws.com
            Action: sts:AssumeRole
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-task-role'

  # CloudWatch Log Group
  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Sub '/ecs/${AWS::StackName}'
      RetentionInDays: 7

  # ECS Task Definition
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      Family: !Sub '${AWS::StackName}-task'
      Cpu: '256'
      Memory: '512'
      NetworkMode: awsvpc
      RequiresCompatibilities:
        - FARGATE
      ExecutionRoleArn: !GetAtt ECSTaskExecutionRole.Arn
      TaskRoleArn: !GetAtt ECSTaskRole.Arn
      ContainerDefinitions:
        - Name: !Ref ServiceName
          Image: !Ref ContainerImage
          PortMappings:
            - ContainerPort: !Ref ContainerPort
              Protocol: tcp
          Essential: true
          LogConfiguration:
            LogDriver: awslogs
            Options:
              awslogs-group: !Ref LogGroup
              awslogs-region: !Ref AWS::Region
              awslogs-stream-prefix: ecs
          HealthCheck:
            Command:
```

```
              - CMD-SHELL
              - curl -f http://localhost/ || exit 1
            Interval: 30
            Timeout: 5
            Retries: 3
            StartPeriod: 60
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-task'

  # ECS Service
  ECSService:
    Type: AWS::ECS::Service
    DependsOn: ALBListener
    Properties:
      ServiceName: !Sub '${AWS::StackName}-service'
      Cluster: !Ref ECSCluster
      TaskDefinition: !Ref TaskDefinition
      DesiredCount: !Ref DesiredCount
      LaunchType: FARGATE
      PlatformVersion: LATEST
      NetworkConfiguration:
        AwsvpcConfiguration:
          AssignPublicIp: DISABLED
          SecurityGroups:
            - !Ref ECSSecurityGroup
          Subnets:
            - !Ref PrivateSubnet1
            - !Ref PrivateSubnet2
      LoadBalancers:
        - ContainerName: !Ref ServiceName
          ContainerPort: !Ref ContainerPort
          TargetGroupArn: !Ref ALBTargetGroup
      DeploymentConfiguration:
        MaximumPercent: 200
        MinimumHealthyPercent: 50
        DeploymentCircuitBreaker:
          Enable: true
          Rollback: true
      EnableExecuteCommand: true  # For debugging
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-service'
```

```
  # Auto Scaling Target
  ServiceScalingTarget:
    Type: AWS::ApplicationAutoScaling::ScalableTarget
    Properties:
      MaxCapacity: !Ref MaxCapacity
      MinCapacity: !Ref MinCapacity
      ResourceId: !Sub 'service/${ECSCluster}/${ECSService.Name}'
      RoleARN: !Sub 'arn:aws:iam::${AWS::AccountId}:role/
aws-service-role/ecs.application-autoscaling.amazonaws.com/
AWSServiceRoleForApplicationAutoScaling_ECSService'
      ScalableDimension: ecs:service:DesiredCount
      ServiceNamespace: ecs

  # Auto Scaling Policy - CPU Utilization
  ServiceScalingPolicy:
    Type: AWS::ApplicationAutoScaling::ScalingPolicy
    Properties:
      PolicyName: !Sub '${AWS::StackName}-cpu-scaling-policy'
      PolicyType: TargetTrackingScaling
      ScalingTargetId: !Ref ServiceScalingTarget
      TargetTrackingScalingPolicyConfiguration:
        PredefinedMetricSpecification:
          PredefinedMetricType: ECSServiceAverageCPUUtilization
        TargetValue: 70.0
        ScaleOutCooldown: 300
        ScaleInCooldown: 300

Outputs:
  VPCId:
    Description: VPC ID
    Value: !Ref VPC
    Export:
      Name: !Sub '${AWS::StackName}-VPC-ID'

  LoadBalancerURL:
    Description: URL of the Application Load Balancer
    Value: !Sub 'http://${ApplicationLoadBalancer.DNSName}'
    Export:
      Name: !Sub '${AWS::StackName}-ALB-URL'

  ECSClusterName:
    Description: Name of the ECS Cluster
    Value: !Ref ECSCluster
    Export:
```

```
      Name: !Sub '${AWS::StackName}-ECS-Cluster'

  ECSServiceName:
    Description: Name of the ECS Service
    Value: !GetAtt ECSService.Name
    Export:
      Name: !Sub '${AWS::StackName}-ECS-Service'

  PrivateSubnet1:
    Description: Private Subnet 1 ID
    Value: !Ref PrivateSubnet1
    Export:
      Name: !Sub '${AWS::StackName}-Private-Subnet-1'

  PrivateSubnet2:
    Description: Private Subnet 2 ID
    Value: !Ref PrivateSubnet2
    Export:
      Name: !Sub '${AWS::StackName}-Private-Subnet-2'
```

このチュートリアルで使用するテンプレートは、Fargate で実行する 2 つのタスクを持つ Amazon ECS サービスを作成します。このチュートリアルで使用するテンプレートは、Fargate で実行する 2 つのタスクを持つ Amazon ECS サービスを作成します。テンプレートは、アプリケーショントラフィックを分散する Application Load Balancer と、CPU 使用率に基づいて、アプリケーションをスケーリングする Application Auto Scaling ポリシーも作成します。更に、テンプレートは、アプリケーションのデプロイに必要なネットワークリソース、コンテナログのログ記録リソース、および Amazon ECS タスク実行 IAM ロールも作成します。タスクの実行ロールの詳細については、「Amazon ECS タスク実行IAM ロール」を参照してください。自動スケーリングの詳細については、「Amazon ECS サービスを自動的にスケールする」を参照してください。

ステップ 2: Amazon ECS リソースのスタックを作成する

テンプレートのファイルを作成したら、以下の手順に従って、CloudFormation コンソールを使用してテンプレートでスタックを作成できます。

CloudFormation コンソールを使用してスタックを作成する方法については、「AWS CloudFormation ユーザーガイド」の「CloudFormation コンソールでのスタックの作成」を参照してください。次の表を使用して、指定するオプションを決定します。

オプション	値
前提条件 - テンプレートの準備	既存のテンプレートを選択する
テンプレートを指定する	Upload a template file (テンプレートファイルのアップロード)
ファイルを選択します。	ecs-tutorial-template.yaml
スタック名	ecs-tutorial-stack
パラメータ	すべてのパラメータはデフォルト値のままにしておく
機能	CloudFormation が IAM リソースを作成することを認めるには、[このテンプレートが IAM リソースを作成する可能性を認識しています] を選択します。

ステップ 3: 確認する

提供されたテンプレートを使用して Amazon ECS リソースの作成を確認するには、次の手順に従います。

スタック情報とリソースを表示する方法については、「AWS CloudFormation ユーザーガイド」[CloudFormation コンソールからスタック情報を表示する」を参照してください。また、次の表を使用して検証対象を決定します。

スタックの詳細フィールド	検索対象
Stack info (スタック情報)	`CREATE_COMPLETE` のステータス

スタックの詳細フィールド	検索対象
リソース	サービスコンソールへのリンクを含む、作成されたリソースのリスト。作成されたサービス、クラスター、およびタスク定義の詳細を Amazon ECS コンソールで表示するには、`ECSCluster` 、`ECSService` 、`TaskDefinition` へのリンクを選択します。
アウトプット	LoadBalancerURL サンプルの Amazon ECS アプリケーションを表示するウェブページを表示するには、URL をウェブブラウザに貼り付けます。

ステップ 4: リソースをクリーンアップする

リソースをクリーンアップし、追加のコストが発生しないようにするには、「CloudFormation ユーザーガイド」の「CloudFormation コンソールからスタックを削除する」の手順に従います。

AWS CLI コマンドを使用した CloudFormation の Amazon ECS リソースの作成

CloudFormation を使用して Amazon ECS を AWS CLI と連携させる別の方法があります。コマンドを使用して、タスク定義、クラスター、サービスなどの Amazon ECS コンポーネントの CloudFormation スタックを作成してデプロイできます。次のチュートリアルでは、AWS CLI を使用して CloudFormation テンプレートを使用して Amazon ECS リソースを作成する方法を学習します。

前提条件

- 「Amazon ECS を使用するようにセットアップする」のステップを完了していること。

- IAM ユーザーに AmazonECS_FullAccess IAM ポリシー例で指定されている必要なアクセス権限があること。

ステップ 1: スタックを作成する

ecs-tutorial-template.yaml というファイルに保存されている AWS CLI を使用してスタックを作成するには、次のコマンドを実行します。

```
cat << 'EOF' > ecs-tutorial-template.yaml
AWSTemplateFormatVersion: '2010-09-09'
Description: '[AWSDocs] ECS: load-balanced-web-application'
Parameters:
  VpcCidr:
    Type: String
    Default: '10.0.0.0/16'
    Description: CIDR block for the VPC
  ContainerImage:
    Type: String
    Default: 'public.ecr.aws/ecs-sample-image/amazon-ecs-sample:latest'
    Description: Container image to use in task definition

  PublicSubnet1Cidr:
    Type: String
    Default: '10.0.1.0/24'
    Description: CIDR block for public subnet 1

  PublicSubnet2Cidr:
    Type: String
    Default: '10.0.2.0/24'
    Description: CIDR block for public subnet 2

  PrivateSubnet1Cidr:
    Type: String
    Default: '10.0.3.0/24'
    Description: CIDR block for private subnet 1

  PrivateSubnet2Cidr:
    Type: String
    Default: '10.0.4.0/24'
    Description: CIDR block for private subnet 2

  ServiceName:
```

```
    Type: String
    Default: 'tutorial-app'
    Description: Name of the ECS service

  ContainerPort:
    Type: Number
    Default: 80
    Description: Port on which the container listens

  DesiredCount:
    Type: Number
    Default: 2
    Description: Desired number of tasks

  MinCapacity:
    Type: Number
    Default: 1
    Description: Minimum number of tasks for auto scaling

  MaxCapacity:
    Type: Number
    Default: 10
    Description: Maximum number of tasks for auto scaling

Resources:
  # VPC and Networking
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: !Ref VpcCidr
      EnableDnsHostnames: true
      EnableDnsSupport: true
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-vpc'

  # Internet Gateway
  InternetGateway:
    Type: AWS::EC2::InternetGateway
    Properties:
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-igw'
```

```
  InternetGatewayAttachment:
    Type: AWS::EC2::VPCGatewayAttachment
    Properties:
      InternetGatewayId: !Ref InternetGateway
      VpcId: !Ref VPC

  # Public Subnets for ALB
  PublicSubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [0, !GetAZs '']
      CidrBlock: !Ref PublicSubnet1Cidr
      MapPublicIpOnLaunch: true
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-public-subnet-1'

  PublicSubnet2:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [1, !GetAZs '']
      CidrBlock: !Ref PublicSubnet2Cidr
      MapPublicIpOnLaunch: true
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-public-subnet-2'

  # Private Subnets for ECS Tasks
  PrivateSubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [0, !GetAZs '']
      CidrBlock: !Ref PrivateSubnet1Cidr
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-private-subnet-1'

  PrivateSubnet2:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
```

```
      AvailabilityZone: !Select [1, !GetAZs '']
      CidrBlock: !Ref PrivateSubnet2Cidr
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-private-subnet-2'

  # NAT Gateways for private subnet internet access
  NatGateway1EIP:
    Type: AWS::EC2::EIP
    DependsOn: InternetGatewayAttachment
    Properties:
      Domain: vpc
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-nat-eip-1'

  NatGateway2EIP:
    Type: AWS::EC2::EIP
    DependsOn: InternetGatewayAttachment
    Properties:
      Domain: vpc
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-nat-eip-2'

  NatGateway1:
    Type: AWS::EC2::NatGateway
    Properties:
      AllocationId: !GetAtt NatGateway1EIP.AllocationId
      SubnetId: !Ref PublicSubnet1
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-nat-1'

  NatGateway2:
    Type: AWS::EC2::NatGateway
    Properties:
      AllocationId: !GetAtt NatGateway2EIP.AllocationId
      SubnetId: !Ref PublicSubnet2
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-nat-2'

  # Route Tables
```

```
  PublicRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-public-routes'

  DefaultPublicRoute:
    Type: AWS::EC2::Route
    DependsOn: InternetGatewayAttachment
    Properties:
      RouteTableId: !Ref PublicRouteTable
      DestinationCidrBlock: 0.0.0.0/0
      GatewayId: !Ref InternetGateway

  PublicSubnet1RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet1

  PublicSubnet2RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet2

  PrivateRouteTable1:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-private-routes-1'

  DefaultPrivateRoute1:
    Type: AWS::EC2::Route
    Properties:
      RouteTableId: !Ref PrivateRouteTable1
      DestinationCidrBlock: 0.0.0.0/0
      NatGatewayId: !Ref NatGateway1

  PrivateSubnet1RouteTableAssociation:
```

```
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PrivateRouteTable1
      SubnetId: !Ref PrivateSubnet1

  PrivateRouteTable2:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-private-routes-2'

  DefaultPrivateRoute2:
    Type: AWS::EC2::Route
    Properties:
      RouteTableId: !Ref PrivateRouteTable2
      DestinationCidrBlock: 0.0.0.0/0
      NatGatewayId: !Ref NatGateway2

  PrivateSubnet2RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PrivateRouteTable2
      SubnetId: !Ref PrivateSubnet2

  # Security Groups
  ALBSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupName: !Sub '${AWS::StackName}-alb-sg'
      GroupDescription: Security group for Application Load Balancer
      VpcId: !Ref VPC
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
          CidrIp: 0.0.0.0/0
          Description: Allow HTTP traffic from internet
      SecurityGroupEgress:
        - IpProtocol: -1
          CidrIp: 0.0.0.0/0
          Description: Allow all outbound traffic
      Tags:
```

```
        - Key: Name
          Value: !Sub '${AWS::StackName}-alb-sg'

  ECSSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupName: !Sub '${AWS::StackName}-ecs-sg'
      GroupDescription: Security group for ECS tasks
      VpcId: !Ref VPC
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: !Ref ContainerPort
          ToPort: !Ref ContainerPort
          SourceSecurityGroupId: !Ref ALBSecurityGroup
          Description: Allow traffic from ALB
      SecurityGroupEgress:
        - IpProtocol: -1
          CidrIp: 0.0.0.0/0
          Description: Allow all outbound traffic
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-ecs-sg'

  # Application Load Balancer
  ApplicationLoadBalancer:
    Type: AWS::ElasticLoadBalancingV2::LoadBalancer
    Properties:
      Name: !Sub '${AWS::StackName}-alb'
      Scheme: internet-facing
      Type: application
      Subnets:
        - !Ref PublicSubnet1
        - !Ref PublicSubnet2
      SecurityGroups:
        - !Ref ALBSecurityGroup
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-alb'

  ALBTargetGroup:
    Type: AWS::ElasticLoadBalancingV2::TargetGroup
    Properties:
      Name: !Sub '${AWS::StackName}-tg'
      Port: !Ref ContainerPort
```

```
      Protocol: HTTP
      VpcId: !Ref VPC
      TargetType: ip
      HealthCheckIntervalSeconds: 30
      HealthCheckPath: /
      HealthCheckProtocol: HTTP
      HealthCheckTimeoutSeconds: 5
      HealthyThresholdCount: 2
      UnhealthyThresholdCount: 5
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-tg'

  ALBListener:
    Type: AWS::ElasticLoadBalancingV2::Listener
    Properties:
      DefaultActions:
        - Type: forward
          TargetGroupArn: !Ref ALBTargetGroup
      LoadBalancerArn: !Ref ApplicationLoadBalancer
      Port: 80
      Protocol: HTTP

  # ECS Cluster
  ECSCluster:
    Type: AWS::ECS::Cluster
    Properties:
      ClusterName: !Sub '${AWS::StackName}-cluster'
      CapacityProviders:
        - FARGATE
        - FARGATE_SPOT
      DefaultCapacityProviderStrategy:
        - CapacityProvider: FARGATE
          Weight: 1
        - CapacityProvider: FARGATE_SPOT
          Weight: 4
      ClusterSettings:
        - Name: containerInsights
          Value: enabled
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-cluster'

  # IAM Roles
```

```
  ECSTaskExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: !Sub '${AWS::StackName}-task-execution-role'
      AssumeRolePolicyDocument:
        Version: '2012-10-17       '
        Statement:
          - Effect: Allow
            Principal:
              Service: ecs-tasks.amazonaws.com
            Action: sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-task-execution-role'

  ECSTaskRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: !Sub '${AWS::StackName}-task-role'
      AssumeRolePolicyDocument:
        Version: '2012-10-17       '
        Statement:
          - Effect: Allow
            Principal:
              Service: ecs-tasks.amazonaws.com
            Action: sts:AssumeRole
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-task-role'

  # CloudWatch Log Group
  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Sub '/ecs/${AWS::StackName}'
      RetentionInDays: 7

  # ECS Task Definition
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      Family: !Sub '${AWS::StackName}-task'
```

```
      Cpu: '256'
      Memory: '512'
      NetworkMode: awsvpc
      RequiresCompatibilities:
        - FARGATE
      ExecutionRoleArn: !GetAtt ECSTaskExecutionRole.Arn
      TaskRoleArn: !GetAtt ECSTaskRole.Arn
      ContainerDefinitions:
        - Name: !Ref ServiceName
          Image: !Ref ContainerImage
          PortMappings:
            - ContainerPort: !Ref ContainerPort
              Protocol: tcp
          Essential: true
          LogConfiguration:
            LogDriver: awslogs
            Options:
              awslogs-group: !Ref LogGroup
              awslogs-region: !Ref AWS::Region
              awslogs-stream-prefix: ecs
          HealthCheck:
            Command:
              - CMD-SHELL
              - curl -f http://localhost/ || exit 1
            Interval: 30
            Timeout: 5
            Retries: 3
            StartPeriod: 60
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-task'

  # ECS Service
  ECSService:
    Type: AWS::ECS::Service
    DependsOn: ALBListener
    Properties:
      ServiceName: !Sub '${AWS::StackName}-service'
      Cluster: !Ref ECSCluster
      TaskDefinition: !Ref TaskDefinition
      DesiredCount: !Ref DesiredCount
      LaunchType: FARGATE
      PlatformVersion: LATEST
      NetworkConfiguration:
```

```
        AwsvpcConfiguration:
          AssignPublicIp: DISABLED
          SecurityGroups:
            - !Ref ECSSecurityGroup
          Subnets:
            - !Ref PrivateSubnet1
            - !Ref PrivateSubnet2
      LoadBalancers:
        - ContainerName: !Ref ServiceName
          ContainerPort: !Ref ContainerPort
          TargetGroupArn: !Ref ALBTargetGroup
      DeploymentConfiguration:
        MaximumPercent: 200
        MinimumHealthyPercent: 50
        DeploymentCircuitBreaker:
          Enable: true
          Rollback: true
      EnableExecuteCommand: true  # For debugging
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-service'

  # Auto Scaling Target
  ServiceScalingTarget:
    Type: AWS::ApplicationAutoScaling::ScalableTarget
    Properties:
      MaxCapacity: !Ref MaxCapacity
      MinCapacity: !Ref MinCapacity
      ResourceId: !Sub 'service/${ECSCluster}/${ECSService.Name}'
      RoleARN: !Sub 'arn:aws:iam::${AWS::AccountId}:role/
aws-service-role/ecs.application-autoscaling.amazonaws.com/
AWSServiceRoleForApplicationAutoScaling_ECSService'
      ScalableDimension: ecs:service:DesiredCount
      ServiceNamespace: ecs

  # Auto Scaling Policy - CPU Utilization
  ServiceScalingPolicy:
    Type: AWS::ApplicationAutoScaling::ScalingPolicy
    Properties:
      PolicyName: !Sub '${AWS::StackName}-cpu-scaling-policy'
      PolicyType: TargetTrackingScaling
      ScalingTargetId: !Ref ServiceScalingTarget
      TargetTrackingScalingPolicyConfiguration:
        PredefinedMetricSpecification:
```

```
          PredefinedMetricType: ECSServiceAverageCPUUtilization
        TargetValue: 70.0
        ScaleOutCooldown: 300
        ScaleInCooldown: 300

Outputs:
  VPCId:
    Description: VPC ID
    Value: !Ref VPC
    Export:
      Name: !Sub '${AWS::StackName}-VPC-ID'

  LoadBalancerURL:
    Description: URL of the Application Load Balancer
    Value: !Sub 'http://${ApplicationLoadBalancer.DNSName}'
    Export:
      Name: !Sub '${AWS::StackName}-ALB-URL'

  ECSClusterName:
    Description: Name of the ECS Cluster
    Value: !Ref ECSCluster
    Export:
      Name: !Sub '${AWS::StackName}-ECS-Cluster'

  ECSServiceName:
    Description: Name of the ECS Service
    Value: !GetAtt ECSService.Name
    Export:
      Name: !Sub '${AWS::StackName}-ECS-Service'

  PrivateSubnet1:
    Description: Private Subnet 1 ID
    Value: !Ref PrivateSubnet1
    Export:
      Name: !Sub '${AWS::StackName}-Private-Subnet-1'

  PrivateSubnet2:
    Description: Private Subnet 2 ID
    Value: !Ref PrivateSubnet2
    Export:
      Name: !Sub '${AWS::StackName}-Private-Subnet-2'
EOF
```

このチュートリアルで使用するテンプレートは、Fargate で実行する 2 つのタスクを持つ Amazon ECS サービスを作成します。このチュートリアルで使用するテンプレートは、Fargate で実行する 2 つのタスクを持つ Amazon ECS サービスを作成します。テンプレートは、アプリケーショントラフィックを分散する Application Load Balancer と、CPU 使用率に基づいて、アプリケーションをスケーリングする Application Auto Scaling ポリシーも作成します。更に、テンプレートは、アプリケーションのデプロイに必要なネットワークリソース、コンテナログのログ記録リソース、および Amazon ECS タスク実行 IAM ロールも作成します。タスクの実行ロールの詳細については、「Amazon ECS タスク実行IAM ロール」を参照してください。自動スケーリングの詳細については、「Amazon ECS サービスを自動的にスケールする」を参照してください。

テンプレートファイルを作成したら、次のコマンドを使用してスタックを作成します。テンプレートで指定した Amazon ECS タスク実行ロールを作成するには、`--capabilities` フラグが必要です。`--parameters` フラグを指定して、テンプレートパラメータをカスタマイズすることもできます。

```
aws cloudformation create-stack \
      --stack-name ecs-tutorial-stack \
      --template-body file://ecs-tutorial-template.yaml \
      --region aws-region \
      --capabilities CAPABILITY_NAMED_IAM
```

`create-stack` コマンドを実行した後、 `describe-stacks` を使用してスタック作成のステータスを確認できます。

```
aws cloudformation describe-stacks \
      --stack-name ecs-tutorial-stack \
      --region aws-region
```

ステップ 2: Amazon ECS リソースの作成を検証する

Amazon ECS リソースが正しく作成されていることを確認するには、次の手順に従います。

1. 次のコマンドを実行して、AWS リージョン 内のすべてのタスク定義を一覧表示します。

   ```
   aws ecs list-task-definitions
   ```

   コマンドは、タスク定義の Amazon リソースネーム (ARN) のリストを返します。テンプレートを使用して作成したタスク定義の ARN は、次の形式で表示されます。

```
{
    "taskDefinitionArns": [
     .....
        "arn:aws:ecs:aws-region:111122223333:task-definition/ecs-tutorial-stack-
task:1",
     .....
    ]
}
```

2. 次のコマンドを実行して、AWS リージョン 内のすべてのクラスターを一覧表示します。

```
aws ecs list-clusters
```

コマンドはクラスター ARN のリストを返します。テンプレートを使用して作成したクラスターの ARN は、次の形式で表示されます。

```
{
    "clusterArns": [
        .....
        "arn:aws:ecs:aws-region:111122223333:cluster/ecs-tutorial-stack-cluster",
        .....
    ]
}
```

3. クラスター `ecs-tutorial-stack-cluster` 内のすべてのサービスを一覧表示するには、次のコマンドを実行します。

```
aws ecs list-services \
      --cluster ecs-tutorial-stack-cluster
```

コマンドは、サービス ARN のリストを返します。テンプレートを使用して作成されたサービスの ARN は、次の形式で表示されます。

```
{
    "serviceArns": [
        "arn:aws:ecs:aws-region:111122223333:service/ecs-tutorial-stack-cluster/
ecs-tutorial-stack-service"
    ]
}
```

作成された Application Load Balancer の DNS 名を取得し、それを使用してリソースの作成を確認することもできます。DNS 名を取得するには、次のコマンドを実行します。

作成されたスタックの出力を取得するには、次のコマンドを実行します。

```
aws cloudformation describe-stacks \
  --stack-name ecs-tutorial-stack \
  --region aws-region \
  --query 'Stacks[0].Outputs[?OutputKey==`LoadBalancerURL`].OutputValue' \
  --output  text
```

出力:

```
http://ecs-tutorial-stack-alb-0123456789.aws-region.elb.amazonaws.com
```

DNS 名をブラウザに貼り付けて、Amazon ECS アプリケーションのサンプルを表示するウェブページを表示します。

ステップ 3: クリーンアップする

作成したリソースをクリーンアップするには、以下のコマンドを実行します。

```
aws cloudformation delete-stack \
      --stack-name ecs-stack
```

delete-stack コマンドは、このチュートリアルで作成した CloudFormation スタックの削除を開始し、スタック内のすべてのリソースを削除します。削除を確認するには、ステップ 2: Amazon ECS リソースの作成を検証する で同じ手順を繰り返します。出力内の ARN のリストには、ecs-tutorial-stack-task というタスク定義や ecs-tutorial-stack-cluster という名前のクラスターが含まれなくなります。list-services 呼び出しは失敗します。

Amazon ECS の CloudFormation テンプレートの例

CloudFormation を使用して Amazon ECS クラスター、タスク定義、サービスを作成できます。以下のトピックには、さまざまな設定でリソースを作成する方法を示すテンプレートが含まれています。CloudFormation コンソールまたは AWS CLI を使用して、これらのテンプレートでリソースを作成することができます。

CloudFormation テンプレートは、CloudFormation スタックにプロビジョニングするリソースを記述する JSON または YAML 形式のテキストファイルです。JSON や YAML 形式に不慣れな方も、AWS

Infrastructure Composer を使用することで CloudFormation テンプレートの使用を開始できます。詳細については、「AWS CloudFormationユーザーガイド」の「Infrastructure Composer でテンプレートを視覚的に作成する」を参照してください。

以下のトピックでは、Amazon ECS タスク定義、クラスター、サービスのテンプレート例を一覧表示します。

トピック

- タスク定義
- キャパシティープロバイダー
- クラスター
- サービス
- Amazon ECS の IAM ロール

タスク定義

タスク定義は、アプリケーションを形成するパラメータと 1 つ以上のコンテナを記述する、アプリケーションの設計図のようなものです。Amazon ECS タスク定義の CloudFormation テンプレートの例を次に示します。Amazon ECS タスク定義の詳細については、「Amazon ECSの タスク定義」を参照してください。

Fargate Linux のタスク定義

以下のテンプレートを使用して、Fargate Linux のサンプルタスクを作成できます。

JSON

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "ECS Task Definition with parameterized values",
  "Parameters": {
    "ContainerImage": {
      "Type": "String",
      "Default": "public.ecr.aws/docker/library/httpd:2.4",
      "Description": "The container image to use for the task"
    },
    "ContainerCpu": {
      "Type": "Number",
      "Default": 256,
```

```
      "Description": "The number of CPU units to reserve for the container",
      "AllowedValues": [256, 512, 1024, 2048, 4096]
    },
    "ContainerMemory": {
      "Type": "Number",
      "Default": 512,
      "Description": "The amount of memory (in MiB) to reserve for the container",
      "AllowedValues": [512, 1024, 2048, 3072, 4096, 5120, 6144, 7168, 8192]
    },
    "TaskFamily": {
      "Type": "String",
      "Default": "task-definition-cfn",
      "Description": "The name of the task definition family"
    },
    "ContainerName": {
      "Type": "String",
      "Default": "sample-fargate-app",
      "Description": "The name of the container"
    },
    "ContainerPort": {
      "Type": "Number",
      "Default": 80,
      "Description": "The port number on the container"
    },
    "HostPort": {
      "Type": "Number",
      "Default": 80,
      "Description": "The port number on the host"
    },
    "ExecutionRoleArn": {
      "Type": "String",
      "Default": "arn:aws:iam::aws_account_id:role/ecsTaskExecutionRole",
      "Description": "The ARN of the task execution role"
    },
    "LogGroup": {
      "Type": "String",
      "Default": "/ecs/fargate-task-definition",
      "Description": "The CloudWatch log group for container logs"
    },
    "NetworkMode": {
      "Type": "String",
      "Default": "awsvpc",
      "Description": "The Docker networking mode to use",
      "AllowedValues": ["awsvpc", "bridge", "host", "none"]
```

```
    },
    "OperatingSystemFamily": {
      "Type": "String",
      "Default": "LINUX",
      "Description": "The operating system for the task",
      "AllowedValues": ["LINUX", "WINDOWS_SERVER_2019_FULL",
 "WINDOWS_SERVER_2019_CORE", "WINDOWS_SERVER_2022_FULL", "WINDOWS_SERVER_2022_CORE"]
    }
  },
  "Resources": {
    "ECSTaskDefinition": {
      "Type": "AWS::ECS::TaskDefinition",
      "Properties": {
        "ContainerDefinitions": [
          {
            "Command": [
              "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample App</
title> <style>body {margin-top: 40px; background-color: #333;} </style> </
head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</
h1> <h2>Congratulations!</h2> <p>Your application is now running on a container in
 Amazon ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html && -
foreground\""
            ],
            "EntryPoint": [
              "sh",
              "-c"
            ],
            "Essential": true,
            "Image": {"Ref": "ContainerImage"},
            "LogConfiguration": {
              "LogDriver": "awslogs",
              "Options": {
                "mode": "non-blocking",
                "max-buffer-size": "25m",
                "awslogs-create-group": "true",
                "awslogs-group": {"Ref": "LogGroup"},
                "awslogs-region": {"Ref": "AWS::Region"},
                "awslogs-stream-prefix": "ecs"
              }
            },
            "Name": {"Ref": "ContainerName"},
            "PortMappings": [
              {
                "ContainerPort": {"Ref": "ContainerPort"},
```

```
                "HostPort": {"Ref": "HostPort"},
                "Protocol": "tcp"
              }
            ]
          }
        ],
        "Cpu": {"Ref": "ContainerCpu"},
        "ExecutionRoleArn": {"Ref": "ExecutionRoleArn"},
        "Family": {"Ref": "TaskFamily"},
        "Memory": {"Ref": "ContainerMemory"},
        "NetworkMode": {"Ref": "NetworkMode"},
        "RequiresCompatibilities": [
          "FARGATE"
        ],
        "RuntimePlatform": {
          "OperatingSystemFamily": {"Ref": "OperatingSystemFamily"}
        }
      }
    }
  },
  "Outputs": {
    "TaskDefinitionArn": {
      "Description": "The ARN of the created task definition",
      "Value": {"Ref": "ECSTaskDefinition"}
    }
  }
}
```

YAML

```
AWSTemplateFormatVersion: 2010-09-09
Description: 'ECS Task Definition to deploy a sample app'
Parameters:
  ContainerImage:
    Type: String
    Default: 'public.ecr.aws/docker/library/httpd:2.4'
    Description: The container image to use for the task
  ContainerCpu:
    Type: Number
    Default: 256
    Description: The number of CPU units to reserve for the container
    AllowedValues: [256, 512, 1024, 2048, 4096]
  ContainerMemory:
```

```
    Type: Number
    Default: 512
    Description: The amount of memory (in MiB) to reserve for the container
    AllowedValues: [512, 1024, 2048, 3072, 4096, 5120, 6144, 7168, 8192]
  TaskFamily:
    Type: String
    Default: 'task-definition-cfn'
    Description: The name of the task definition family
  ContainerName:
    Type: String
    Default: 'sample-fargate-app'
    Description: The name of the container
  ContainerPort:
    Type: Number
    Default: 80
    Description: The port number on the container
  HostPort:
    Type: Number
    Default: 80
    Description: The port number on the host
  ExecutionRoleArn:
    Type: String
    Default: 'arn:aws:iam::111122223333:role/ecsTaskExecutionRole'
    Description: The ARN of the task execution role
  LogGroup:
    Type: String
    Default: '/ecs/fargate-task-definition'
    Description: The CloudWatch log group for container logs
  NetworkMode:
    Type: String
    Default: 'awsvpc'
    Description: The Docker networking mode to use
    AllowedValues: ['awsvpc', 'bridge', 'host', 'none']
  OperatingSystemFamily:
    Type: String
    Default: 'LINUX'
    Description: The operating system for the task
    AllowedValues: ['LINUX', 'WINDOWS_SERVER_2019_FULL', 'WINDOWS_SERVER_2019_CORE',
 'WINDOWS_SERVER_2022_FULL', 'WINDOWS_SERVER_2022_CORE']
Resources:
  ECSTaskDefinition:
    Type: 'AWS::ECS::TaskDefinition'
    Properties:
      ContainerDefinitions:
```

```
        - Command:
            - >-
              /bin/sh -c "echo '<html> <head> <title>Amazon ECS Sample
              App</title> <style>body {margin-top: 40px; background-color:
              #333;} </style> </head><body> <div
              style=color:white;text-align:center> <h1>Amazon ECS Sample
              App</h1> <h2>Congratulations!</h2> <p>Your application is now
              running on a container in Amazon ECS.</p> </div></body></html>' >
              /usr/local/apache2/htdocs/index.html && httpd-foreground"
          EntryPoint:
            - sh
            - '-c'
          Essential: true
          Image: !Ref ContainerImage
          LogConfiguration:
            LogDriver: awslogs
            Options:
              mode: non-blocking
              max-buffer-size: 25m
              awslogs-create-group: 'true'
              awslogs-group: !Ref LogGroup
              awslogs-region: !Ref AWS::Region
              awslogs-stream-prefix: ecs
          Name: !Ref ContainerName
          PortMappings:
            - ContainerPort: !Ref ContainerPort
              HostPort: !Ref HostPort
              Protocol: tcp
      Cpu: !Ref ContainerCpu
      ExecutionRoleArn: !Ref ExecutionRoleArn
      Family: !Ref TaskFamily
      Memory: !Ref ContainerMemory
      NetworkMode: !Ref NetworkMode
      RequiresCompatibilities:
        - FARGATE
      RuntimePlatform:
        OperatingSystemFamily: !Ref OperatingSystemFamily
Outputs:
  TaskDefinitionArn:
    Description: The ARN of the created task definition
    Value: !Ref ECSTaskDefinition
```

Amazon EFS のタスク定義

次のテンプレートを使用して、作成した Amazon EFS ファイルシステムを使用するタスクを作成できます。Amazon ECS での Amazon EFS ボリューム使用の詳細については、「Amazon ECS での Amazon EFS ボリュームの使用」を参照してください。

JSON

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "Create a task definition for a web server with parameterized
 values.",
  "Parameters": {
    "ExecutionRoleArn": {
      "Type": "String",
      "Default": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
      "Description": "The ARN of the task execution role"
    },
    "NetworkMode": {
      "Type": "String",
      "Default": "awsvpc",
      "Description": "The Docker networking mode to use",
      "AllowedValues": ["awsvpc", "bridge", "host", "none"]
    },
    "TaskFamily": {
      "Type": "String",
      "Default": "my-ecs-task",
      "Description": "The name of the task definition family"
    },
    "ContainerCpu": {
      "Type": "String",
      "Default": "256",
      "Description": "The number of CPU units to reserve for the container",
      "AllowedValues": ["256", "512", "1024", "2048", "4096"]
    },
    "ContainerMemory": {
      "Type": "String",
      "Default": "512",
      "Description": "The amount of memory (in MiB) to reserve for the container",
      "AllowedValues": ["512", "1024", "2048", "3072", "4096", "5120", "6144",
 "7168", "8192"]
    },
    "ContainerName": {
```

```
      "Type": "String",
      "Default": "nginx",
      "Description": "The name of the container"
    },
    "ContainerImage": {
      "Type": "String",
      "Default": "public.ecr.aws/nginx/nginx:latest",
      "Description": "The container image to use for the task"
    },
    "ContainerPort": {
      "Type": "Number",
      "Default": 80,
      "Description": "The port number on the container"
    },
    "InitProcessEnabled": {
      "Type": "String",
      "Default": "true",
      "Description": "Whether to enable the init process inside the container",
      "AllowedValues": ["true", "false"]
    },
    "EfsVolumeName": {
      "Type": "String",
      "Default": "efs-volume",
      "Description": "The name of the EFS volume"
    },
    "EfsContainerPath": {
      "Type": "String",
      "Default": "/usr/share/nginx/html",
      "Description": "The path in the container where the EFS volume will be
mounted"
    },
    "LogGroup": {
      "Type": "String",
      "Default": "LogGroup",
      "Description": "The CloudWatch log group for container logs"
    },
    "LogStreamPrefix": {
      "Type": "String",
      "Default": "efs-task",
      "Description": "The prefix for the log stream"
    },
    "EfsFilesystemId": {
      "Type": "String",
      "Default": "fs-1234567890abcdef0",
```

```
      "Description": "The ID of the EFS filesystem"
    },
    "EfsRootDirectory": {
      "Type": "String",
      "Default": "/",
      "Description": "The root directory in the EFS filesystem"
    },
    "EfsTransitEncryption": {
      "Type": "String",
      "Default": "ENABLED",
      "Description": "Whether to enable transit encryption for EFS",
      "AllowedValues": ["ENABLED", "DISABLED"]
    }
  },
  "Resources": {
    "ECSTaskDefinition": {
      "Type": "AWS::ECS::TaskDefinition",
      "Properties": {
        "ExecutionRoleArn": {"Ref": "ExecutionRoleArn"},
        "NetworkMode": {"Ref": "NetworkMode"},
        "RequiresCompatibilities": ["FARGATE"],
        "Family": {"Ref": "TaskFamily"},
        "Cpu": {"Ref": "ContainerCpu"},
        "Memory": {"Ref": "ContainerMemory"},
        "ContainerDefinitions": [
          {
            "Name": {"Ref": "ContainerName"},
            "Image": {"Ref": "ContainerImage"},
            "Essential": true,
            "PortMappings": [
              {
                "ContainerPort": {"Ref": "ContainerPort"},
                "Protocol": "tcp"
              }
            ],
            "LinuxParameters": {
              "InitProcessEnabled": {"Ref": "InitProcessEnabled"}
            },
            "MountPoints": [
              {
                "SourceVolume": {"Ref": "EfsVolumeName"},
                "ContainerPath": {"Ref": "EfsContainerPath"}
              }
            ],
```

```
            "LogConfiguration": {
              "LogDriver": "awslogs",
              "Options": {
                "mode": "non-blocking",
                "max-buffer-size": "25m",
                "awslogs-group": {"Ref": "LogGroup"},
                "awslogs-region": {"Ref": "AWS::Region"},
                "awslogs-create-group": "true",
                "awslogs-stream-prefix": {"Ref": "LogStreamPrefix"}
              }
            }
          }
        ],
        "Volumes": [
          {
            "Name": {"Ref": "EfsVolumeName"},
            "EFSVolumeConfiguration": {
              "FilesystemId": {"Ref": "EfsFilesystemId"},
              "RootDirectory": {"Ref": "EfsRootDirectory"},
              "TransitEncryption": {"Ref": "EfsTransitEncryption"}
            }
          }
        ]
      }
    }
  },
  "Outputs": {
    "TaskDefinitionArn": {
      "Description": "The ARN of the created task definition",
      "Value": {"Ref": "ECSTaskDefinition"}
    }
  }
}
```

YAML

```
AWSTemplateFormatVersion: 2010-09-09
Description: Create a task definition for a web server with parameterized values.
Parameters:
  ExecutionRoleArn:
    Type: String
    Default: arn:aws:iam::123456789012:role/ecsTaskExecutionRole
    Description: The ARN of the task execution role
```

```
  NetworkMode:
    Type: String
    Default: awsvpc
    Description: The Docker networking mode to use
    AllowedValues: [awsvpc, bridge, host, none]
  TaskFamily:
    Type: String
    Default: my-ecs-task
    Description: The name of the task definition family
  ContainerCpu:
    Type: String
    Default: "256"
    Description: The number of CPU units to reserve for the container
    AllowedValues: ["256", "512", "1024", "2048", "4096"]
  ContainerMemory:
    Type: String
    Default: "512"
    Description: The amount of memory (in MiB) to reserve for the container
    AllowedValues: ["512", "1024", "2048", "3072", "4096", "5120", "6144", "7168",
"8192"]
  ContainerName:
    Type: String
    Default: nginx
    Description: The name of the container
  ContainerImage:
    Type: String
    Default: public.ecr.aws/nginx/nginx:latest
    Description: The container image to use for the task
  ContainerPort:
    Type: Number
    Default: 80
    Description: The port number on the container
  InitProcessEnabled:
    Type: String
    Default: "true"
    Description: Whether to enable the init process inside the container
    AllowedValues: ["true", "false"]
  EfsVolumeName:
    Type: String
    Default: efs-volume
    Description: The name of the EFS volume
  EfsContainerPath:
    Type: String
    Default: /usr/share/nginx/html
```

```
    Description: The path in the container where the EFS volume will be mounted
  LogGroup:
    Type: String
    Default: LogGroup
    Description: The CloudWatch log group for container logs
  LogStreamPrefix:
    Type: String
    Default: efs-task
    Description: The prefix for the log stream
  EfsFilesystemId:
    Type: String
    Default: fs-1234567890abcdef0
    Description: The ID of the EFS filesystem
  EfsRootDirectory:
    Type: String
    Default: /
    Description: The root directory in the EFS filesystem
  EfsTransitEncryption:
    Type: String
    Default: ENABLED
    Description: Whether to enable transit encryption for EFS
    AllowedValues: [ENABLED, DISABLED]
Resources:
  ECSTaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      ExecutionRoleArn: !Ref ExecutionRoleArn
      NetworkMode: !Ref NetworkMode
      RequiresCompatibilities:
        - FARGATE
      Family: !Ref TaskFamily
      Cpu: !Ref ContainerCpu
      Memory: !Ref ContainerMemory
      ContainerDefinitions:
        - Name: !Ref ContainerName
          Image: !Ref ContainerImage
          Essential: true
          PortMappings:
            - ContainerPort: !Ref ContainerPort
              Protocol: tcp
          LinuxParameters:
            InitProcessEnabled: !Ref InitProcessEnabled
          MountPoints:
            - SourceVolume: !Ref EfsVolumeName
```

```
              ContainerPath: !Ref EfsContainerPath
          LogConfiguration:
            LogDriver: awslogs
            Options:
              mode: non-blocking
              max-buffer-size: 25m
              awslogs-group: !Ref LogGroup
              awslogs-region: !Ref AWS::Region
              awslogs-create-group: "true"
              awslogs-stream-prefix: !Ref LogStreamPrefix
      Volumes:
        - Name: !Ref EfsVolumeName
          EFSVolumeConfiguration:
            FilesystemId: !Ref EfsFilesystemId
            RootDirectory: !Ref EfsRootDirectory
            TransitEncryption: !Ref EfsTransitEncryption
Outputs:
  TaskDefinitionArn:
    Description: The ARN of the created task definition
    Value: !Ref ECSTaskDefinition
```

キャパシティープロバイダー

キャパシティプロバイダーは Amazon ECS クラスターに関連付けられ、ワークロードのコンピューティングキャパシティを管理するために使用されます。

Amazon ECS マネージドインスタンスのキャパシティプロバイダーを作成する

デフォルトでは、Amazon ECS は、最もコスト最適化された汎用インスタンスタイプを自動的に選択するキャパシティプロバイダーを備えています。ただし、カスタムキャパシティプロバイダーを作成して、インスタンスタイプ、CPU 製造元、アクセラレータータイプ、その他の要件などのインスタンス要件を指定することもできます。次のテンプレートを使用して、指定されたメモリと CPU の要件を満たす Amazon ECS マネージドインスタンスのキャパシティプロバイダーを作成できます。

JSON

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Resources": {
        "MyCapacityProvider": {
            "Type": "AWS::ECS::CapacityProvider",
```

```
            "Properties": {
                "ManagedInstancesProvider": {
                    "InfrastructureRoleArn": "arn:aws:iam::123456789012:role/
ECSInfrastructureRole",
                    "InstanceLaunchTemplate": {
                        "Ec2InstanceProfileArn":
 "arn:aws:iam::123456789012:instance-profile/ecsInstanceProfile",
                        "NetworkConfiguration": null,
                        "Subnets": [
                            "subnet-12345678"
                        ],
                        "SecurityGroups": [
                            "sg-87654321"
                        ]
                    },
                    "StorageConfiguration": {
                        "StorageSizeGiB": 30
                    },
                    "InstanceRequirements": {
                        "VCpuCount": {
                            "Min": 1,
                            "Max": 4
                        },
                        "MemoryMiB": {
                            "Min": 2048,
                            "Max": 8192
                        }
                    }
                }
            }
        }
    }
}
```

YAML

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  MyCapacityProvider:
    Type: AWS::ECS::CapacityProvider
    Properties:
      ManagedInstancesProvider:
```

```
        InfrastructureRoleArn: "arn:aws:iam::123456789012:role/
ECSInfrastructureRole"
        InstanceLaunchTemplate:
          Ec2InstanceProfileArn: "arn:aws:iam::123456789012:instance-profile/
ecsInstanceProfile"
          NetworkConfiguration:
          Subnets:
            - "subnet-12345678"
          SecurityGroups:
            - "sg-87654321"
        StorageConfiguration:
          StorageSizeGiB: 30
        InstanceRequirements:
          VCpuCount:
            Min: 1
            Max: 4
          MemoryMiB:
            Min: 2048
            Max: 8192
```

クラスター

Amazon ECS クラスターは、タスクまたはサービスの論理グループです。次のテンプレートを使用して、さまざまな設定のクラスターを作成できます。Amazon ECS クラスターの詳細については、「Amazon ECS クラスター」を参照してください。

デフォルト設定で空のクラスターを作成する

以下のテンプレートを使用して、デフォルト設定で空のクラスターを作成できます。

JSON

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Resources": {
        "ECSCluster": {
            "Type": "AWS::ECS::Cluster",
            "Properties": {
                "ClusterName": "MyEmptyCluster"
            }
        }
    }
```

```
}
```

YAML

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  ECSCluster:
    Type: 'AWS::ECS::Cluster'
    Properties:
      ClusterName: MyEmptyCluster
```

マネージドストレージ暗号化と拡張 Container Insights を使用して空のクラスターを作成する

次のテンプレートを使用して、クラスターレベルのマネージドストレージと拡張 Container Insights を有効にしたクラスターを作成できます。クラスターレベルの暗号化は、Amazon EBS ボリュームなどの Amazon ECS マネージド型データボリュームに適用されます。Amazon EBS の暗号化の詳細については、「Amazon ECS タスクにアタッチされた Amazon EBS ボリュームに保存されているデータの暗号化」を参照してください。拡張オブザーバビリティで Container Insights を使用する方法の詳細については、「オブザーバビリティが強化された Container Insights を使用し、Amazon ECS コンテナを監視する」を参照してください。

JSON

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
   "Resources": {
        "Cluster": {
            "Type": "AWS::ECS::Cluster",
            "Properties": {
                "ClusterName": "EncryptedEnhancedCluster",
                "ClusterSettings": [
                    {
                        "Name": "containerInsights",
                        "Value": "enhanced"
                    }
                ],
                "Configuration": {
                    "ManagedStorageConfiguration": {
                        "KmsKeyId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

```
                    }
                }
            }
        }
    }
}
```

YAML

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  Cluster:
    Type: AWS::ECS::Cluster
    Properties:
      ClusterName: EncryptedEnhancedCluster
      ClusterSettings:
        - Name: containerInsights
          Value: enhanced
      Configuration:
        ManagedStorageConfiguration:
          KmsKeyId: a1b2c3d4-5678-90ab-cdef-EXAMPLE11111
```

AL2023 Amazon ECS-Optimized-AMI を使用してクラスターを作成する

以下のテンプレートを用いて、キャパシティープロバイダーを使用して Amazon EC2 で AL2023 インスタンスを起動するクラスターを定義できます。

> **Important**
>
> 最新の AMI ID については、「Amazon Elastic Container Service デベロッパーガイド」の「Amazon ECS-optimized AMI」(Amazon ECS に最適化された AMI) を参照してください。

JSON

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "EC2 ECS cluster that starts out empty, with no EC2 instances
 yet. An ECS capacity provider automatically launches more EC2 instances as required
 on the fly when you request ECS to launch services or standalone tasks.",
    "Parameters": {
```

```
        "InstanceType": {
            "Type": "String",
            "Description": "EC2 instance type",
            "Default": "t2.medium",
            "AllowedValues": [
                "t1.micro",
                "t2.2xlarge",
                "t2.large",
                "t2.medium",
                "t2.micro",
                "t2.nano",
                "t2.small",
                "t2.xlarge",
                "t3.2xlarge",
                "t3.large",
                "t3.medium",
                "t3.micro",
                "t3.nano",
                "t3.small",
                "t3.xlarge"
            ]
        },
        "DesiredCapacity": {
            "Type": "Number",
            "Default": "0",
            "Description": "Number of EC2 instances to launch in your ECS cluster."
        },
        "MaxSize": {
            "Type": "Number",
            "Default": "100",
            "Description": "Maximum number of EC2 instances that can be launched in
 your ECS cluster."
        },
        "ECSAMI": {
            "Description": "The Amazon Machine Image ID used for the cluster",
            "Type": "AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>",
            "Default": "/aws/service/ecs/optimized-ami/amazon-linux-2023/
recommended/image_id"
        },
        "VpcId": {
            "Type": "AWS::EC2::VPC::Id",
            "Description": "VPC ID where the ECS cluster is launched",
            "Default": "vpc-1234567890abcdef0"
        },
```

```
        "SubnetIds": {
            "Type": "List<AWS::EC2::Subnet::Id>",
            "Description": "List of subnet IDs where the EC2 instances will be
launched",
            "Default": "subnet-021345abcdef67890"
        }
    },
    "Resources": {
        "ECSCluster": {
            "Type": "AWS::ECS::Cluster",
            "Properties": {
                "ClusterSettings": [
                    {
                        "Name": "containerInsights",
                        "Value": "enabled"
                    }
                ]
            }
        },
        "ECSAutoScalingGroup": {
            "Type": "AWS::AutoScaling::AutoScalingGroup",
            "DependsOn": [
                "ECSCluster",
                "EC2Role"
            ],
            "Properties": {
                "VPCZoneIdentifier": {
                    "Ref": "SubnetIds"
                },
                "LaunchTemplate": {
                    "LaunchTemplateId": {
                        "Ref": "ContainerInstances"
                    },
                    "Version": {
                        "Fn::GetAtt": [
                            "ContainerInstances",
                            "LatestVersionNumber"
                        ]
                    }
                },
                "MinSize": 0,
                "MaxSize": {
                    "Ref": "MaxSize"
                },
```

```
                "DesiredCapacity": {
                    "Ref": "DesiredCapacity"
                },
                "NewInstancesProtectedFromScaleIn": true
            },
            "UpdatePolicy": {
                "AutoScalingReplacingUpdate": {
                    "WillReplace": "true"
                }
            }
        },
        "ContainerInstances": {
            "Type": "AWS::EC2::LaunchTemplate",
            "Properties": {
                "LaunchTemplateName": "asg-launch-template-2",
                "LaunchTemplateData": {
                    "ImageId": {
                        "Ref": "ECSAMI"
                    },
                    "InstanceType": {
                        "Ref": "InstanceType"
                    },
                    "IamInstanceProfile": {
                        "Name": {
                            "Ref": "EC2InstanceProfile"
                        }
                    },
                    "SecurityGroupIds": [
                        {
                            "Ref": "ContainerHostSecurityGroup"
                        }
                    ],
                    "UserData": {
                        "Fn::Base64": {
                            "Fn::Sub": "#!/bin/bash -xe\n echo ECS_CLUSTER=
${ECSCluster} >> /etc/ecs/ecs.config\n yum install -y aws-cfn-bootstrap\n /opt/aws/
bin/cfn-init -v --stack ${AWS::StackId} --resource ContainerInstances --configsets
 full_install --region ${AWS::Region} &\n"
                        }
                    },
                    "MetadataOptions": {
                        "HttpEndpoint": "enabled",
                        "HttpTokens": "required"
                    }
```

```
                }
            }
        },
        "EC2InstanceProfile": {
            "Type": "AWS::IAM::InstanceProfile",
            "Properties": {
                "Path": "/",
                "Roles": [
                    {
                        "Ref": "EC2Role"
                    }
                ]
            }
        },
        "CapacityProvider": {
            "Type": "AWS::ECS::CapacityProvider",
            "Properties": {
                "AutoScalingGroupProvider": {
                    "AutoScalingGroupArn": {
                        "Ref": "ECSAutoScalingGroup"
                    },
                    "ManagedScaling": {
                        "InstanceWarmupPeriod": 60,
                        "MinimumScalingStepSize": 1,
                        "MaximumScalingStepSize": 100,
                        "Status": "ENABLED",
                        "TargetCapacity": 100
                    },
                    "ManagedTerminationProtection": "ENABLED"
                }
            }
        },
        "CapacityProviderAssociation": {
            "Type": "AWS::ECS::ClusterCapacityProviderAssociations",
            "Properties": {
                "CapacityProviders": [
                    {
                        "Ref": "CapacityProvider"
                    }
                ],
                "Cluster": {
                    "Ref": "ECSCluster"
                },
                "DefaultCapacityProviderStrategy": [
```

```
                    {
                        "Base": 0,
                        "CapacityProvider": {
                            "Ref": "CapacityProvider"
                        },
                        "Weight": 1
                    }
                ]
            }
        },
        "ContainerHostSecurityGroup": {
            "Type": "AWS::EC2::SecurityGroup",
            "Properties": {
                "GroupDescription": "Access to the EC2 hosts that run containers",
                "VpcId": {
                    "Ref": "VpcId"
                }
            }
        },
        "EC2Role": {
            "Type": "AWS::IAM::Role",
            "Properties": {
                "AssumeRolePolicyDocument": {
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {
                                "Service": [
                                    "ec2.amazonaws.com"
                                ]
                            },
                            "Action": [
                                "sts:AssumeRole"
                            ]
                        }
                    ]
                },
                "Path": "/",
                "ManagedPolicyArns": [
                    "arn:aws:iam::aws:policy/service-role/
AmazonEC2ContainerServiceforEC2Role",
                    "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore"
                ]
            }
```

```
        },
        "ECSTaskExecutionRole": {
            "Type": "AWS::IAM::Role",
            "Properties": {
                "AssumeRolePolicyDocument": {
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {
                                "Service": [
                                    "ecs-tasks.amazonaws.com"
                                ]
                            },
                            "Action": [
                                "sts:AssumeRole"
                            ],
                            "Condition": {
                                "ArnLike": {
                                    "aws:SourceArn": {
                                        "Fn::Sub": "arn:${AWS::Partition}:ecs:
${AWS::Region}:${AWS::AccountId}:*"
                                    }
                                },
                                "StringEquals": {
                                    "aws:SourceAccount": {
                                        "Fn::Sub": "${AWS::AccountId}"
                                    }
                                }
                            }
                        }
                    ]
                },
                "Path": "/",
                "ManagedPolicyArns": [
                    "arn:aws:iam::aws:policy/service-role/
AmazonECSTaskExecutionRolePolicy"
                ]
            }
        }
    },
    "Outputs": {
        "ClusterName": {
            "Description": "The ECS cluster into which to launch resources",
            "Value": "ECSCluster"
```

```
        },
        "ECSTaskExecutionRole": {
            "Description": "The role used to start up a task",
            "Value": "ECSTaskExecutionRole"
        },
        "CapacityProvider": {
            "Description": "The cluster capacity provider that the service should
 use to request capacity when it wants to start up a task",
            "Value": "CapacityProvider"
        }
    }
}
```

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Description: EC2 ECS cluster that starts out empty, with no EC2 instances yet. An
 ECS capacity provider automatically launches more EC2 instances as required on the
 fly when you request ECS to launch services or standalone tasks.
Parameters:
  InstanceType:
    Type: String
    Description: EC2 instance type
    Default: t2.medium
    AllowedValues:
      - t1.micro
      - t2.2xlarge
      - t2.large
      - t2.medium
      - t2.micro
      - t2.nano
      - t2.small
      - t2.xlarge
      - t3.2xlarge
      - t3.large
      - t3.medium
      - t3.micro
      - t3.nano
      - t3.small
      - t3.xlarge
  DesiredCapacity:
    Type: Number
    Default: '0'
```

```
    Description: Number of EC2 instances to launch in your ECS cluster.
  MaxSize:
    Type: Number
    Default: '100'
    Description: Maximum number of EC2 instances that can be launched in your ECS
 cluster.
  ECSAMI:
    Description: The Amazon Machine Image ID used for the cluster
    Type: AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>
    Default: /aws/service/ecs/optimized-ami/amazon-linux-2023/recommended/image_id
  VpcId:
    Type: AWS::EC2::VPC::Id
    Description: VPC ID where the ECS cluster is launched
    Default: vpc-1234567890abcdef0
  SubnetIds:
    Type: List<AWS::EC2::Subnet::Id>
    Description: List of subnet IDs where the EC2 instances will be launched
    Default: subnet-021345abcdef67890
Resources:
  ECSCluster:
    Type: AWS::ECS::Cluster
    Properties:
      ClusterSettings:
        - Name: containerInsights
          Value: enabled
  ECSAutoScalingGroup:
    Type: AWS::AutoScaling::AutoScalingGroup
    DependsOn:
      - ECSCluster
      - EC2Role
    Properties:
      VPCZoneIdentifier: !Ref SubnetIds
      LaunchTemplate:
        LaunchTemplateId: !Ref ContainerInstances
        Version: !GetAtt ContainerInstances.LatestVersionNumber
      MinSize: 0
      MaxSize: !Ref MaxSize
      DesiredCapacity: !Ref DesiredCapacity
      NewInstancesProtectedFromScaleIn: true
    UpdatePolicy:
      AutoScalingReplacingUpdate:
        WillReplace: 'true'
  ContainerInstances:
    Type: AWS::EC2::LaunchTemplate
```

```
    Properties:
      LaunchTemplateName: asg-launch-template-2
      LaunchTemplateData:
        ImageId: !Ref ECSAMI
        InstanceType: !Ref InstanceType
        IamInstanceProfile:
          Name: !Ref EC2InstanceProfile
        SecurityGroupIds:
          - !Ref ContainerHostSecurityGroup
        UserData: !Base64
          Fn::Sub: |
            #!/bin/bash -xe
            echo ECS_CLUSTER=${ECSCluster} >> /etc/ecs/ecs.config
            yum install -y aws-cfn-bootstrap
            /opt/aws/bin/cfn-init -v --stack ${AWS::StackId} --resource
ContainerInstances --configsets full_install --region ${AWS::Region} &
        MetadataOptions:
          HttpEndpoint: enabled
          HttpTokens: required
  EC2InstanceProfile:
    Type: AWS::IAM::InstanceProfile
    Properties:
      Path: /
      Roles:
        - !Ref EC2Role
  CapacityProvider:
    Type: AWS::ECS::CapacityProvider
    Properties:
      AutoScalingGroupProvider:
        AutoScalingGroupArn: !Ref ECSAutoScalingGroup
        ManagedScaling:
          InstanceWarmupPeriod: 60
          MinimumScalingStepSize: 1
          MaximumScalingStepSize: 100
          Status: ENABLED
          TargetCapacity: 100
        ManagedTerminationProtection: ENABLED
  CapacityProviderAssociation:
    Type: AWS::ECS::ClusterCapacityProviderAssociations
    Properties:
      CapacityProviders:
        - !Ref CapacityProvider
      Cluster: !Ref ECSCluster
      DefaultCapacityProviderStrategy:
```

```
        - Base: 0
          CapacityProvider: !Ref CapacityProvider
          Weight: 1
  ContainerHostSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Access to the EC2 hosts that run containers
      VpcId: !Ref VpcId
  EC2Role:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - ec2.amazonaws.com
            Action:
              - sts:AssumeRole
      Path: /
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AmazonEC2ContainerServiceforEC2Role
        - arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
  ECSTaskExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - ecs-tasks.amazonaws.com
            Action:
              - sts:AssumeRole
            Condition:
              ArnLike:
                aws:SourceArn: !Sub arn:${AWS::Partition}:ecs:${AWS::Region}:
${AWS::AccountId}:*
              StringEquals:
                aws:SourceAccount: !Sub ${AWS::AccountId}
      Path: /
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy
Outputs:
```

```
  ClusterName:
    Description: The ECS cluster into which to launch resources
    Value: ECSCluster
  ECSTaskExecutionRole:
    Description: The role used to start up a task
    Value: ECSTaskExecutionRole
  CapacityProvider:
    Description: The cluster capacity provider that the service should use to
request capacity when it wants to start up a task
    Value: CapacityProvider
```

サービス

Amazon ECS サービスを使用すると、Amazon ECS クラスター内で、タスク定義の指定した数のインスタンスを同時に実行して維持できます。タスクの 1 つが失敗または停止した場合、Amazon ECS サービススケジューラはタスク定義の別のインスタンスを起動してそれを置き換えます。これは、サービスで必要な数のタスクを維持するのに役立ちます。以下のテンプレートを使用して各種サービスをデプロイできます。Amazon ECS サービスの詳細については、「Amazon ECS サービス」を参照してください。

ロードバランシングされたウェブアプリケーションをデプロイする

次のテンプレートは、Fargate で実行する 2 つのタスクを持つ Amazon ECS サービスを作成します。タスクにはそれぞれ NGINX コンテナがあります。テンプレートは、アプリケーショントラフィックを分散する Application Load Balancer と、CPU 使用率に基づいて、アプリケーションをスケーリングする Application Auto Scaling ポリシーも作成します。更に、テンプレートは、アプリケーションのデプロイに必要なネットワークリソース、コンテナログのログ記録リソース、および Amazon ECS タスク実行 IAM ロールも作成します。タスクの実行ロールの詳細については、「Amazon ECS タスク実行IAM ロール」を参照してください。自動スケーリングの詳細については、「Amazon ECS サービスを自動的にスケールする」を参照してください。

JSON

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "[AWSDocs] ECS: load-balanced-web-application",
    "Parameters": {
        "VpcCidr": {
            "Type": "String",
            "Default": "10.0.0.0/16",
```

```
            "Description": "CIDR block for the VPC"
        },
        "ContainerImage": {
            "Type": "String",
            "Default": "public.ecr.aws/ecs-sample-image/amazon-ecs-sample:latest",
            "Description": "Container image to use in task definition"
        },
        "PublicSubnet1Cidr": {
            "Type": "String",
            "Default": "10.0.1.0/24",
            "Description": "CIDR block for public subnet 1"
        },
        "PublicSubnet2Cidr": {
            "Type": "String",
            "Default": "10.0.2.0/24",
            "Description": "CIDR block for public subnet 2"
        },
        "PrivateSubnet1Cidr": {
            "Type": "String",
            "Default": "10.0.3.0/24",
            "Description": "CIDR block for private subnet 1"
        },
        "PrivateSubnet2Cidr": {
            "Type": "String",
            "Default": "10.0.4.0/24",
            "Description": "CIDR block for private subnet 2"
        },
        "ServiceName": {
            "Type": "String",
            "Default": "tutorial-app",
            "Description": "Name of the ECS service"
        },
        "ContainerPort": {
            "Type": "Number",
            "Default": 80,
            "Description": "Port on which the container listens"
        },
        "DesiredCount": {
            "Type": "Number",
            "Default": 2,
            "Description": "Desired number of tasks"
        },
        "MinCapacity": {
            "Type": "Number",
```

```
            "Default": 1,
            "Description": "Minimum number of tasks for auto scaling"
        },
        "MaxCapacity": {
            "Type": "Number",
            "Default": 10,
            "Description": "Maximum number of tasks for auto scaling"
        }
    },
    "Resources": {
        "VPC": {
            "Type": "AWS::EC2::VPC",
            "Properties": {
                "CidrBlock": {
                    "Ref": "VpcCidr"
                },
                "EnableDnsHostnames": true,
                "EnableDnsSupport": true,
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-vpc"
                        }
                    }
                ]
            }
        },
        "InternetGateway": {
            "Type": "AWS::EC2::InternetGateway",
            "Properties": {
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-igw"
                        }
                    }
                ]
            }
        },
        "InternetGatewayAttachment": {
            "Type": "AWS::EC2::VPCGatewayAttachment",
            "Properties": {
```

```
                "InternetGatewayId": {
                    "Ref": "InternetGateway"
                },
                "VpcId": {
                    "Ref": "VPC"
                }
            }
        },
        "PublicSubnet1": {
            "Type": "AWS::EC2::Subnet",
            "Properties": {
                "VpcId": {
                    "Ref": "VPC"
                },
                "AvailabilityZone": {
                    "Fn::Select": [
                        0,
                        {
                            "Fn::GetAZs": ""
                        }
                    ]
                },
                "CidrBlock": {
                    "Ref": "PublicSubnet1Cidr"
                },
                "MapPublicIpOnLaunch": true,
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-public-subnet-1"
                        }
                    }
                ]
            }
        },
        "PublicSubnet2": {
            "Type": "AWS::EC2::Subnet",
            "Properties": {
                "VpcId": {
                    "Ref": "VPC"
                },
                "AvailabilityZone": {
                    "Fn::Select": [
```

```
                    1,
                    {
                        "Fn::GetAZs": ""
                    }
                ]
            },
            "CidrBlock": {
                "Ref": "PublicSubnet2Cidr"
            },
            "MapPublicIpOnLaunch": true,
            "Tags": [
                {
                    "Key": "Name",
                    "Value": {
                        "Fn::Sub": "${AWS::StackName}-public-subnet-2"
                    }
                }
            ]
        }
    },
    "PrivateSubnet1": {
        "Type": "AWS::EC2::Subnet",
        "Properties": {
            "VpcId": {
                "Ref": "VPC"
            },
            "AvailabilityZone": {
                "Fn::Select": [
                    0,
                    {
                        "Fn::GetAZs": ""
                    }
                ]
            },
            "CidrBlock": {
                "Ref": "PrivateSubnet1Cidr"
            },
            "Tags": [
                {
                    "Key": "Name",
                    "Value": {
                        "Fn::Sub": "${AWS::StackName}-private-subnet-1"
                    }
                }
```

```
                ]
            }
        },
        "PrivateSubnet2": {
            "Type": "AWS::EC2::Subnet",
            "Properties": {
                "VpcId": {
                    "Ref": "VPC"
                },
                "AvailabilityZone": {
                    "Fn::Select": [
                        1,
                        {
                            "Fn::GetAZs": ""
                        }
                    ]
                },
                "CidrBlock": {
                    "Ref": "PrivateSubnet2Cidr"
                },
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-private-subnet-2"
                        }
                    }
                ]
            }
        },
        "NatGateway1EIP": {
            "Type": "AWS::EC2::EIP",
            "DependsOn": "InternetGatewayAttachment",
            "Properties": {
                "Domain": "vpc",
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-nat-eip-1"
                        }
                    }
                ]
            }
```

```
        },
        "NatGateway2EIP": {
            "Type": "AWS::EC2::EIP",
            "DependsOn": "InternetGatewayAttachment",
            "Properties": {
                "Domain": "vpc",
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-nat-eip-2"
                        }
                    }
                ]
            }
        },
        "NatGateway1": {
            "Type": "AWS::EC2::NatGateway",
            "Properties": {
                "AllocationId": {
                    "Fn::GetAtt": [
                        "NatGateway1EIP",
                        "AllocationId"
                    ]
                },
                "SubnetId": {
                    "Ref": "PublicSubnet1"
                },
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-nat-1"
                        }
                    }
                ]
            }
        },
        "NatGateway2": {
            "Type": "AWS::EC2::NatGateway",
            "Properties": {
                "AllocationId": {
                    "Fn::GetAtt": [
                        "NatGateway2EIP",
```

```
                    "AllocationId"
                ]
            },
            "SubnetId": {
                "Ref": "PublicSubnet2"
            },
            "Tags": [
                {
                    "Key": "Name",
                    "Value": {
                        "Fn::Sub": "${AWS::StackName}-nat-2"
                    }
                }
            ]
        }
    },
    "PublicRouteTable": {
        "Type": "AWS::EC2::RouteTable",
        "Properties": {
            "VpcId": {
                "Ref": "VPC"
            },
            "Tags": [
                {
                    "Key": "Name",
                    "Value": {
                        "Fn::Sub": "${AWS::StackName}-public-routes"
                    }
                }
            ]
        }
    },
    "DefaultPublicRoute": {
        "Type": "AWS::EC2::Route",
        "DependsOn": "InternetGatewayAttachment",
        "Properties": {
            "RouteTableId": {
                "Ref": "PublicRouteTable"
            },
            "DestinationCidrBlock": "0.0.0.0/0",
            "GatewayId": {
                "Ref": "InternetGateway"
            }
        }
```

```
        },
        "PublicSubnet1RouteTableAssociation": {
            "Type": "AWS::EC2::SubnetRouteTableAssociation",
            "Properties": {
                "RouteTableId": {
                    "Ref": "PublicRouteTable"
                },
                "SubnetId": {
                    "Ref": "PublicSubnet1"
                }
            }
        },
        "PublicSubnet2RouteTableAssociation": {
            "Type": "AWS::EC2::SubnetRouteTableAssociation",
            "Properties": {
                "RouteTableId": {
                    "Ref": "PublicRouteTable"
                },
                "SubnetId": {
                    "Ref": "PublicSubnet2"
                }
            }
        },
        "PrivateRouteTable1": {
            "Type": "AWS::EC2::RouteTable",
            "Properties": {
                "VpcId": {
                    "Ref": "VPC"
                },
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-private-routes-1"
                        }
                    }
                ]
            }
        },
        "DefaultPrivateRoute1": {
            "Type": "AWS::EC2::Route",
            "Properties": {
                "RouteTableId": {
                    "Ref": "PrivateRouteTable1"
```

```
                },
                "DestinationCidrBlock": "0.0.0.0/0",
                "NatGatewayId": {
                    "Ref": "NatGateway1"
                }
            }
        },
        "PrivateSubnet1RouteTableAssociation": {
            "Type": "AWS::EC2::SubnetRouteTableAssociation",
            "Properties": {
                "RouteTableId": {
                    "Ref": "PrivateRouteTable1"
                },
                "SubnetId": {
                    "Ref": "PrivateSubnet1"
                }
            }
        },
        "PrivateRouteTable2": {
            "Type": "AWS::EC2::RouteTable",
            "Properties": {
                "VpcId": {
                    "Ref": "VPC"
                },
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-private-routes-2"
                        }
                    }
                ]
            }
        },
        "DefaultPrivateRoute2": {
            "Type": "AWS::EC2::Route",
            "Properties": {
                "RouteTableId": {
                    "Ref": "PrivateRouteTable2"
                },
                "DestinationCidrBlock": "0.0.0.0/0",
                "NatGatewayId": {
                    "Ref": "NatGateway2"
                }
```

```
            }
        },
        "PrivateSubnet2RouteTableAssociation": {
            "Type": "AWS::EC2::SubnetRouteTableAssociation",
            "Properties": {
                "RouteTableId": {
                    "Ref": "PrivateRouteTable2"
                },
                "SubnetId": {
                    "Ref": "PrivateSubnet2"
                }
            }
        },
        "ALBSecurityGroup": {
            "Type": "AWS::EC2::SecurityGroup",
            "Properties": {
                "GroupName": {
                    "Fn::Sub": "${AWS::StackName}-alb-sg"
                },
                "GroupDescription": "Security group for Application Load Balancer",
                "VpcId": {
                    "Ref": "VPC"
                },
                "SecurityGroupIngress": [
                    {
                        "IpProtocol": "tcp",
                        "FromPort": 80,
                        "ToPort": 80,
                        "CidrIp": "0.0.0.0/0",
                        "Description": "Allow HTTP traffic from internet"
                    }
                ],
                "SecurityGroupEgress": [
                    {
                        "IpProtocol": -1,
                        "CidrIp": "0.0.0.0/0",
                        "Description": "Allow all outbound traffic"
                    }
                ],
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-alb-sg"
```

```
                    }
                }
            ]
        }
    },
    "ECSSecurityGroup": {
        "Type": "AWS::EC2::SecurityGroup",
        "Properties": {
            "GroupName": {
                "Fn::Sub": "${AWS::StackName}-ecs-sg"
            },
            "GroupDescription": "Security group for ECS tasks",
            "VpcId": {
                "Ref": "VPC"
            },
            "SecurityGroupIngress": [
                {
                    "IpProtocol": "tcp",
                    "FromPort": {
                        "Ref": "ContainerPort"
                    },
                    "ToPort": {
                        "Ref": "ContainerPort"
                    },
                    "SourceSecurityGroupId": {
                        "Ref": "ALBSecurityGroup"
                    },
                    "Description": "Allow traffic from ALB"
                }
            ],
            "SecurityGroupEgress": [
                {
                    "IpProtocol": -1,
                    "CidrIp": "0.0.0.0/0",
                    "Description": "Allow all outbound traffic"
                }
            ],
            "Tags": [
                {
                    "Key": "Name",
                    "Value": {
                        "Fn::Sub": "${AWS::StackName}-ecs-sg"
                    }
                }
```

```
                ]
            }
        },
        "ApplicationLoadBalancer": {
            "Type": "AWS::ElasticLoadBalancingV2::LoadBalancer",
            "Properties": {
                "Name": {
                    "Fn::Sub": "${AWS::StackName}-alb"
                },
                "Scheme": "internet-facing",
                "Type": "application",
                "Subnets": [
                    {
                        "Ref": "PublicSubnet1"
                    },
                    {
                        "Ref": "PublicSubnet2"
                    }
                ],
                "SecurityGroups": [
                    {
                        "Ref": "ALBSecurityGroup"
                    }
                ],
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-alb"
                        }
                    }
                ]
            }
        },
        "ALBTargetGroup": {
            "Type": "AWS::ElasticLoadBalancingV2::TargetGroup",
            "Properties": {
                "Name": {
                    "Fn::Sub": "${AWS::StackName}-tg"
                },
                "Port": {
                    "Ref": "ContainerPort"
                },
                "Protocol": "HTTP",
```

```
                "VpcId": {
                    "Ref": "VPC"
                },
                "TargetType": "ip",
                "HealthCheckIntervalSeconds": 30,
                "HealthCheckPath": "/",
                "HealthCheckProtocol": "HTTP",
                "HealthCheckTimeoutSeconds": 5,
                "HealthyThresholdCount": 2,
                "UnhealthyThresholdCount": 5,
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-tg"
                        }
                    }
                ]
            }
        },
        "ALBListener": {
            "Type": "AWS::ElasticLoadBalancingV2::Listener",
            "Properties": {
                "DefaultActions": [
                    {
                        "Type": "forward",
                        "TargetGroupArn": {
                            "Ref": "ALBTargetGroup"
                        }
                    }
                ],
                "LoadBalancerArn": {
                    "Ref": "ApplicationLoadBalancer"
                },
                "Port": 80,
                "Protocol": "HTTP"
            }
        },
        "ECSCluster": {
            "Type": "AWS::ECS::Cluster",
            "Properties": {
                "ClusterName": {
                    "Fn::Sub": "${AWS::StackName}-cluster"
                },
```

```
                "CapacityProviders": [
                    "FARGATE",
                    "FARGATE_SPOT"
                ],
                "DefaultCapacityProviderStrategy": [
                    {
                        "CapacityProvider": "FARGATE",
                        "Weight": 1
                    },
                    {
                        "CapacityProvider": "FARGATE_SPOT",
                        "Weight": 4
                    }
                ],
                "ClusterSettings": [
                    {
                        "Name": "containerInsights",
                        "Value": "enabled"
                    }
                ],
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-cluster"
                        }
                    }
                ]
            }
        },
        "ECSTaskExecutionRole": {
            "Type": "AWS::IAM::Role",
            "Properties": {
                "RoleName": {
                    "Fn::Sub": "${AWS::StackName}-task-execution-role"
                },
                "AssumeRolePolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {
                                "Service": "ecs-tasks.amazonaws.com"
                            },
```

```
                            "Action": "sts:AssumeRole"
                        }
                    ]
                },
                "ManagedPolicyArns": [
                    "arn:aws:iam::aws:policy/service-role/
AmazonECSTaskExecutionRolePolicy"
                ],
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-task-execution-role"
                        }
                    }
                ]
            }
        },
        "ECSTaskRole": {
            "Type": "AWS::IAM::Role",
            "Properties": {
                "RoleName": {
                    "Fn::Sub": "${AWS::StackName}-task-role"
                },
                "AssumeRolePolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {
                                "Service": "ecs-tasks.amazonaws.com"
                            },
                            "Action": "sts:AssumeRole"
                        }
                    ]
                },
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-task-role"
                        }
                    }
                ]
```

```
            }
        },
        "LogGroup": {
            "Type": "AWS::Logs::LogGroup",
            "Properties": {
                "LogGroupName": {
                    "Fn::Sub": "/ecs/${AWS::StackName}"
                },
                "RetentionInDays": 7
            }
        },
        "TaskDefinition": {
            "Type": "AWS::ECS::TaskDefinition",
            "Properties": {
                "Family": {
                    "Fn::Sub": "${AWS::StackName}-task"
                },
                "Cpu": "256",
                "Memory": "512",
                "NetworkMode": "awsvpc",
                "RequiresCompatibilities": [
                    "FARGATE"
                ],
                "ExecutionRoleArn": {
                    "Fn::GetAtt": [
                        "ECSTaskExecutionRole",
                        "Arn"
                    ]
                },
                "TaskRoleArn": {
                    "Fn::GetAtt": [
                        "ECSTaskRole",
                        "Arn"
                    ]
                },
                "ContainerDefinitions": [
                    {
                        "Name": {
                            "Ref": "ServiceName"
                        },
                        "Image": {
                            "Ref": "ContainerImage"
                        },
                        "PortMappings": [
```

```
                        {
                            "ContainerPort": {
                                "Ref": "ContainerPort"
                            },
                            "Protocol": "tcp"
                        }
                    ],
                    "Essential": true,
                    "LogConfiguration": {
                        "LogDriver": "awslogs",
                        "Options": {
                            "awslogs-group": {
                                "Ref": "LogGroup"
                            },
                            "awslogs-region": {
                                "Ref": "AWS::Region"
                            },
                            "awslogs-stream-prefix": "ecs"
                        }
                    },
                    "HealthCheck": {
                        "Command": [
                            "CMD-SHELL",
                            "curl -f http://localhost/ || exit 1"
                        ],
                        "Interval": 30,
                        "Timeout": 5,
                        "Retries": 3,
                        "StartPeriod": 60
                    }
                }
            ],
            "Tags": [
                {
                    "Key": "Name",
                    "Value": {
                        "Fn::Sub": "${AWS::StackName}-task"
                    }
                }
            ]
        }
    },
    "ECSService": {
        "Type": "AWS::ECS::Service",
```

```
            "DependsOn": "ALBListener",
            "Properties": {
                "ServiceName": {
                    "Fn::Sub": "${AWS::StackName}-service"
                },
                "Cluster": {
                    "Ref": "ECSCluster"
                },
                "TaskDefinition": {
                    "Ref": "TaskDefinition"
                },
                "DesiredCount": {
                    "Ref": "DesiredCount"
                },
                "LaunchType": "FARGATE",
                "PlatformVersion": "LATEST",
                "NetworkConfiguration": {
                    "AwsvpcConfiguration": {
                        "AssignPublicIp": "DISABLED",
                        "SecurityGroups": [
                            {
                                "Ref": "ECSSecurityGroup"
                            }
                        ],
                        "Subnets": [
                            {
                                "Ref": "PrivateSubnet1"
                            },
                            {
                                "Ref": "PrivateSubnet2"
                            }
                        ]
                    }
                },
                "LoadBalancers": [
                    {
                        "ContainerName": {
                            "Ref": "ServiceName"
                        },
                        "ContainerPort": {
                            "Ref": "ContainerPort"
                        },
                        "TargetGroupArn": {
                            "Ref": "ALBTargetGroup"
```

```
                        }
                    }
                ],
                "DeploymentConfiguration": {
                    "MaximumPercent": 200,
                    "MinimumHealthyPercent": 50,
                    "DeploymentCircuitBreaker": {
                        "Enable": true,
                        "Rollback": true
                    }
                },
                "EnableExecuteCommand": true,
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-service"
                        }
                    }
                ]
            }
        },
        "ServiceScalingTarget": {
            "Type": "AWS::ApplicationAutoScaling::ScalableTarget",
            "Properties": {
                "MaxCapacity": {
                    "Ref": "MaxCapacity"
                },
                "MinCapacity": {
                    "Ref": "MinCapacity"
                },
                "ResourceId": {
                    "Fn::Sub": "service/${ECSCluster}/${ECSService.Name}"
                },
                "RoleARN": {
                    "Fn::Sub": "arn:aws:iam::${AWS::AccountId}:role/
aws-service-role/ecs.application-autoscaling.amazonaws.com/
AWSServiceRoleForApplicationAutoScaling_ECSService"
                },
                "ScalableDimension": "ecs:service:DesiredCount",
                "ServiceNamespace": "ecs"
            }
        },
        "ServiceScalingPolicy": {
```

```
            "Type": "AWS::ApplicationAutoScaling::ScalingPolicy",
            "Properties": {
                "PolicyName": {
                    "Fn::Sub": "${AWS::StackName}-cpu-scaling-policy"
                },
                "PolicyType": "TargetTrackingScaling",
                "ScalingTargetId": {
                    "Ref": "ServiceScalingTarget"
                },
                "TargetTrackingScalingPolicyConfiguration": {
                    "PredefinedMetricSpecification": {
                        "PredefinedMetricType": "ECSServiceAverageCPUUtilization"
                    },
                    "TargetValue": 70,
                    "ScaleOutCooldown": 300,
                    "ScaleInCooldown": 300
                }
            }
        }
    },
    "Outputs": {
        "VPCId": {
            "Description": "VPC ID",
            "Value": {
                "Ref": "VPC"
            },
            "Export": {
                "Name": {
                    "Fn::Sub": "${AWS::StackName}-VPC-ID"
                }
            }
        },
        "LoadBalancerURL": {
            "Description": "URL of the Application Load Balancer",
            "Value": {
                "Fn::Sub": "http://${ApplicationLoadBalancer.DNSName}"
            },
            "Export": {
                "Name": {
                    "Fn::Sub": "${AWS::StackName}-ALB-URL"
                }
            }
        },
        "ECSClusterName": {
```

```
            "Description": "Name of the ECS Cluster",
            "Value": {
                "Ref": "ECSCluster"
            },
            "Export": {
                "Name": {
                    "Fn::Sub": "${AWS::StackName}-ECS-Cluster"
                }
            }
        },
        "ECSServiceName": {
            "Description": "Name of the ECS Service",
            "Value": {
                "Fn::GetAtt": [
                    "ECSService",
                    "Name"
                ]
            },
            "Export": {
                "Name": {
                    "Fn::Sub": "${AWS::StackName}-ECS-Service"
                }
            }
        },
        "PrivateSubnet1": {
            "Description": "Private Subnet 1 ID",
            "Value": {
                "Ref": "PrivateSubnet1"
            },
            "Export": {
                "Name": {
                    "Fn::Sub": "${AWS::StackName}-Private-Subnet-1"
                }
            }
        },
        "PrivateSubnet2": {
            "Description": "Private Subnet 2 ID",
            "Value": {
                "Ref": "PrivateSubnet2"
            },
            "Export": {
                "Name": {
                    "Fn::Sub": "${AWS::StackName}-Private-Subnet-2"
                }
```

```
            }
        }
    }
}
```

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Description: '[AWSDocs] ECS: load-balanced-web-application'

Parameters:
  VpcCidr:
    Type: String
    Default: '10.0.0.0/16'
    Description: CIDR block for the VPC
  ContainerImage:
    Type: String
    Default: 'public.ecr.aws/ecs-sample-image/amazon-ecs-sample:latest'
    Description: Container image to use in task definition

  PublicSubnet1Cidr:
    Type: String
    Default: '10.0.1.0/24'
    Description: CIDR block for public subnet 1

  PublicSubnet2Cidr:
    Type: String
    Default: '10.0.2.0/24'
    Description: CIDR block for public subnet 2

  PrivateSubnet1Cidr:
    Type: String
    Default: '10.0.3.0/24'
    Description: CIDR block for private subnet 1

  PrivateSubnet2Cidr:
    Type: String
    Default: '10.0.4.0/24'
    Description: CIDR block for private subnet 2

  ServiceName:
    Type: String
    Default: 'tutorial-app'
```

```
    Description: Name of the ECS service

  ContainerPort:
    Type: Number
    Default: 80
    Description: Port on which the container listens

  DesiredCount:
    Type: Number
    Default: 2
    Description: Desired number of tasks

  MinCapacity:
    Type: Number
    Default: 1
    Description: Minimum number of tasks for auto scaling

  MaxCapacity:
    Type: Number
    Default: 10
    Description: Maximum number of tasks for auto scaling

Resources:
  # VPC and Networking
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: !Ref VpcCidr
      EnableDnsHostnames: true
      EnableDnsSupport: true
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-vpc'

  # Internet Gateway
  InternetGateway:
    Type: AWS::EC2::InternetGateway
    Properties:
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-igw'

  InternetGatewayAttachment:
    Type: AWS::EC2::VPCGatewayAttachment
```

```
  Properties:
    InternetGatewayId: !Ref InternetGateway
    VpcId: !Ref VPC

# Public Subnets for ALB
PublicSubnet1:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref VPC
    AvailabilityZone: !Select [0, !GetAZs '']
    CidrBlock: !Ref PublicSubnet1Cidr
    MapPublicIpOnLaunch: true
    Tags:
      - Key: Name
        Value: !Sub '${AWS::StackName}-public-subnet-1'

PublicSubnet2:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref VPC
    AvailabilityZone: !Select [1, !GetAZs '']
    CidrBlock: !Ref PublicSubnet2Cidr
    MapPublicIpOnLaunch: true
    Tags:
      - Key: Name
        Value: !Sub '${AWS::StackName}-public-subnet-2'

# Private Subnets for ECS Tasks
PrivateSubnet1:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref VPC
    AvailabilityZone: !Select [0, !GetAZs '']
    CidrBlock: !Ref PrivateSubnet1Cidr
    Tags:
      - Key: Name
        Value: !Sub '${AWS::StackName}-private-subnet-1'

PrivateSubnet2:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref VPC
    AvailabilityZone: !Select [1, !GetAZs '']
    CidrBlock: !Ref PrivateSubnet2Cidr
```

```
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-private-subnet-2'

  # NAT Gateways for private subnet internet access
  NatGateway1EIP:
    Type: AWS::EC2::EIP
    DependsOn: InternetGatewayAttachment
    Properties:
      Domain: vpc
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-nat-eip-1'

  NatGateway2EIP:
    Type: AWS::EC2::EIP
    DependsOn: InternetGatewayAttachment
    Properties:
      Domain: vpc
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-nat-eip-2'

  NatGateway1:
    Type: AWS::EC2::NatGateway
    Properties:
      AllocationId: !GetAtt NatGateway1EIP.AllocationId
      SubnetId: !Ref PublicSubnet1
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-nat-1'

  NatGateway2:
    Type: AWS::EC2::NatGateway
    Properties:
      AllocationId: !GetAtt NatGateway2EIP.AllocationId
      SubnetId: !Ref PublicSubnet2
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-nat-2'

  # Route Tables
  PublicRouteTable:
    Type: AWS::EC2::RouteTable
```

```
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-public-routes'

  DefaultPublicRoute:
    Type: AWS::EC2::Route
    DependsOn: InternetGatewayAttachment
    Properties:
      RouteTableId: !Ref PublicRouteTable
      DestinationCidrBlock: 0.0.0.0/0
      GatewayId: !Ref InternetGateway

  PublicSubnet1RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet1

  PublicSubnet2RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet2

  PrivateRouteTable1:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-private-routes-1'

  DefaultPrivateRoute1:
    Type: AWS::EC2::Route
    Properties:
      RouteTableId: !Ref PrivateRouteTable1
      DestinationCidrBlock: 0.0.0.0/0
      NatGatewayId: !Ref NatGateway1

  PrivateSubnet1RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
```

```
      RouteTableId: !Ref PrivateRouteTable1
      SubnetId: !Ref PrivateSubnet1

  PrivateRouteTable2:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-private-routes-2'

  DefaultPrivateRoute2:
    Type: AWS::EC2::Route
    Properties:
      RouteTableId: !Ref PrivateRouteTable2
      DestinationCidrBlock: 0.0.0.0/0
      NatGatewayId: !Ref NatGateway2

  PrivateSubnet2RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PrivateRouteTable2
      SubnetId: !Ref PrivateSubnet2

  # Security Groups
  ALBSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupName: !Sub '${AWS::StackName}-alb-sg'
      GroupDescription: Security group for Application Load Balancer
      VpcId: !Ref VPC
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
          CidrIp: 0.0.0.0/0
          Description: Allow HTTP traffic from internet
      SecurityGroupEgress:
        - IpProtocol: -1
          CidrIp: 0.0.0.0/0
          Description: Allow all outbound traffic
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-alb-sg'
```

```
  ECSSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupName: !Sub '${AWS::StackName}-ecs-sg'
      GroupDescription: Security group for ECS tasks
      VpcId: !Ref VPC
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: !Ref ContainerPort
          ToPort: !Ref ContainerPort
          SourceSecurityGroupId: !Ref ALBSecurityGroup
          Description: Allow traffic from ALB
      SecurityGroupEgress:
        - IpProtocol: -1
          CidrIp: 0.0.0.0/0
          Description: Allow all outbound traffic
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-ecs-sg'

  # Application Load Balancer
  ApplicationLoadBalancer:
    Type: AWS::ElasticLoadBalancingV2::LoadBalancer
    Properties:
      Name: !Sub '${AWS::StackName}-alb'
      Scheme: internet-facing
      Type: application
      Subnets:
        - !Ref PublicSubnet1
        - !Ref PublicSubnet2
      SecurityGroups:
        - !Ref ALBSecurityGroup
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-alb'

  ALBTargetGroup:
    Type: AWS::ElasticLoadBalancingV2::TargetGroup
    Properties:
      Name: !Sub '${AWS::StackName}-tg'
      Port: !Ref ContainerPort
      Protocol: HTTP
      VpcId: !Ref VPC
```

```
      TargetType: ip
      HealthCheckIntervalSeconds: 30
      HealthCheckPath: /
      HealthCheckProtocol: HTTP
      HealthCheckTimeoutSeconds: 5
      HealthyThresholdCount: 2
      UnhealthyThresholdCount: 5
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-tg'

  ALBListener:
    Type: AWS::ElasticLoadBalancingV2::Listener
    Properties:
      DefaultActions:
        - Type: forward
          TargetGroupArn: !Ref ALBTargetGroup
      LoadBalancerArn: !Ref ApplicationLoadBalancer
      Port: 80
      Protocol: HTTP

  # ECS Cluster
  ECSCluster:
    Type: AWS::ECS::Cluster
    Properties:
      ClusterName: !Sub '${AWS::StackName}-cluster'
      CapacityProviders:
        - FARGATE
        - FARGATE_SPOT
      DefaultCapacityProviderStrategy:
        - CapacityProvider: FARGATE
          Weight: 1
        - CapacityProvider: FARGATE_SPOT
          Weight: 4
      ClusterSettings:
        - Name: containerInsights
          Value: enabled
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-cluster'

  # IAM Roles
  ECSTaskExecutionRole:
    Type: AWS::IAM::Role
```

```
    Properties:
      RoleName: !Sub '${AWS::StackName}-task-execution-role'
      AssumeRolePolicyDocument:
        Version: '2012-10-17       '
        Statement:
          - Effect: Allow
            Principal:
              Service: ecs-tasks.amazonaws.com
            Action: sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-task-execution-role'

  ECSTaskRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: !Sub '${AWS::StackName}-task-role'
      AssumeRolePolicyDocument:
        Version: '2012-10-17       '
        Statement:
          - Effect: Allow
            Principal:
              Service: ecs-tasks.amazonaws.com
            Action: sts:AssumeRole
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-task-role'

  # CloudWatch Log Group
  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Sub '/ecs/${AWS::StackName}'
      RetentionInDays: 7

  # ECS Task Definition
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      Family: !Sub '${AWS::StackName}-task'
      Cpu: '256'
      Memory: '512'
```

```
      NetworkMode: awsvpc
      RequiresCompatibilities:
        - FARGATE
      ExecutionRoleArn: !GetAtt ECSTaskExecutionRole.Arn
      TaskRoleArn: !GetAtt ECSTaskRole.Arn
      ContainerDefinitions:
        - Name: !Ref ServiceName
          Image: !Ref ContainerImage
          PortMappings:
            - ContainerPort: !Ref ContainerPort
              Protocol: tcp
          Essential: true
          LogConfiguration:
            LogDriver: awslogs
            Options:
              awslogs-group: !Ref LogGroup
              awslogs-region: !Ref AWS::Region
              awslogs-stream-prefix: ecs
          HealthCheck:
            Command:
              - CMD-SHELL
              - curl -f http://localhost/ || exit 1
            Interval: 30
            Timeout: 5
            Retries: 3
            StartPeriod: 60
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-task'

  # ECS Service
  ECSService:
    Type: AWS::ECS::Service
    DependsOn: ALBListener
    Properties:
      ServiceName: !Sub '${AWS::StackName}-service'
      Cluster: !Ref ECSCluster
      TaskDefinition: !Ref TaskDefinition
      DesiredCount: !Ref DesiredCount
      LaunchType: FARGATE
      PlatformVersion: LATEST
      NetworkConfiguration:
        AwsvpcConfiguration:
          AssignPublicIp: DISABLED
```

```
          SecurityGroups:
            - !Ref ECSSecurityGroup
          Subnets:
            - !Ref PrivateSubnet1
            - !Ref PrivateSubnet2
      LoadBalancers:
        - ContainerName: !Ref ServiceName
          ContainerPort: !Ref ContainerPort
          TargetGroupArn: !Ref ALBTargetGroup
      DeploymentConfiguration:
        MaximumPercent: 200
        MinimumHealthyPercent: 50
        DeploymentCircuitBreaker:
          Enable: true
          Rollback: true
      EnableExecuteCommand: true  # For debugging
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-service'

  # Auto Scaling Target
  ServiceScalingTarget:
    Type: AWS::ApplicationAutoScaling::ScalableTarget
    Properties:
      MaxCapacity: !Ref MaxCapacity
      MinCapacity: !Ref MinCapacity
      ResourceId: !Sub 'service/${ECSCluster}/${ECSService.Name}'
      RoleARN: !Sub 'arn:aws:iam::${AWS::AccountId}:role/
aws-service-role/ecs.application-autoscaling.amazonaws.com/
AWSServiceRoleForApplicationAutoScaling_ECSService'
      ScalableDimension: ecs:service:DesiredCount
      ServiceNamespace: ecs

  # Auto Scaling Policy - CPU Utilization
  ServiceScalingPolicy:
    Type: AWS::ApplicationAutoScaling::ScalingPolicy
    Properties:
      PolicyName: !Sub '${AWS::StackName}-cpu-scaling-policy'
      PolicyType: TargetTrackingScaling
      ScalingTargetId: !Ref ServiceScalingTarget
      TargetTrackingScalingPolicyConfiguration:
        PredefinedMetricSpecification:
          PredefinedMetricType: ECSServiceAverageCPUUtilization
        TargetValue: 70.0
```

```
        ScaleOutCooldown: 300
        ScaleInCooldown: 300

Outputs:
  VPCId:
    Description: VPC ID
    Value: !Ref VPC
    Export:
      Name: !Sub '${AWS::StackName}-VPC-ID'

  LoadBalancerURL:
    Description: URL of the Application Load Balancer
    Value: !Sub 'http://${ApplicationLoadBalancer.DNSName}'
    Export:
      Name: !Sub '${AWS::StackName}-ALB-URL'

  ECSClusterName:
    Description: Name of the ECS Cluster
    Value: !Ref ECSCluster
    Export:
      Name: !Sub '${AWS::StackName}-ECS-Cluster'

  ECSServiceName:
    Description: Name of the ECS Service
    Value: !GetAtt ECSService.Name
    Export:
      Name: !Sub '${AWS::StackName}-ECS-Service'

  PrivateSubnet1:
    Description: Private Subnet 1 ID
    Value: !Ref PrivateSubnet1
    Export:
      Name: !Sub '${AWS::StackName}-Private-Subnet-1'

  PrivateSubnet2:
    Description: Private Subnet 2 ID
    Value: !Ref PrivateSubnet2
    Export:
      Name: !Sub '${AWS::StackName}-Private-Subnet-2'
```

ECS Exec を有効にしてサービスをデプロイする

次のテンプレートを使用して、ECS Exec を有効にしてサービスをデプロイできます。このサービスは、ECS Exec セッションを暗号化する KMS キーと、実行コマンドセッションログを Amazon S3 バケットにリダイレクトするログ設定を持つクラスターで実行されます。詳細については、「ECS Exec を使用して Amazon ECS コンテナをモニタリングする」を参照してください。

JSON

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "ECS Cluster with Fargate Service and Task Definition and ECS
 Exec enabled.",
    "Parameters": {
        "ClusterName": {
            "Type": "String",
            "Default": "CFNCluster",
            "Description": "Name of the ECS Cluster"
        },
        "S3BucketName": {
            "Type": "String",
            "Default": "amzn-s3-demo-bucket",
            "Description": "S3 bucket for ECS execute command logs"
        },
        "KmsKeyId": {
            "Type": "String",
            "Default": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
            "Description": "KMS Key ID for ECS execute command encryption"
        },
        "ContainerImage": {
            "Type": "String",
            "Default": "public.ecr.aws/docker/library/httpd:2.4",
            "Description": "Container image to use for the task"
        },
        "ContainerCpu": {
            "Type": "Number",
            "Default": 256,
            "AllowedValues": [256, 512, 1024, 2048, 4096],
            "Description": "CPU units for the container (256 = 0.25 vCPU)"
        },
        "ContainerMemory": {
            "Type": "Number",
            "Default": 512,
```

```
            "AllowedValues": [512, 1024, 2048, 3072, 4096, 5120, 6144, 7168, 8192],
            "Description": "Memory for the container (in MiB)"
        },
        "DesiredCount": {
            "Type": "Number",
            "Default": 1,
            "Description": "Desired count of tasks in the service"
        },
       "SecurityGroups": {
 "Type": "List<AWS::EC2::SecurityGroup::Id>",
 "Description": "Security Group IDs for the ECS Service"
},
"Subnets": {
 "Type": "List<AWS::EC2::Subnet::Id>",
 "Description": "Subnet IDs for the ECS Service"
},
        "ServiceName": {
            "Type": "String",
            "Default": "cfn-service",
            "Description": "Name of the ECS service"
        },
        "TaskFamily": {
            "Type": "String",
            "Default": "task-definition-cfn",
            "Description": "Family name for the task definition"
        },
        "TaskExecutionRoleArn": {
            "Type": "String",
            "Description": "ARN of an existing IAM role for ECS task execution",
            "Default": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole"
        },
        "TaskRoleArn": {
            "Type": "String",
            "Description": "ARN of an existing IAM role for ECS tasks",
            "Default": "arn:aws:iam::111122223333:role/execTaskRole"
        }
    },
    "Resources": {
        "ECSCluster": {
            "Type": "AWS::ECS::Cluster",
            "Properties": {
                "ClusterName": {"Ref": "ClusterName"},
                "Configuration": {
                    "ExecuteCommandConfiguration": {
```

```
                        "Logging": "OVERRIDE",
                        "LogConfiguration": {
                            "S3BucketName": {"Ref": "S3BucketName"}
                        },
                        "KmsKeyId": {"Ref": "KmsKeyId"}
                    }
                },
                "Tags": [
                    {
                        "Key": "Environment",
                        "Value": {"Ref": "AWS::StackName"}
                    }
                ]
            }
        },
        "ECSTaskDefinition": {
            "Type": "AWS::ECS::TaskDefinition",
            "Properties": {
                "ContainerDefinitions": [
                    {
                        "Command": [
                            "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS
 Sample App</title> <style>body {margin-top: 40px; background-color: #333;} </style>
 </head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</
h1> <h2>Congratulations!</h2> <p>Your application is now running on a container in
 Amazon ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html &&
 httpd-foreground\""
                        ],
                        "EntryPoint": [
                            "sh",
                            "-c"
                        ],
                        "Essential": true,
                        "Image": {"Ref": "ContainerImage"},
                        "LogConfiguration": {
                            "LogDriver": "awslogs",
                            "Options": {
                                "mode": "non-blocking",
                                "max-buffer-size": "25m",
                                "awslogs-create-group": "true",
                                "awslogs-group": {"Fn::Sub": "/ecs/
${AWS::StackName}"},
                                "awslogs-region": {"Ref": "AWS::Region"},
                                "awslogs-stream-prefix": "ecs"
```

```
                        }
                    },
                    "Name": "sample-fargate-app",
                    "PortMappings": [
                        {
                            "ContainerPort": 80,
                            "HostPort": 80,
                            "Protocol": "tcp"
                        }
                    ]
                }
            ],
            "Cpu": {"Ref": "ContainerCpu"},
            "ExecutionRoleArn": {"Ref": "TaskExecutionRoleArn"},
            "TaskRoleArn": {"Ref": "TaskRoleArn"},
            "Family": {"Ref": "TaskFamily"},
            "Memory": {"Ref": "ContainerMemory"},
            "NetworkMode": "awsvpc",
            "RequiresCompatibilities": [
                "FARGATE"
            ],
            "RuntimePlatform": {
                "OperatingSystemFamily": "LINUX"
            },
            "Tags": [
                {
                    "Key": "Name",
                    "Value": {"Fn::Sub": "${AWS::StackName}-TaskDefinition"}
                }
            ]
        }
    },
    "ECSService": {
        "Type": "AWS::ECS::Service",
        "Properties": {
            "ServiceName": {"Ref": "ServiceName"},
            "Cluster": {"Ref": "ECSCluster"},
            "DesiredCount": {"Ref": "DesiredCount"},
            "EnableExecuteCommand": true,
            "LaunchType": "FARGATE",
            "NetworkConfiguration": {
                "AwsvpcConfiguration": {
                    "AssignPublicIp": "ENABLED",
                    "SecurityGroups": {"Ref": "SecurityGroups"},
```

```
                        "Subnets": {"Ref": "Subnets"}
                    }
                },
                "TaskDefinition": {"Ref": "ECSTaskDefinition"},
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {"Fn::Sub": "${AWS::StackName}-Service"}
                    }
                ]
            }
        }
    },
    "Outputs": {
        "ClusterName": {
            "Description": "The name of the ECS cluster",
            "Value": {"Ref": "ECSCluster"}
        },
        "ServiceName": {
            "Description": "The name of the ECS service",
            "Value": {"Ref": "ServiceName"}
        },
        "TaskDefinitionArn": {
            "Description": "The ARN of the task definition",
            "Value": {"Ref": "ECSTaskDefinition"}
        },
        "ClusterArn": {
            "Description": "The ARN of the ECS cluster",
            "Value": {"Fn::GetAtt": ["ECSCluster", "Arn"]}
        },
        "StackName": {
            "Description": "The name of this stack",
            "Value": {"Ref": "AWS::StackName"}
        },
        "StackId": {
            "Description": "The unique identifier for this stack",
            "Value": {"Ref": "AWS::StackId"}
        },
        "Region": {
            "Description": "The AWS Region where the stack is deployed",
            "Value": {"Ref": "AWS::Region"}
        },
        "AccountId": {
            "Description": "The AWS Account ID",
```

```
            "Value": {"Ref": "AWS::AccountId"}
        }
    }
}
```

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Description: ECS Cluster with Fargate Service and Task Definition and ECS Exec
 enabled.
Parameters:
  ClusterName:
    Type: String
    Default: CFNCluster
    Description: Name of the ECS Cluster
  S3BucketName:
    Type: String
    Default: amzn-s3-demo-bucket
    Description: S3 bucket for ECS execute command logs
  KmsKeyId:
    Type: String
    Default: a1b2c3d4-5678-90ab-cdef-EXAMPLE11111
    Description: KMS Key ID for ECS execute command encryption
  ContainerImage:
    Type: String
    Default: public.ecr.aws/docker/library/httpd:2.4
    Description: Container image to use for the task
  ContainerCpu:
    Type: Number
    Default: 256
    AllowedValues: [256, 512, 1024, 2048, 4096]
    Description: CPU units for the container (256 = 0.25 vCPU)
  ContainerMemory:
    Type: Number
    Default: 512
    AllowedValues: [512, 1024, 2048, 3072, 4096, 5120, 6144, 7168, 8192]
    Description: Memory for the container (in MiB)
  DesiredCount:
    Type: Number
    Default: 1
    Description: Desired count of tasks in the service
 SecurityGroups:
    Type: List<AWS::EC2::SecurityGroup::Id>
```

```
    Description: Security Group IDs for the ECS Service
  Subnets:
    Type: List<AWS::EC2::Subnet::Id>
    Description: Subnet IDs for the ECS Service
  ServiceName:
    Type: String
    Default: cfn-service
    Description: Name of the ECS service
  TaskFamily:
    Type: String
    Default: task-definition-cfn
    Description: Family name for the task definition
  TaskExecutionRoleArn:
    Type: String
    Description: ARN of an existing IAM role for ECS task execution
    Default: 'arn:aws:iam::111122223333:role/ecsTaskExecutionRole'
  TaskRoleArn:
    Type: String
    Description: ARN of an existing IAM role for ECS tasks
    Default: 'arn:aws:iam::111122223333:role/execTaskRole'
Resources:
  ECSCluster:
    Type: AWS::ECS::Cluster
    Properties:
      ClusterName: !Ref ClusterName
      Configuration:
        ExecuteCommandConfiguration:
          Logging: OVERRIDE
          LogConfiguration:
            S3BucketName: !Ref S3BucketName
          KmsKeyId: !Ref KmsKeyId
      Tags:
        - Key: Environment
          Value: !Ref AWS::StackName
  ECSTaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      ContainerDefinitions:
        - Command:
            - >-
             /bin/sh -c "echo '<html> <head> <title>Amazon ECS Sample
              App</title> <style>body {margin-top: 40px; background-color:
              #333;} </style> </head><body> <div
              style=color:white;text-align:center> <h1>Amazon ECS Sample
```

```
            App</h1> <h2>Congratulations!</h2> <p>Your application is now
            running on a container in Amazon ECS.</p> </div></body></html>' >
            /usr/local/apache2/htdocs/index.html && httpd-foreground"
        EntryPoint:
          - sh
          - '-c'
        Essential: true
        Image: !Ref ContainerImage
        LogConfiguration:
          LogDriver: awslogs
          Options:
            mode: non-blocking
            max-buffer-size: 25m
            awslogs-create-group: 'true'
            awslogs-group: !Sub /ecs/${AWS::StackName}
            awslogs-region: !Ref AWS::Region
            awslogs-stream-prefix: ecs
        Name: sample-fargate-app
        PortMappings:
          - ContainerPort: 80
            HostPort: 80
            Protocol: tcp
    Cpu: !Ref ContainerCpu
    ExecutionRoleArn: !Ref TaskExecutionRoleArn
    TaskRoleArn: !Ref TaskRoleArn
    Family: !Ref TaskFamily
    Memory: !Ref ContainerMemory
    NetworkMode: awsvpc
    RequiresCompatibilities:
      - FARGATE
    RuntimePlatform:
      OperatingSystemFamily: LINUX
    Tags:
      - Key: Name
        Value: !Sub ${AWS::StackName}-TaskDefinition
  ECSService:
    Type: AWS::ECS::Service
    Properties:
      ServiceName: !Ref ServiceName
      Cluster: !Ref ECSCluster
      DesiredCount: !Ref DesiredCount
      EnableExecuteCommand: true
      LaunchType: FARGATE
      NetworkConfiguration:
```

```
        AwsvpcConfiguration:
          AssignPublicIp: ENABLED
          SecurityGroups: !Ref SecurityGroups
          Subnets: !Ref Subnets
      TaskDefinition: !Ref ECSTaskDefinition
      Tags:
        - Key: Name
          Value: !Sub ${AWS::StackName}-Service
Outputs:
  ClusterName:
    Description: The name of the ECS cluster
    Value: !Ref ECSCluster
  ServiceName:
    Description: The name of the ECS service
    Value: !Ref ServiceName
  TaskDefinitionArn:
    Description: The ARN of the task definition
    Value: !Ref ECSTaskDefinition
  ClusterArn:
    Description: The ARN of the ECS cluster
    Value: !GetAtt ECSCluster.Arn
  StackName:
    Description: The name of this stack
    Value: !Ref AWS::StackName
  StackId:
    Description: The unique identifier for this stack
    Value: !Ref AWS::StackId
  Region:
    Description: The AWS Region where the stack is deployed
    Value: !Ref AWS::Region
  AccountId:
    Description: The AWS Account ID
    Value: !Ref AWS::AccountId
```

Amazon VPC Lattice を使用するサービスをデプロイする

次のテンプレートを使用して、VPC Lattice で Amazon ECS サービスの作成を開始できます。VPC Lattice をセットアップするには、以下の追加ステップを完了する必要がある場合があります。

- VPC Lattice のセキュリティグループのインバウンドルールを更新して、インバウンドルールの `vpc-lattice` プレフィックスを許可し、ポート 80 でのトラフィックを許可します。
- サービスの VPC を VPC Lattice サービスネットワークに関連付けます。

- Amazon Route 53 でプライベートホストゾーンまたはパブリックホストゾーンを設定します。
- VPC Lattice サービスでリスナーとリスナールールを設定します。
- ターゲットグループのヘルスチェック設定を確認します。

Amazon ECS での VPC Lattice の使用に関する詳細は、「Amazon VPC Lattice を使用して Amazon ECS サービスを接続、監視、保護する」を参照してください。

JSON

```
{
 "AWSTemplateFormatVersion": "2010-09-09",
 "Description": "The template used to create an ECS Service with VPC Lattice.",
 "Parameters": {
  "ECSClusterName": {
   "Type": "String",
   "Default": "vpc-lattice-cluster"
  },
  "ECSServiceName": {
   "Type": "String",
   "Default": "vpc-lattice-service"
  },
  "SecurityGroupIDs": {
   "Type": "List<AWS::EC2::SecurityGroup::Id>",
   "Description": "Security Group IDs for the ECS Service"
  },
  "SubnetIDs": {
   "Type": "List<AWS::EC2::Subnet::Id>",
   "Description": "Subnet IDs for the ECS Service"
  },
  "VpcID": {
   "Type": "AWS::EC2::VPC::Id",
   "Description": "VPC ID for the resources"
  },
  "ContainerImage": {
   "Type": "String",
   "Default": "public.ecr.aws/docker/library/httpd:2.4",
   "Description": "Container image to use for the task"
  },
  "TaskCpu": {
   "Type": "Number",
   "Default": 256,
   "AllowedValues": [256, 512, 1024, 2048, 4096],
```

```
    "Description": "CPU units for the task"
   },
   "TaskMemory": {
    "Type": "Number",
    "Default": 512,
    "AllowedValues": [512, 1024, 2048, 4096, 8192, 16384],
    "Description": "Memory (in MiB) for the task"
   },
   "LogGroupName": {
    "Type": "String",
    "Default": "/ecs/vpc-lattice-task",
    "Description": "CloudWatch Log Group name"
   },
   "EnableContainerInsights": {
    "Type": "String",
    "Default": "enabled",
    "AllowedValues": ["enabled", "disabled"],
    "Description": "Enable or disable CloudWatch Container Insights for the cluster"
   }
  },
  "Resources": {
   "ECSCluster": {
    "Type": "AWS::ECS::Cluster",
    "Properties": {
     "ClusterName": {"Ref": "ECSClusterName"},
     "ClusterSettings": [
      {
       "Name": "containerInsights",
       "Value": {"Ref": "EnableContainerInsights"}
      }
     ],
     "Tags": [
      {
       "Key": "Name",
       "Value": {"Ref": "ECSClusterName"}
      }
     ]
    }
   },
   "ECSTaskExecutionRole": {
    "Type": "AWS::IAM::Role",
    "Properties": {
     "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
```

```
    "Statement": [
     {
      "Effect": "Allow",
      "Principal": {
       "Service": "ecs-tasks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
     }
    ]
   },
   "ManagedPolicyArns": [
    "arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy"
   ]
  }
 },
 "TaskLogGroup": {
  "Type": "AWS::Logs::LogGroup",
  "DeletionPolicy": "Retain",
  "UpdateReplacePolicy": "Retain",
  "Properties": {
   "LogGroupName": {"Ref": "LogGroupName"},
   "RetentionInDays": 30
  }
 },
 "VpcLatticeTaskDefinition": {
  "Type": "AWS::ECS::TaskDefinition",
  "Properties": {
   "ContainerDefinitions": [
    {
     "Command": [
      "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample App</title>
<style>body {margin-top: 40px; background-color: #333;} </style> </head><body>
<div style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1>
<h2>Congratulations!</h2> <p>Your application is now running on a container in
Amazon ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html &&
httpd-foreground\""
     ],
     "EntryPoint": [
      "sh",
      "-c"
     ],
     "Essential": true,
     "Image": {"Ref": "ContainerImage"},
     "LogConfiguration": {
```

```
        "LogDriver": "awslogs",
        "Options": {
         "mode": "non-blocking",
         "max-buffer-size": "25m",
         "awslogs-create-group": "true",
         "awslogs-group": {"Ref": "LogGroupName"},
         "awslogs-region": {"Ref": "AWS::Region"},
         "awslogs-stream-prefix": "ecs"
        }
       },
       "Name": "vpc-lattice-container",
       "PortMappings": [
        {
         "ContainerPort": 80,
         "HostPort": 80,
         "Protocol": "tcp",
         "Name": "vpc-lattice-port"
        }
       ]
      }
     ],
     "Cpu": {"Ref": "TaskCpu"},
     "ExecutionRoleArn": {"Fn::GetAtt": ["ECSTaskExecutionRole", "Arn"]},
     "Family": "vpc-lattice-task-definition",
     "Memory": {"Ref": "TaskMemory"},
     "NetworkMode": "awsvpc",
     "RequiresCompatibilities": [
      "FARGATE"
     ],
     "RuntimePlatform": {
      "OperatingSystemFamily": "LINUX"
     }
    }
   },
   "ECSService": {
    "Type": "AWS::ECS::Service",
    "Properties": {
     "Cluster": {"Ref": "ECSCluster"},
     "TaskDefinition": {"Ref": "VpcLatticeTaskDefinition"},
     "LaunchType": "FARGATE",
     "ServiceName": {"Ref": "ECSServiceName"},
     "SchedulingStrategy": "REPLICA",
     "DesiredCount": 2,
     "AvailabilityZoneRebalancing": "ENABLED",
```

```
    "NetworkConfiguration": {
     "AwsvpcConfiguration": {
      "AssignPublicIp": "ENABLED",
      "SecurityGroups": {
       "Ref": "SecurityGroupIDs"
      },
      "Subnets": {
       "Ref": "SubnetIDs"
      }
     }
    },
    "PlatformVersion": "LATEST",
    "VpcLatticeConfigurations": [
     {
      "RoleArn": "arn:aws:iam::111122223333:role/ecsInfrastructureRole",
      "PortName": "vpc-lattice-port",
      "TargetGroupArn": {
       "Ref": "TargetGroup1"
      }
     }
    ],
    "DeploymentConfiguration": {
     "DeploymentCircuitBreaker": {
      "Enable": true,
      "Rollback": true
     },
     "MaximumPercent": 200,
     "MinimumHealthyPercent": 100
    },
    "DeploymentController": {
     "Type": "ECS"
    },
    "ServiceConnectConfiguration": {
     "Enabled": false
    },
    "Tags": [],
    "EnableECSManagedTags": true
   }
  },
  "TargetGroup1": {
   "Type": "AWS::VpcLattice::TargetGroup",
   "Properties": {
    "Type": "IP",
    "Name": "first-target-group",
```

```
    "Config": {
     "Port": 80,
     "Protocol": "HTTP",
     "VpcIdentifier": {"Ref": "VpcID"},
     "HealthCheck": {
      "Enabled": true,
      "Path": "/"
     }
    },
    "Tags": [
     {
      "Key": "ecs-application-networking/ServiceName",
      "Value": {"Ref": "ECSServiceName"}
     },
     {
      "Key": "ecs-application-networking/ClusterName",
      "Value": {"Ref": "ECSClusterName"}
     },
     {
      "Key": "ecs-application-networking/TaskDefinition",
      "Value": {"Ref": "VpcLatticeTaskDefinition"}
     },
     {
      "Key": "ecs-application-networking/VpcId",
      "Value": {"Ref": "VpcID"}
     }
    ]
   }
  }
 },
"Outputs": {
 "ClusterName": {
  "Description": "The cluster used to create the service.",
  "Value": {
   "Ref": "ECSCluster"
  }
 },
 "ClusterArn": {
  "Description": "The ARN of the ECS cluster",
  "Value": {
   "Fn::GetAtt": ["ECSCluster", "Arn"]
  }
 },
 "ECSService": {
```

```
   "Description": "The created service.",
   "Value": {
    "Ref": "ECSService"
   }
  },
  "TaskDefinitionArn": {
   "Description": "The ARN of the task definition",
   "Value": {
    "Ref": "VpcLatticeTaskDefinition"
   }
  }
 }
}
```

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Description: The template used to create an ECS Service with VPC Lattice.

Parameters:
  ECSClusterName:
    Type: String
    Default: vpc-lattice-cluster
  ECSServiceName:
    Type: String
    Default: vpc-lattice-service
  SecurityGroupIDs:
    Type: List<AWS::EC2::SecurityGroup::Id>
    Description: Security Group IDs for the ECS Service
  SubnetIDs:
    Type: List<AWS::EC2::Subnet::Id>
    Description: Subnet IDs for the ECS Service
  VpcID:
    Type: AWS::EC2::VPC::Id
    Description: VPC ID for the resources
  ContainerImage:
    Type: String
    Default: public.ecr.aws/docker/library/httpd:2.4
    Description: Container image to use for the task
  TaskCpu:
    Type: Number
    Default: 256
    AllowedValues: [256, 512, 1024, 2048, 4096]
```

```
    Description: CPU units for the task
  TaskMemory:
    Type: Number
    Default: 512
    AllowedValues: [512, 1024, 2048, 4096, 8192, 16384]
    Description: Memory (in MiB) for the task
  LogGroupName:
    Type: String
    Default: /ecs/vpc-lattice-task
    Description: CloudWatch Log Group name
  EnableContainerInsights:
    Type: String
    Default: 'enhanced'
    AllowedValues: ['enabled', 'disabled', 'enhanced']
    Description: Enable or disable CloudWatch Container Insights for the cluster

Resources:
  # ECS Cluster
  ECSCluster:
    Type: AWS::ECS::Cluster
    Properties:
      ClusterName: !Ref ECSClusterName
      ClusterSettings:
        - Name: containerInsights
          Value: !Ref EnableContainerInsights
      Tags:
        - Key: Name
          Value: !Ref ECSClusterName

  # IAM Roles
  ECSTaskExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17       '
        Statement:
          - Effect: Allow
            Principal:
              Service: ecs-tasks.amazonaws.com
            Action: sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy

  # CloudWatch Logs
```

```
  TaskLogGroup:
    Type: AWS::Logs::LogGroup
    DeletionPolicy: Retain
    UpdateReplacePolicy: Retain
    Properties:
      LogGroupName: !Ref LogGroupName
      RetentionInDays: 30

  # Task Definition
  VpcLatticeTaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      ContainerDefinitions:
        - Command:
            - >-
              /bin/sh -c "echo '<html> <head> <title>Amazon ECS Sample
              App</title> <style>body {margin-top: 40px; background-color:
              #333;} </style> </head><body> <div
              style=color:white;text-align:center> <h1>Amazon ECS Sample
              App</h1> <h2>Congratulations!</h2> <p>Your application is now
              running on a container in Amazon ECS.</p> </div></body></html>' >
              /usr/local/apache2/htdocs/index.html && httpd-foreground"
          EntryPoint:
            - sh
            - '-c'
          Essential: true
          Image: !Ref ContainerImage
          LogConfiguration:
            LogDriver: awslogs
            Options:
              mode: non-blocking
              max-buffer-size: 25m
              awslogs-create-group: 'true'
              awslogs-group: !Ref LogGroupName
              awslogs-region: !Ref 'AWS::Region'
              awslogs-stream-prefix: ecs
          Name: vpc-lattice-container
          PortMappings:
            - ContainerPort: 80
              HostPort: 80
              Protocol: tcp
              Name: vpc-lattice-port
      Cpu: !Ref TaskCpu
      ExecutionRoleArn: !GetAtt ECSTaskExecutionRole.Arn
```

```
      Family: vpc-lattice-task-definition
      Memory: !Ref TaskMemory
      NetworkMode: awsvpc
      RequiresCompatibilities:
        - FARGATE
      RuntimePlatform:
        OperatingSystemFamily: LINUX

  ECSService:
    Type: AWS::ECS::Service
    Properties:
      Cluster: !Ref ECSCluster
      TaskDefinition: !Ref VpcLatticeTaskDefinition
      LaunchType: FARGATE
      ServiceName: !Ref ECSServiceName
      SchedulingStrategy: REPLICA
      DesiredCount: 2
      AvailabilityZoneRebalancing: ENABLED
      NetworkConfiguration:
        AwsvpcConfiguration:
          AssignPublicIp: ENABLED
          SecurityGroups: !Ref SecurityGroupIDs
          Subnets: !Ref SubnetIDs
      PlatformVersion: LATEST
      VpcLatticeConfigurations:
        - RoleArn: arn:aws:iam::111122223333:role/ecsInfrastructureRole
          PortName: vpc-lattice-port
          TargetGroupArn: !Ref TargetGroup1
      DeploymentConfiguration:
        DeploymentCircuitBreaker:
          Enable: true
          Rollback: true
        MaximumPercent: 200
        MinimumHealthyPercent: 100
      DeploymentController:
        Type: ECS
      ServiceConnectConfiguration:
        Enabled: false
      Tags: []
      EnableECSManagedTags: true

  TargetGroup1:
    Type: AWS::VpcLattice::TargetGroup
    Properties:
```

```
      Type: IP
      Name: first-target-group
      Config:
        Port: 80
        Protocol: HTTP
        VpcIdentifier: !Ref VpcID
        HealthCheck:
          Enabled: true
          Path: /
      Tags:
        - Key: ecs-application-networking/ServiceName
          Value: !Ref ECSServiceName
        - Key: ecs-application-networking/ClusterName
          Value: !Ref ECSClusterName
        - Key: ecs-application-networking/TaskDefinition
          Value: !Ref VpcLatticeTaskDefinition
        - Key: ecs-application-networking/VpcId
          Value: !Ref VpcID

Outputs:
  ClusterName:
    Description: The cluster used to create the service.
    Value: !Ref ECSCluster
  ClusterArn:
    Description: The ARN of the ECS cluster
    Value: !GetAtt ECSCluster.Arn
  ECSService:
    Description: The created service.
    Value: !Ref ECSService
  TaskDefinitionArn:
    Description: The ARN of the task definition
    Value: !Ref VpcLatticeTaskDefinition
```

ボリューム設定を使用してサービスをデプロイする

次のテンプレートでは、サービス定義にボリューム設定が含まれています。Amazon ECS では、起動時にボリューム設定を使用して以下のデータボリュームを設定できます: Amazon EBS ボリューム。Amazon ECS での Amazon EBS ボリュームの使用 ボリュームの詳細については、「Amazon EBS」を参照してください。

JSON

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "The template used to create an ECS Service that includes a
 volume configuration. The configuration is used to create Amazon EBS volumes for
 attachment to the tasks. One volume is attached per task.",
  "Parameters": {
    "ECSClusterName": {
      "Type": "String",
      "Default": "volume-config-cluster",
      "Description": "Name of the ECS cluster"
    },
    "SecurityGroupIDs": {
      "Type": "List<AWS::EC2::SecurityGroup::Id>",
      "Description": "Security Group IDs for the ECS Service"
    },
    "SubnetIDs": {
      "Type": "List<AWS::EC2::Subnet::Id>",
      "Description": "Subnet IDs for the ECS Service"
    },
    "InfrastructureRoleArn": {
      "Type": "String",
      "Description": "ARN of the IAM role that ECS will use to manage EBS volumes"
    },
    "ContainerImage": {
      "Type": "String",
      "Default": "public.ecr.aws/nginx/nginx:latest",
      "Description": "Container image to use for the task"
    },
    "TaskCpu": {
      "Type": "String",
      "Default": "2048",
      "Description": "CPU units for the task"
    },
    "TaskMemory": {
      "Type": "String",
      "Default": "4096",
      "Description": "Memory (in MiB) for the task"
    },
    "VolumeSize": {
      "Type": "String",
      "Default": "10",
      "Description": "Size of the EBS volume in GiB"
```

```
    },
    "VolumeType": {
      "Type": "String",
      "Default": "gp3",
      "AllowedValues": ["gp2", "gp3", "io1", "io2", "st1", "sc1", "standard"],
      "Description": "EBS volume type"
    },
    "VolumeIops": {
      "Type": "String",
      "Default": "3000",
      "Description": "IOPS for the EBS volume (required for io1, io2, and gp3)"
    },
    "VolumeThroughput": {
      "Type": "String",
      "Default": "125",
      "Description": "Throughput for the EBS volume (only for gp3)"
    },
    "FilesystemType": {
      "Type": "String",
      "Default": "xfs",
      "AllowedValues": ["xfs", "ext4"],
      "Description": "Filesystem type for the EBS volume"
    },
    "EnableContainerInsights": {
      "Type": "String",
      "Default": "enhanced",
      "AllowedValues": ["enabled", "disabled", "enhanced"],
      "Description": "Enable or disable CloudWatch Container Insights for the
cluster"
    }
  },
  "Resources": {
    "ECSCluster": {
      "Type": "AWS::ECS::Cluster",
      "Properties": {
        "ClusterName": {"Ref": "ECSClusterName"},
        "ClusterSettings": [
          {
            "Name": "containerInsights",
            "Value": {"Ref": "EnableContainerInsights"}
          }
        ],
        "Tags": [
          {
```

```
            "Key": "Name",
            "Value": {"Ref": "ECSClusterName"}
          }
        ]
      }
    },
    "ECSTaskExecutionRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Effect": "Allow",
              "Principal": {
                "Service": "ecs-tasks.amazonaws.com"
              },
              "Action": "sts:AssumeRole"
            }
          ]
        },
        "ManagedPolicyArns": [
          "arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy"
        ]
      }
    },
    "EBSTaskDefinition": {
      "Type": "AWS::ECS::TaskDefinition",
      "Properties": {
        "Family": "ebs-task-attach-task-def",
        "ExecutionRoleArn": {"Fn::GetAtt": ["ECSTaskExecutionRole", "Arn"]},
        "NetworkMode": "awsvpc",
        "RequiresCompatibilities": [
          "EC2",
          "FARGATE"
        ],
        "Cpu": {"Ref": "TaskCpu"},
        "Memory": {"Ref": "TaskMemory"},
        "ContainerDefinitions": [
          {
            "Name": "nginx",
            "Image": {"Ref": "ContainerImage"},
            "Essential": true,
            "PortMappings": [
```

```
            {
              "Name": "nginx-80-tcp",
              "ContainerPort": 80,
              "HostPort": 80,
              "Protocol": "tcp",
              "AppProtocol": "http"
            }
          ],
          "MountPoints": [
            {
              "SourceVolume": "ebs-vol",
              "ContainerPath": "/foo-container-path",
              "ReadOnly": false
            }
          ]
        }
      ],
      "Volumes": [
        {
          "Name": "ebs-vol",
          "ConfiguredAtLaunch": true
        }
      ]
    }
  },
  "ECSService": {
    "Type": "AWS::ECS::Service",
    "Properties": {
      "Cluster": {"Ref": "ECSCluster"},
      "TaskDefinition": {"Ref": "EBSTaskDefinition"},
      "LaunchType": "FARGATE",
      "ServiceName": "ebs",
      "SchedulingStrategy": "REPLICA",
      "DesiredCount": 1,
      "NetworkConfiguration": {
        "AwsvpcConfiguration": {
          "AssignPublicIp": "ENABLED",
          "SecurityGroups": {"Ref": "SecurityGroupIDs"},
          "Subnets": {"Ref": "SubnetIDs"}
        }
      },
      "PlatformVersion": "LATEST",
      "DeploymentConfiguration": {
        "MaximumPercent": 200,
```

```
          "MinimumHealthyPercent": 100,
          "DeploymentCircuitBreaker": {
            "Enable": true,
            "Rollback": true
          }
        },
        "DeploymentController": {
          "Type": "ECS"
        },
        "Tags": [],
        "EnableECSManagedTags": true,
        "VolumeConfigurations": [
          {
            "Name": "ebs-vol",
            "ManagedEBSVolume": {
              "RoleArn": {"Ref": "InfrastructureRoleArn"},
              "VolumeType": {"Ref": "VolumeType"},
              "Iops": {"Ref": "VolumeIops"},
              "Throughput": {"Ref": "VolumeThroughput"},
              "SizeInGiB": {"Ref": "VolumeSize"},
              "FilesystemType": {"Ref": "FilesystemType"},
              "TagSpecifications": [
                {
                  "ResourceType": "volume",
                  "PropagateTags": "TASK_DEFINITION"
                }
              ]
            }
          }
        ]
      }
    }
  },
  "Outputs": {
    "ClusterName": {
      "Description": "The cluster used to create the service.",
      "Value": {"Ref": "ECSCluster"}
    },
    "ClusterArn": {
      "Description": "The ARN of the ECS cluster",
      "Value": {"Fn::GetAtt": ["ECSCluster", "Arn"]}
    },
    "ECSService": {
      "Description": "The created service.",
```

```
      "Value": {"Ref": "ECSService"}
    },
    "TaskDefinitionArn": {
      "Description": "The ARN of the task definition",
      "Value": {"Ref": "EBSTaskDefinition"}
    }
  }
}
```

YAML

```
AWSTemplateFormatVersion: 2010-09-09
Description: The template used to create an ECS Service that includes a volume
 configuration. The configuration is used to create Amazon EBS volumes for
 attachment to the tasks. One volume is attached per task.
Parameters:
  ECSClusterName:
    Type: String
    Default: volume-config-cluster
    Description: Name of the ECS cluster

  SecurityGroupIDs:
    Type: List<AWS::EC2::SecurityGroup::Id>
    Description: Security Group IDs for the ECS Service

  SubnetIDs:
    Type: List<AWS::EC2::Subnet::Id>
    Description: Subnet IDs for the ECS Service

  InfrastructureRoleArn:
    Type: String
    Description: ARN of the IAM role that ECS will use to manage EBS volumes

  ContainerImage:
    Type: String
    Default: public.ecr.aws/nginx/nginx:latest
    Description: Container image to use for the task

  TaskCpu:
    Type: String
    Default: "2048"
    Description: CPU units for the task
```

```
  TaskMemory:
    Type: String
    Default: "4096"
    Description: Memory (in MiB) for the task

  VolumeSize:
    Type: String
    Default: "10"
    Description: Size of the EBS volume in GiB

  VolumeType:
    Type: String
    Default: gp3
    AllowedValues: [gp2, gp3, io1, io2, st1, sc1, standard]
    Description: EBS volume type

  VolumeIops:
    Type: String
    Default: "3000"
    Description: IOPS for the EBS volume (required for io1, io2, and gp3)

  VolumeThroughput:
    Type: String
    Default: "125"
    Description: Throughput for the EBS volume (only for gp3)

  FilesystemType:
    Type: String
    Default: xfs
    AllowedValues: [xfs, ext4]
    Description: Filesystem type for the EBS volume

  EnableContainerInsights:
    Type: String
    Default: 'enhanced'
    AllowedValues: ['enabled', 'disabled', 'enhanced']
    Description: Enable or disable CloudWatch Container Insights for the cluster

Resources:
  # ECS Cluster
  ECSCluster:
    Type: AWS::ECS::Cluster
    Properties:
      ClusterName: !Ref ECSClusterName
```

```
      ClusterSettings:
        - Name: containerInsights
          Value: !Ref EnableContainerInsights
      Tags:
        - Key: Name
          Value: !Ref ECSClusterName

  # IAM Role for Task Execution
  ECSTaskExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17       '
        Statement:
          - Effect: Allow
            Principal:
              Service: ecs-tasks.amazonaws.com
            Action: sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy

  # Task Definition
  EBSTaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      Family: ebs-task-attach-task-def
      ExecutionRoleArn: !GetAtt ECSTaskExecutionRole.Arn
      NetworkMode: awsvpc
      RequiresCompatibilities:
        - EC2
        - FARGATE
      Cpu: !Ref TaskCpu
      Memory: !Ref TaskMemory
      ContainerDefinitions:
        - Name: nginx
          Image: !Ref ContainerImage
          Essential: true
          PortMappings:
            - Name: nginx-80-tcp
              ContainerPort: 80
              HostPort: 80
              Protocol: tcp
              AppProtocol: http
          MountPoints:
```

```
            - SourceVolume: ebs-vol
              ContainerPath: /foo-container-path
              ReadOnly: false
      Volumes:
        - Name: ebs-vol
          ConfiguredAtLaunch: true

  ECSService:
    Type: AWS::ECS::Service
    Properties:
      Cluster: !Ref ECSCluster
      TaskDefinition: !Ref EBSTaskDefinition
      LaunchType: FARGATE
      ServiceName: ebs
      SchedulingStrategy: REPLICA
      DesiredCount: 1
      NetworkConfiguration:
        AwsvpcConfiguration:
          AssignPublicIp: ENABLED
          SecurityGroups: !Ref SecurityGroupIDs
          Subnets: !Ref SubnetIDs
      PlatformVersion: LATEST
      DeploymentConfiguration:
        MaximumPercent: 200
        MinimumHealthyPercent: 100
        DeploymentCircuitBreaker:
          Enable: true
          Rollback: true
      DeploymentController:
        Type: ECS
      Tags: []
      EnableECSManagedTags: true
      VolumeConfigurations:
        - Name: ebs-vol
          ManagedEBSVolume:
            RoleArn: !Ref InfrastructureRoleArn
            VolumeType: !Ref VolumeType
            Iops: !Ref VolumeIops
            Throughput: !Ref VolumeThroughput
            SizeInGiB: !Ref VolumeSize
            FilesystemType: !Ref FilesystemType
            TagSpecifications:
              - ResourceType: volume
                PropagateTags: TASK_DEFINITION
```

```
Outputs:
  ClusterName:
    Description: The cluster used to create the service.
    Value: !Ref ECSCluster
  ClusterArn:
    Description: The ARN of the ECS cluster
    Value: !GetAtt ECSCluster.Arn
  ECSService:
    Description: The created service.
    Value: !Ref ECSService
  TaskDefinitionArn:
    Description: The ARN of the task definition
    Value: !Ref EBSTaskDefinition
```

キャパシティープロバイダーを使用してサービスをデプロイする

次のテンプレートは、キャパシティープロバイダーを使用して AL2023 キャパシティーの実行をリクエストするサービスを定義しています。コンテナは、オンラインになると AL2023 インスタンスで起動します。

JSON

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "An example service that deploys in AWS VPC networking mode on
 EC2 capacity. Service uses a capacity provider to request EC2 instances to run
 on. Service runs with networking in private subnets, but still accessible to the
 internet via a load balancer hosted in public subnets.",
  "Parameters": {
      "VpcId": {
          "Type": "String",
          "Description": "The VPC that the service is running inside of"
      },
      "PublicSubnetIds": {
          "Type": "List<AWS::EC2::Subnet::Id>",
          "Description": "List of public subnet ID's to put the load balancer in"
      },
      "PrivateSubnetIds": {
          "Type": "List<AWS::EC2::Subnet::Id>",
          "Description": "List of private subnet ID's that the AWS VPC tasks are in"
      },
```

```
    "ClusterName": {
        "Type": "String",
        "Description": "The name of the ECS cluster into which to launch
capacity."
    },
    "ECSTaskExecutionRole": {
        "Type": "String",
        "Description": "The role used to start up an ECS task"
    },
    "CapacityProvider": {
        "Type": "String",
        "Description": "The cluster capacity provider that the service should use
to request capacity when it wants to start up a task"
    },
    "ServiceName": {
        "Type": "String",
        "Default": "web",
        "Description": "A name for the service"
    },
    "ImageUrl": {
        "Type": "String",
        "Default": "public.ecr.aws/docker/library/nginx:latest",
        "Description": "The url of a docker image that contains the application
process that will handle the traffic for this service"
    },
    "ContainerCpu": {
        "Type": "Number",
        "Default": 256,
        "Description": "How much CPU to give the container. 1024 is 1 CPU"
    },
    "ContainerMemory": {
        "Type": "Number",
        "Default": 512,
        "Description": "How much memory in megabytes to give the container"
    },
    "ContainerPort": {
        "Type": "Number",
        "Default": 80,
        "Description": "What port that the application expects traffic on"
    },
    "DesiredCount": {
        "Type": "Number",
        "Default": 2,
        "Description": "How many copies of the service task to run"
```

```
        }
    },
    "Resources": {
        "TaskDefinition": {
            "Type": "AWS::ECS::TaskDefinition",
            "Properties": {
                "Family": {
                    "Ref": "ServiceName"
                },
                "Cpu": {
                    "Ref": "ContainerCpu"
                },
                "Memory": {
                    "Ref": "ContainerMemory"
                },
                "NetworkMode": "awsvpc",
                "RequiresCompatibilities": [
                    "EC2"
                ],
                "ExecutionRoleArn": {
                    "Ref": "ECSTaskExecutionRole"
                },
                "ContainerDefinitions": [
                    {
                        "Name": {
                            "Ref": "ServiceName"
                        },
                        "Cpu": {
                            "Ref": "ContainerCpu"
                        },
                        "Memory": {
                            "Ref": "ContainerMemory"
                        },
                        "Image": {
                            "Ref": "ImageUrl"
                        },
                        "PortMappings": [
                            {
                                "ContainerPort": {
                                    "Ref": "ContainerPort"
                                },
                                "HostPort": {
                                    "Ref": "ContainerPort"
                                }
```

```
                        }
                    ],
                    "LogConfiguration": {
                        "LogDriver": "awslogs",
                        "Options": {
                            "mode": "non-blocking",
                            "max-buffer-size": "25m",
                            "awslogs-group": {
                                "Ref": "LogGroup"
                            },
                            "awslogs-region": {
                                "Ref": "AWS::Region"
                            },
                            "awslogs-stream-prefix": {
                                "Ref": "ServiceName"
                            }
                        }
                    }
                }
            ]
        }
    },
    "Service": {
        "Type": "AWS::ECS::Service",
        "DependsOn": "PublicLoadBalancerListener",
        "Properties": {
            "ServiceName": {
                "Ref": "ServiceName"
            },
            "Cluster": {
                "Ref": "ClusterName"
            },
            "PlacementStrategies": [
                {
                    "Field": "attribute:ecs.availability-zone",
                    "Type": "spread"
                },
                {
                    "Field": "cpu",
                    "Type": "binpack"
                }
            ],
            "CapacityProviderStrategy": [
                {
```

```
                    "Base": 0,
                    "CapacityProvider": {
                        "Ref": "CapacityProvider"
                    },
                    "Weight": 1
                }
            ],
            "NetworkConfiguration": {
                "AwsvpcConfiguration": {
                    "SecurityGroups": [
                        {
                            "Ref": "ServiceSecurityGroup"
                        }
                    ],
                    "Subnets": {
                        "Ref": "PrivateSubnetIds"
                    }
                }
            },
            "DeploymentConfiguration": {
                "MaximumPercent": 200,
                "MinimumHealthyPercent": 75
            },
            "DesiredCount": {
                "Ref": "DesiredCount"
            },
            "TaskDefinition": {
                "Ref": "TaskDefinition"
            },
            "LoadBalancers": [
                {
                    "ContainerName": {
                        "Ref": "ServiceName"
                    },
                    "ContainerPort": {
                        "Ref": "ContainerPort"
                    },
                    "TargetGroupArn": {
                        "Ref": "ServiceTargetGroup"
                    }
                }
            ]
        }
    },
```

```
        "ServiceSecurityGroup": {
            "Type": "AWS::EC2::SecurityGroup",
            "Properties": {
                "GroupDescription": "Security group for service",
                "VpcId": {
                    "Ref": "VpcId"
                }
            }
        },
        "ServiceTargetGroup": {
            "Type": "AWS::ElasticLoadBalancingV2::TargetGroup",
            "Properties": {
                "HealthCheckIntervalSeconds": 6,
                "HealthCheckPath": "/",
                "HealthCheckProtocol": "HTTP",
                "HealthCheckTimeoutSeconds": 5,
                "HealthyThresholdCount": 2,
                "TargetType": "ip",
                "Port": {
                    "Ref": "ContainerPort"
                },
                "Protocol": "HTTP",
                "UnhealthyThresholdCount": 10,
                "VpcId": {
                    "Ref": "VpcId"
                },
                "TargetGroupAttributes": [
                    {
                        "Key": "deregistration_delay.timeout_seconds",
                        "Value": 0
                    }
                ]
            }
        },
        "PublicLoadBalancerSG": {
            "Type": "AWS::EC2::SecurityGroup",
            "Properties": {
                "GroupDescription": "Access to the public facing load balancer",
                "VpcId": {
                    "Ref": "VpcId"
                },
                "SecurityGroupIngress": [
                    {
                        "CidrIp": "0.0.0.0/0",
```

```
                    "IpProtocol": -1
                }
            ]
        }
    },
    "PublicLoadBalancer": {
        "Type": "AWS::ElasticLoadBalancingV2::LoadBalancer",
        "Properties": {
            "Scheme": "internet-facing",
            "LoadBalancerAttributes": [
                {
                    "Key": "idle_timeout.timeout_seconds",
                    "Value": "30"
                }
            ],
            "Subnets": {
                "Ref": "PublicSubnetIds"
            },
            "SecurityGroups": [
                {
                    "Ref": "PublicLoadBalancerSG"
                }
            ]
        }
    },
    "PublicLoadBalancerListener": {
        "Type": "AWS::ElasticLoadBalancingV2::Listener",
        "Properties": {
            "DefaultActions": [
                {
                    "Type": "forward",
                    "ForwardConfig": {
                        "TargetGroups": [
                            {
                                "TargetGroupArn": {
                                    "Ref": "ServiceTargetGroup"
                                },
                                "Weight": 100
                            }
                        ]
                    }
                }
            ],
            "LoadBalancerArn": {
```

```
                    "Ref": "PublicLoadBalancer"
                },
                "Port": 80,
                "Protocol": "HTTP"
            }
        },
        "ServiceIngressfromLoadBalancer": {
            "Type": "AWS::EC2::SecurityGroupIngress",
            "Properties": {
                "Description": "Ingress from the public ALB",
                "GroupId": {
                    "Ref": "ServiceSecurityGroup"
                },
                "IpProtocol": -1,
                "SourceSecurityGroupId": {
                    "Ref": "PublicLoadBalancerSG"
                }
            }
        },
        "LogGroup": {
            "Type": "AWS::Logs::LogGroup"
        }
    }
}
```

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Description: >-
  An example service that deploys in AWS VPC networking mode on EC2 capacity.
  Service uses a capacity provider to request EC2 instances to run on. Service
  runs with networking in private subnets, but still accessible to the internet
  via a load balancer hosted in public subnets.
Parameters:
  VpcId:
    Type: String
    Description: The VPC that the service is running inside of
  PublicSubnetIds:
    Type: 'List<AWS::EC2::Subnet::Id>'
    Description: List of public subnet ID's to put the load balancer in
  PrivateSubnetIds:
    Type: 'List<AWS::EC2::Subnet::Id>'
    Description: List of private subnet ID's that the AWS VPC tasks are in
```

```
  ClusterName:
    Type: String
    Description: The name of the ECS cluster into which to launch capacity.
  ECSTaskExecutionRole:
    Type: String
    Description: The role used to start up an ECS task
  CapacityProvider:
    Type: String
    Description: >-
      The cluster capacity provider that the service should use to request
      capacity when it wants to start up a task
  ServiceName:
    Type: String
    Default: web
    Description: A name for the service
  ImageUrl:
    Type: String
    Default: 'public.ecr.aws/docker/library/nginx:latest'
    Description: >-
      The url of a docker image that contains the application process that will
      handle the traffic for this service
  ContainerCpu:
    Type: Number
    Default: 256
    Description: How much CPU to give the container. 1024 is 1 CPU
  ContainerMemory:
    Type: Number
    Default: 512
    Description: How much memory in megabytes to give the container
  ContainerPort:
    Type: Number
    Default: 80
    Description: What port that the application expects traffic on
  DesiredCount:
    Type: Number
    Default: 2
    Description: How many copies of the service task to run
Resources:
  TaskDefinition:
    Type: 'AWS::ECS::TaskDefinition'
    Properties:
      Family: !Ref ServiceName
      Cpu: !Ref ContainerCpu
      Memory: !Ref ContainerMemory
```

```
      NetworkMode: awsvpc
      RequiresCompatibilities:
        - EC2
      ExecutionRoleArn: !Ref ECSTaskExecutionRole
      ContainerDefinitions:
        - Name: !Ref ServiceName
          Cpu: !Ref ContainerCpu
          Memory: !Ref ContainerMemory
          Image: !Ref ImageUrl
          PortMappings:
            - ContainerPort: !Ref ContainerPort
              HostPort: !Ref ContainerPort
          LogConfiguration:
            LogDriver: awslogs
            Options:
              mode: non-blocking
              max-buffer-size: 25m
              awslogs-group: !Ref LogGroup
              awslogs-region: !Ref AWS::Region
              awslogs-stream-prefix: !Ref ServiceName
  Service:
    Type: AWS::ECS::Service
    DependsOn: PublicLoadBalancerListener
    Properties:
      ServiceName: !Ref ServiceName
      Cluster: !Ref ClusterName
      PlacementStrategies:
        - Field: 'attribute:ecs.availability-zone'
          Type: spread
        - Field: cpu
          Type: binpack
      CapacityProviderStrategy:
        - Base: 0
          CapacityProvider: !Ref CapacityProvider
          Weight: 1
      NetworkConfiguration:
        AwsvpcConfiguration:
          SecurityGroups:
            - !Ref ServiceSecurityGroup
          Subnets: !Ref PrivateSubnetIds
      DeploymentConfiguration:
        MaximumPercent: 200
        MinimumHealthyPercent: 75
      DesiredCount: !Ref DesiredCount
```

```
      TaskDefinition: !Ref TaskDefinition
      LoadBalancers:
        - ContainerName: !Ref ServiceName
          ContainerPort: !Ref ContainerPort
          TargetGroupArn: !Ref ServiceTargetGroup
  ServiceSecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: Security group for service
      VpcId: !Ref VpcId
  ServiceTargetGroup:
    Type: 'AWS::ElasticLoadBalancingV2::TargetGroup'
    Properties:
      HealthCheckIntervalSeconds: 6
      HealthCheckPath: /
      HealthCheckProtocol: HTTP
      HealthCheckTimeoutSeconds: 5
      HealthyThresholdCount: 2
      TargetType: ip
      Port: !Ref ContainerPort
      Protocol: HTTP
      UnhealthyThresholdCount: 10
      VpcId: !Ref VpcId
      TargetGroupAttributes:
        - Key: deregistration_delay.timeout_seconds
          Value: 0
  PublicLoadBalancerSG:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: Access to the public facing load balancer
      VpcId: !Ref VpcId
      SecurityGroupIngress:
        - CidrIp: 0.0.0.0/0
          IpProtocol: -1
  PublicLoadBalancer:
    Type: 'AWS::ElasticLoadBalancingV2::LoadBalancer'
    Properties:
      Scheme: internet-facing
      LoadBalancerAttributes:
        - Key: idle_timeout.timeout_seconds
          Value: '30'
      Subnets: !Ref PublicSubnetIds
      SecurityGroups:
        - !Ref PublicLoadBalancerSG
```

```
  PublicLoadBalancerListener:
    Type: 'AWS::ElasticLoadBalancingV2::Listener'
    Properties:
      DefaultActions:
        - Type: forward
          ForwardConfig:
            TargetGroups:
              - TargetGroupArn: !Ref ServiceTargetGroup
                Weight: 100
      LoadBalancerArn: !Ref PublicLoadBalancer
      Port: 80
      Protocol: HTTP
  ServiceIngressfromLoadBalancer:
    Type: 'AWS::EC2::SecurityGroupIngress'
    Properties:
      Description: Ingress from the public ALB
      GroupId: !Ref ServiceSecurityGroup
      IpProtocol: -1
      SourceSecurityGroupId: !Ref PublicLoadBalancerSG
  LogGroup:
    Type: 'AWS::Logs::LogGroup'
```

Amazon ECS の IAM ロール

CloudFormation テンプレートを使用して、Amazon ECS で使用する IAM ロールを作成できます。Amazon ECS の IAM ロールの詳細については、「Amazon ECS の IAM ロール」を参照してください。

Amazon ECS タスク実行ロール

タスク実行ロールは、ユーザーに代わって AWS API コールを実行するためのアクセス許可を Amazon ECS コンテナと Fargate エージェントに付与します。このロールは、タスクの要件に応じて必要となります。詳細については、「Amazon ECS タスク実行IAM ロール」を参照してください。

次のテンプレートを使用して、AmazonECSTaskExecutionRolePolicy マネージドポリシーを使用するシンプルなタスク実行ロールを作成できます。

JSON

```
{
```

```
"AWSTemplateFormatVersion": "2010-09-09",
"Description": "CloudFormation template for ECS Task Execution Role",
"Resources": {
  "ECSTaskExecutionRole": {
    "Type": "AWS::IAM::Role",
    "Properties": {
      "AssumeRolePolicyDocument": {
        "Statement": [
          {
            "Effect": "Allow",
            "Principal": {
              "Service": ["ecs-tasks.amazonaws.com"]
            },
            "Action": ["sts:AssumeRole"],
            "Condition": {
              "ArnLike": {
                "aws:SourceArn": {
                  "Fn::Sub": "arn:aws:ecs:${AWS::Region}:${AWS::AccountId}:*"
                }
              },
              "StringEquals": {
                "aws:SourceAccount": {
                  "Ref": "AWS::AccountId"
                }
              }
            }
          }
        ]
      },
      "Path": "/",
      "ManagedPolicyArns": [
        "arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy"
      ]
    }
  }
},
"Outputs": {
  "ECSTaskExecutionRoleARN": {
    "Description": "ARN of the ECS Task Execution Role",
    "Value": {
      "Fn::GetAtt": ["ECSTaskExecutionRole", "Arn"]
    },
    "Export": {
      "Name": {
```

```
          "Fn::Sub": "${AWS::StackName}-ECSTaskExecutionRoleARN"
        }
      }
    },
    "ECSTaskExecutionRoleName": {
      "Description": "Name of the ECS Task Execution Role",
      "Value": {
        "Ref": "ECSTaskExecutionRole"
      },
      "Export": {
        "Name": {
          "Fn::Sub": "${AWS::StackName}-ECSTaskExecutionRoleName"
        }
      }
    }
  }
}
```

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Description: 'CloudFormation template for ECS Task Execution Role'
Resources:
  ECSTaskExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Effect: Allow
            Principal:
              Service: [ecs-tasks.amazonaws.com]
            Action: ['sts:AssumeRole']
            Condition:
              ArnLike:
                aws:SourceArn: !Sub arn:aws:ecs:${AWS::Region}:${AWS::AccountId}:*
              StringEquals:
                aws:SourceAccount: !Ref AWS::AccountId
      Path: /
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy
Outputs:
  ECSTaskExecutionRoleARN:
    Description: ARN of the ECS Task Execution Role
```

```
    Value: !GetAtt ECSTaskExecutionRole.Arn
    Export:
      Name: !Sub "${AWS::StackName}-ECSTaskExecutionRoleARN"
  ECSTaskExecutionRoleName:
    Description: Name of the ECS Task Execution Role
    Value: !Ref ECSTaskExecutionRole
    Export:
      Name: !Sub "${AWS::StackName}-ECSTaskExecutionRoleName"
```

Amazon ECS のベストプラクティス

以下のページを使用して、Amazon ECS ネットワーキングに関する最も重要な運用上のベストプラクティスを学習できます。

ベストプラクティスの概要	詳細	
アプリケーションをインターネットに接続する	Amazon ECS アプリケーションをインターネットに接続する	
インターネットから Amazon ECS へのインバウンド接続を受信する	インターネットから Amazon ECS へのインバウンド接続を受信するためのベストプラクティス	
Amazon ECS を VPC 内の他の AWS サービスに接続する	Amazon ECS を VPC 内から AWS サービスに接続するためのベストプラクティス	
AWS アカウント と VPC 間のネットワークサービス	AWS アカウントおよび VPC の間で Amazon ECS サービスをネットワーキングするためのベストプラクティス	
ネットワークの問題のトラブルシューティング	AWSAmazon ECS ネットワーキングのトラブルシューティングのための サービス	

以下のページを使用して、Fargate on Amazon ECS に関する最も重要な運用上のベストプラクティスを学習できます。

ベストプラクティスの概要	詳細	
Fargate のセキュリティ	Amazon ECS の Fargate セキュリティのベストプラクティス	
Fargate のセキュリティに関する考慮事項	Amazon ECS の Fargate のセキュリティに関する考慮事項	
Fargate 上の Linux コンテナにおけるコンテナイメージのプル動作	Fargate 上の Linux コンテナにおける Amazon ECS のコンテナイメージのプル動作	
Fargate 上の Windows コンテナにおけるコンテナイメージのプル動作	Fargate 上の Windows コンテナにおける Amazon ECS のコンテナイメージのプル動作	
Fargate のタスク廃止	AWS Fargate on Amazon ECS のタスクの廃止とメンテナンス	

以下のページを使用して、タスク定義に関する最も重要な運用上のベストプラクティスを学習できます。

ベストプラクティスの概要	詳細	
コンテナイメージ	Amazon ECS コンテナイメージのベストプラクティス	
タスクサイズ	Amazon ECS タスクサイズのベストプラクティス	
ボリュームのベストプラクティス	Amazon ECS タスクのストレージオプション	

以下のページを使用して、クラスターとキャパシティに関する最も重要な運用上のベストプラクティスを学習できます。

ベストプラクティスの概要	詳細	
Fargate のセキュリティ	Amazon ECS の Fargate セキュリティのベストプラクティス	
EC2 コンテナインスタンスのセキュリティに関する考慮事項	Amazon ECS の Amazon EC2 コンテナインスタンスのセキュリティに関する考慮事項	
クラスターの自動スケーリング	Amazon ECS クラスターの自動スケーリングの最適化	
大規模な運用	Amazon ECS を大規模に運用する	
自動スケーリングと容量管理	Amazon ECS 自動スケーリングとキャパシティ管理におけるベストプラクティス	
Amazon ECS マネージドインスタンスにおけるインスタンス選択のベストプラクティス	Amazon ECS マネージドインスタンスにおけるインスタンス選択のベストプラクティス	

以下のページを使用して、タスクとサービスに関する最も重要な運用上のベストプラクティスを学習できます。

ベストプラクティスの概要	詳細	
タスクの起動時間を最適化する	Amazon ECS タスクの起動時間を最適化する	
サービスパラメータ	Amazon ECS サービスパラメータのベストプラクティス	

ベストプラクティスの概要	詳細	
ロードバランサーのヘルスチェックパラメータを最適化する	Amazon ECS のロードバランサーのヘルスチェックパラメータを最適化する	
ロードバランサーの Connection Draining パラメータを最適化する	Amazon ECS のロードバランサー Connection Draining パラメータを最適化する	
サービス自動スケーリングを最適化する	Amazon ECS サービスの自動スケーリングの最適化	

以下のページを使用して、セキュリティに関する最も重要な運用上のベストプラクティスを学習できます。

ベストプラクティスの概要	詳細	
ネットワークセキュリティ	Amazon ECS のネットワークセキュリティのベストプラクティス	
タスクとコンテナのセキュリティ	Amazon ECS タスクおよびコンテナのセキュリティのベストプラクティス	

Amazon ECS のソリューションの構築

アプリケーションをコンテナ上で実行できるように設計する必要があります。コンテナとは、ソフトウェアアプリケーションの実行に必要なものをすべて保持する、標準化されたソフトウェア開発の単位です。これには、関連するコード、ランタイム、システムツール、およびシステムライブラリが含まれます。コンテナは、イメージと呼ばれる読み取り専用テンプレートから作成されます。コンテナイメージは、アプリケーションとその依存関係を含む静的アーティファクトです。イメージは通常 Dockerfile から構築されます。Dockerfile は、コンテナを構築するための手順を含むプレーンテキストファイルです。これらのイメージは、Amazon ECR など、構築された後にダウンロード可能な場所であるレジストリに保存されます。

イメージを作成して保存した後、Amazon ECS のタスク定義を作成します。タスク定義はアプリケーションのブループリントです。これは、アプリケーションを形成するパラメータと 1 つ以上のコンテナを記述する JSON 形式のテキストファイルです。例えば、オペレーティングシステムのパラメータ、使用するコンテナ、アプリケーションで開くポート、タスクのコンテナで使用するデータボリュームなどの指定に使用できます。タスク定義で使用できる特定のパラメータは、お客様の特定のアプリケーションのニーズによって異なります。

タスク定義を定義したら、それをサービスまたはタスクとしてクラスターにデプロイします。クラスターは、クラスターに登録されているキャパシティインフラストラクチャ上で実行されるタスクまたはサービスを論理的にグループ化したものです。

タスクはクラスター内のタスク定義のインスタンス化です。スタンドアロンタスクを実行することもできますし、サービスの一部としてタスクを実行することもできます。Amazon ECS サービスを使用すると、Amazon ECS クラスターで必要な数のタスクを同時に実行して維持できます。仕組みとしては、タスクがいずれかの理由で失敗または停止した場合に、Amazon ECS サービススケジューラがタスク定義に基づいて別のインスタンスを起動することによって動作します。これは、それを置き換え、サービス内の必要な数のタスクを維持するために行われます。

コンテナエージェントは Amazon ECS クラスター内の各コンテナインスタンス上で実行されます。エージェントは、現在実行中のタスクとリソースのコンテナの使用率に関する情報を Amazon ECS に送信します。Amazon ECS からのリクエストを受信するたびに、タスクを開始および停止します。

タスクまたはサービスをデプロイしたら、次のいずれかのツールを使用してデプロイとアプリケーションを監視できます。

- CloudWatch

- Runtime Monitoring

Capacity

容量は、コンテナが実行されるインフラストラクチャです。Amazon ECS では、クラスター用に 3 つのインフラストラクチャタイプが用意されています。

- Amazon ECS マネージドインスタンス – AWS はプロビジョニング、パッチ適用、スケーリングなど、アカウントで動作している基盤となる Amazon EC2 インスタンスを完全に管理します。このオプションを使用すると、パフォーマンス、コスト効率、運用のシンプルさについて最適なバランスが得られます。
- サーバーレス (Fargate) – インフラストラクチャを管理せずに、タスクで使用されるリソースに対してのみ支払います。可変ワークロードやすぐに始めたいときに最適です。
- Amazon EC2 インスタンス – 基盤となる Amazon EC2 インスタンスの管理、例えばインスタンスの選択、設定、メンテナンスなどを直接行います。

Amazon ECS マネージドインスタンスは、次のような場合に使用します。

- 基盤となるインフラストラクチャをより細かく制御できる Fargate のシンプルさを求めている場合。
- 予測可能なパフォーマンスとコスト最適化が必要な場合。
- AWS を使用して、柔軟に対応しながらインフラストラクチャを管理したい場合。

Fargate は、次のような場合に使用します。

- インフラストラクチャを管理せずにアプリケーションに集中したい場合。
- 可変または予測不可能なワークロードがある場合。
- コンテナの使用を開始していて、シンプルなデプロイオプションが必要な場合。

Amazon EC2 インスタンスは、次のような場合に使用します。

- 特殊なハードウェア要件 (GPU アクセラレーション、高性能コンピューティング) が必要な場合。
- キャパシティ予約または特定のインスタンスタイプが必要な場合。
- 特権機能またはカスタム AMI が必要な場合。

クラスターを作成する際、インフラストラクチャを指定します。タスク定義を登録する際、インフラストラクチャタイプも指定します。タスク定義では、インフラストラクチャを「起動タイプ」と呼びます。キャパシティプロバイダーを使用することもできます。

サービスエンドポイント

サービスエンドポイントは、インターネットプロトコルバージョン 4 (IPv4) またはインターネットプロトコルバージョン 6 (IPv6) を使用してプログラムでサービスに接続するために使用する Amazon ECS のエントリポイントの URL です。デフォルトでは、Amazon ECS にプログラムで接続するために行うリクエストは、IPv4 リクエストのみをサポートするサービスエンドポイントを使用します。IPv4 と IPv6 リクエストのどちらからでもプログラムで Amazon ECS に接続したい場合は、デュアルスタックエンドポイントを使用できます。デュアルスタックのエンドポイントの詳細については、「Amazon ECS デュアルスタックエンドポイントの使用」を参照してください。

ネットワーク

AWS リソースはサブネットに作成されます。EC2 インスタンスを使用する場合、Amazon ECS はクラスターの作成時に指定したサブネットでインスタンスを起動します。タスクはインスタンスのサブネットで実行されます。Fargate またはオンプレミス仮想マシンの場合は、タスクを実行するとき、またはサービスを作成するときにサブネットを指定します。

アプリケーションに応じて、サブネットはプライベートサブネットでもパブリックサブネットでもよく、サブネットは次の AWS リソースのどれにあってもかまいません。

- アベイラビリティーゾーン
- ローカルゾーン
- Wavelength Zone
- AWS リージョン
- AWS Outposts

詳細については「共有サブネット、Local Zones、および Wavelength Zones の Amazon ECS アプリケーション」または「AWS OutpostsのAmazon Elastic Container Service」を参照してください。

次のいずれかの方法を使用して、アプリケーションをインターネットに接続できます。

- インターネットゲートウェイを持つパブリックサブネット

大量の帯域幅や最小のレイテンシーを必要とするパブリックアプリケーションがある場合は、パブリックサブネットを使用してください。該当するシナリオには、ビデオストリーミングやゲームサービスが含まれます。

- NAT ゲートウェイを持つプライベートサブネット

  外部からの直接アクセスからコンテナを保護するには、プライベートサブネットを使用してください。該当するシナリオには、支払い処理システムや、ユーザーデータとパスワードを格納するコンテナなどがあります。

- AWS PrivateLink

  トラフィックをパブリックインターネットに公開することのない、VPC、AWS サービス、オンプレミスネットワークの間のプライベート接続を提供するには、AWS PrivateLink を使用してください。

機能アクセス

Amazon ECS アカウント設定を使用して、次の機能にアクセスできます。

- Container Insights

  CloudWatch Container Insights は、コンテナ化されたアプリケーションとマイクロサービスのメトリクスとログを収集、集約、要約します。このメトリクスには、CPU、メモリ、ディスク、ネットワークなどのリソース使用率が含まれます。

- `awsvpc` トランキング

  特定の EC2 インスタンスタイプでは、新しく起動したコンテナインスタンスで追加のネットワークインターフェイス (ENI) を使用できます。

- タグ付け認可

  ユーザーには、リソースを作成するアクションの許可が必要です (`ecsCreateCluster` など)。リソース作成アクションでタグが指定されている場合、AWS は `ecs:TagResource` アクションに対して追加の認可を実行して、ユーザーまたはロールがタグを作成するための許可を持っているかどうかを確認します。

- Fargate FIPS-140 コンプライアンス

  Fargate は、機密情報を保護する暗号モジュールのセキュリティ要件を指定する連邦情報処理標準 (FIPS-140) をサポートしています。これは現在の米国およびカナダの政府標準であ

り、Federal Information Security Management Act (FISMA) または Federal Risk and Authorization Management Program (FedRAMP) への準拠が要求されるシステムに適用されます。

- Fargate タスクの廃止時間の変更

  Fargate タスクが廃止されるまでの待機時間を設定できます。

- デュアルスタック VPC

  タスクが IPv4、IPv6、またはその両方を介して通信できるようにします。

- Amazon リソースネーム (ARN) 形式

  タグ付け認可などの特定の機能には、新しい Amazon リソースネーム (ARN) 形式が必要です。

詳細については、「アカウント設定による Amazon ECS 機能へのアクセス」を参照してください。

IAM ロール

IAM ロールは、特定の許可があり、アカウントで作成できるもう 1 つの IAM アイデンティティです。Amazon ECS では、コンテナやサービスなどの Amazon ECS リソースにアクセス許可を付与するロールを作成できます。

Amazon ECS の一部の機能にはロールが必要です。詳細については、「Amazon ECS の IAM ロール」を参照してください。

ログ記録

ログ記録およびモニタリングは、Amazon ECS ワークロードの信頼性、可用性、パフォーマンスを維持する上で重要な要素です。以下のオプションが利用できます。

- Amazon CloudWatch ログ- ログを にルーティングするAmazon CloudWatch
- Amazon ECS 用 FireLens - ログを AWS サービスまたは AWS Partner Network の宛先にルーティングして、ログの保存と分析を行います。AWS Partner Network は、プログラム、専門知識、リソースを活用して顧客向けサービスの構築、マーケティング、販売を行うパートナーのグローバルコミュニティです。

コンテナイメージのベストプラクティス

Amazon ECS コンテナイメージの主な原則は次のとおりです。

- イメージにすべての依存関係を含める
- コンテナごとに 1 つのプロセスを実行する
- 正常なシャットダウンのために SIGTERM を使用する
- ログを stdoutと stderr に書き込む
- 本番環境では latest を避け、一意のタグを使用する

Amazon ECS の起動タイプとキャパシティプロバイダー

Amazon ECS には、ワークロードのキャパシティを設定するための 2 つの方法があります。起動タイプかキャパシティプロバイダーを使用できます。起動タイプには、EC2、Fargate、External があります。キャパシティプロバイダーは、キャパシティ管理のための高い柔軟性と高度な機能を備えています。Fargate および Fargate スポットキャパシティプロバイダーを使用するサーバーレスコンピューティング、Auto Scaling グループキャパシティプロバイダーを使用するセルフマネージド EC2 インスタンス、または Fargate のシンプルさと EC2 コンピューティングの柔軟性を組み合わせた Amazon ECS マネージドインスタンスキャパシティプロバイダーを使用するフルマネージドコンピューティングでワークロードを実行できます。キャパシティプロバイダーは、リソース割り当てをより適切に制御し、パフォーマンスとコストの両方を最適化することができます。従来の起動タイプと比較すると、ワークロードのキャパシティの設定のために推奨される方法です。キャパシティプロバイダーと起動タイプの違いを理解するために、以下を参考にしてください。

ベストプラクティス

ベストプラクティスは以下のとおりです。

起動タイプを使用してインフラストラクチャの互換性を定義する

起動タイプでは、タスクとサービスを実行するインフラストラクチャを定義します。タスクを定義するときは、RequiresCompatibilities を指定して、タスクと互換性のある 1 つ以上の起動タイプを含めます。起動タイプとして、EC2、Fargate、External、Amazon ECS マネージドインスタンスを使用できます。起動タイプを使用してタスクまたはサービスを実行することもできますが、起動タイプはタスク定義での互換性の定義のみに使用し、タスクやサービスの起動にはキャパシティプロバイダーを使用することをお勧めします。タスクの互換性を定義には、1 つ以上の起動タイプを選択することができます。

キャパシティプロバイダーを使用してコンピューティングキャパシティーを設定する

タスクまたはサービスを起動するときは、キャパシティプロバイダー戦略を設定します。Amazon ECS でサポートされているキャパシティプロバイダーは、Fargate および

FARGATE_SPOT、セルフマネージド EC2 インスタンス用の Auto Scaling グループ、Amazon ECS マネージドインスタンスです。スポットフリートはキャパシティプロバイダーとしてのみ使用でき、起動タイプとしては使用できないことに注意してください。クラスター内には、Amazon ECS マネージドインスタンスキャパシティプロバイダーまたは Amazon EC2 Auto Scaling グループキャパシティプロバイダーを 1 つ、または複数作成できます。Fargate および Fargate Spot キャパシティプロバイダーは、すべてのクラスターで Amazon ECS によって作成および管理されるため、作成する必要はありません。クラスターではすべてのタイプのキャパシティプロバイダーを組み合わせることができますが、キャパシティプロバイダー戦略では異なるタイプのキャパシティプロバイダーを組み合わせることはできません。

サービスのキャパシティを更新する

サービスのキャパシティプロバイダー戦略を更新するだけで、サービスをコンピューティングタイプ間で移動させることができます。

サービスのミュータビリティ

Amazon ECS は、異なるキャパシティプロバイダー間のサービスの更新をサポートしています。そのため、次のことが可能になります。

- 起動タイプからキャパシティプロバイダーへのシームレスな更新
- 異なるタイプのキャパシティプロバイダー間の移行
- サービスの再作成なしでさまざまなコンピューティングオプションをテスト

以下に、プロセスの概要を示します。

1. タスク定義を更新する – 例えば、`requiresCompatibilities` にターゲットキャパシティプロバイダー (`MANAGED_INSTANCES` など) が含まれていることを確認します。

   > **Note**
   >
   > タスク定義は、ターゲットキャパシティプロバイダーの互換性検証に合格する必要があります。タスク定義バージョンの `requiresCompatibilities` チェックが失敗した場合、`UpdateService` 呼び出しは失敗します。

2. キャパシティプロバイダーを作成する – カスタム Amazon EC2 Auto Scaling グループを使用する場合は、キャパシティプロバイダーを作成します。

3. サービスを更新する – 起動タイプの代わりにキャパシティプロバイダー戦略を使用するようにサービスを変更します。
4. デプロイを検証する – タスクが正常にデプロイされたことを確認します。
5. モニタリングして最適化する — 必要に応じてキャパシティプロバイダーの設定を調整します。

キャパシティプロバイダーからキャパシティプロバイダー

キャパシティプロバイダーからキャパシティプロバイダーへの更新はすべてサポートされています。

- Amazon EC2 Auto Scaling グループキャパシティプロバイダーから Amazon ECS マネージドインスタンス
- Fargate キャパシティプロバイダーから Amazon ECS マネージドインスタンス
- Amazon EC2 Auto Scaling グループキャパシティプロバイダーから Fargate キャパシティプロバイダー
- Amazon ECS マネージドインスタンスから Fargate キャパシティプロバイダー
- Fargate キャパシティプロバイダーから Amazon EC2 Auto Scaling グループキャパシティプロバイダー
- Amazon ECS マネージドインスタンスから Amazon EC2 Auto Scaling グループキャパシティプロバイダー

起動タイプからキャパシティプロバイダー

起動タイプからキャパシティプロバイダーへの更新はすべてサポートされています。

- EC2 起動タイプから Amazon ECS マネージドインスタンス
- Fargate 起動タイプから Amazon ECS マネージドインスタンス
- EC2 起動タイプから Fargate キャパシティプロバイダー
- EC2 起動タイプから EC2 Auto Scaling グループキャパシティプロバイダー
- Fargate 起動タイプから Amazon EC2 Auto Scaling グループキャパシティプロバイダー
- Fargate 起動タイプから Fargate キャパシティプロバイダー
- 外部起動タイプから Amazon ECS マネージドインスタンス
- 外部起動タイプから Fargate キャパシティプロバイダー
- 外部起動タイプから Amazon EC2 Auto Scaling グループキャパシティプロバイダー

起動タイプから起動タイプ

起動タイプから起動タイプへの更新はサポートされていません。

- EC2 起動タイプから Fargate 起動タイプ (代わりに Fargate キャパシティプロバイダーを使用)
- Fargate 起動タイプから EC2 起動タイプ (代わりに Amazon EC2 Auto Scaling グループキャパシティプロバイダーを使用)

起動タイプ間での移行の代わりに、機能の強化と将来の互換性の確保のために、同等のキャパシティプロバイダーに移行します。

キャパシティプロバイダーから起動タイプ

キャパシティプロバイダーから起動タイプへの更新はサポートされていません。最初に起動タイプを使用してサービスを作成し、次にキャパシティプロバイダーを使用するようにサービスを更新した場合は、そのサービスを最初の起動タイプに戻すことができます。戻すには、`UpdateService` を呼び出すときに空のリストを `capacityProviderStrategy` として渡します。

> **Note**
>
> これにより、サービスは、サービスが最初に作成されたときに指定された起動タイプに戻されます。この方法を使用して、サービスを別の起動タイプに切り替えることはできません。

Amazon ECS デュアルスタックエンドポイントの使用

Amazon ECS デュアルスタックエンドポイントは、インターネットプロトコルバージョン 4 (IPv4) およびインターネットプロトコルバージョン 6 (IPv6) を介した Amazon ECS へのリクエストをサポートします。すべての Amazon ECS エンドポイントのリストについては、「AWS 全般のリファレンス」の「Amazon ECS のエンドポイントとクォータ」を参照してください。

REST API を使用する場合、エンドポイント名 (URI) を使用して直接 Amazon ECS エンドポイントにアクセスします。Amazon ECS はリージョン固有のデュアルスタックエンドポイント名のみをサポートしており、名前の一部としてリージョンを指定する必要があります。

デュアルスタックエンドポイント名には、次の命名規則を使用します: `ecs.region.api.aws`。

AWS Command Line Interface (AWS CLI) や AWS SDK を使用している場合、パラメータまたはフラグを使ってデュアルスタックエンドポイントに変更できます。設定ファイル内の Amazon ECS エン

ドポイントをオーバーライドして、デュアルスタックエンドポイントを直接指定することもできます。

以下のセクションでは、AWS CLI、AWS SDK、および REST API からデュアルスタックエンドポイントを使用する方法について説明します。

トピック

- AWS CLI からのデュアルスタックのエンドポイントの使用
- AWS SDK からデュアルスタックのエンドポイントを使用する
- REST API からのデュアルスタックのエンドポイントの使用

AWS CLI からのデュアルスタックのエンドポイントの使用

このセクションでは、デュアルスタックのエンドポイントへリクエストするのに使用される AWS CLI コマンドの例を示します。AWS CLI のインストールまたは最新バージョンへのアップグレードについては、「バージョン 2 向けの AWS Command Line Interface ユーザーガイド」の「AWS CLI の最新バージョンのインストールまたは更新」を参照してください。

デュアルスタックエンドポイントを使用する際には、AWS CLI の `config` ファイルの設定値 `use_dualstack_endpoint` を `true` に設定することで、`ecs` AWS CLI コマンドによるすべての Amazon ECS リクエストを指定されたリージョンのデュアルスタックエンドポイントに転送することができます。`--region` オプションを使用して `config` ファイルまたはコマンドでリージョンを指定します。AWS CLI の構成ファイルの詳細については、「バージョン 2 向けの AWS Command Line Interface ユーザーガイド」の「AWS CLI の構成および認証ファイル設定」を参照してください。

特定の AWS CLI コマンドについてデュアルスタックエンドポイントを使用するには、次のいずれかの方法を使用できます。

- 任意の `ecs` コマンドについて `--endpoint-url` パラメータを `https://ecs.aws-region.api.aws` もしくは `http://ecs.aws-region.api.aws` に設定することで、個別のコマンド単位でデュアルスタックエンドポイントを使用できます。

  次のコマンド例では、使用可能なすべてのクラスターを一覧表示し、リクエストにデュアルスタックエンドポイントを使用しています。

  ```
  $ aws ecs list-clusters --endpoint-url https://ecs.aws-region.api.aws
  ```

- AWS Config ファイル内で別々のプロファイルを設定する。たとえば、use_dualstack_endpoint を true に設定するプロファイルと use_dualstack_endpoint を設定しないプロファイルを作成します。コマンドを実行する際には、デュアルスタックのエンドポイントを使用するかどうかによって、適切なプロファイルを指定します。

AWS SDK からデュアルスタックのエンドポイントを使用する

このセクションでは、AWS SDK を使用してデュアルスタックのエンドポイントにアクセスする方法の例を示します。

AWS SDK for Java 2.x

次の例は、AWS SDK for Java 2.x を使用して us-east-1 リージョンのデュアルスタックエンドポイントを指定する方法を示しています。

```
Region region = Region.US_EAST_1
EcsClient client =
 EcsClient.builder().region(region).dualstackEnabled(true).build();
```

AWS SDK for Go

次の例は、AWS SDK for Go を使用して us-east-1 リージョンのデュアルスタックエンドポイントを指定する方法を示しています。

```
sess := session.Must(session.NewSession())
svc := ecs.New(sess, &aws.Config{
    Region: aws.String(endpoints.UsEast1RegionID),
    Endpoint: aws.String("https://ecs.us-east-1.api.aws")
})
```

詳細については、「AWS SDK とツールリファレンスガイド」の「デュアルスタックおよび FIPS エンドポイント」を参照してください。

REST API からのデュアルスタックのエンドポイントの使用

REST API を使用する場合、リクエスト内でデュアルスタックエンドポイントを指定することで、特定のデュアルスタックエンドポイントに直接アクセスできます。次の例では、デュアルスタックエン

ドポイントを使用して、us-east-1 リージョン内のすべての Amazon ECS クラスターを一覧表示しています。

```
POST / HTTP/1.1
Host: ecs.us-east-1.api.aws
Accept-Encoding: identity
Content-Length: 2
X-Amz-Target: AmazonEC2ContainerServiceV20141113.ListClusters
X-Amz-Date: 20150429T170621Z
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{}
```

共有サブネット、Local Zones、および Wavelength Zones の Amazon ECS アプリケーション

Amazon ECS は、低レイテンシーまたはローカルデータ処理が必要な場合に、Local Zones、Wavelength Zone、および AWS Outposts を利用するワークロードをサポートします。

- Local Zones を AWS リージョン の拡張に使用して、エンドユーザーにより近い複数の場所にリソースを配置できます。
- Wavelength Zone を使用すると、5G デバイスやエンドユーザーに非常に低いレイテンシーを提供するアプリケーションを構築できます。Wavelength は標準の AWS コンピューティングおよびストレージサービスを通信事業者の 5G ネットワークのエッジにデプロイします。
- AWS Outposts では、ネイティブの AWS のサービス、インフラストラクチャ、運用モデルをほぼすべてのデータセンター、コロケーションスペース、オンプレミスの施設で利用できます。

> **Important**
>
> AWS Fargate ワークロードの Amazon ECS は、現時点では、Local Zones、Wavelength Zone、または AWS Outposts ではサポートされていません。

ローカルゾーン、Wavelength ゾーン、AWS Outposts の違いについては、「AWS Wavelength のよくある質問」の「低レイテンシーやローカルデータ処理を必要とするアプリケーションに、AWS

Wavelength ゾーン、AWS ローカルゾーン、または AWS Outposts をいつ使用するとよいですか?」を参照してください。

共有サブネット

VPC 共有を使用して、同じ AWS Organizations 内でサブネットを他の AWS アカウントと共有できます。

EC2 には共有 VPC を使用できますが、次の点を考慮してください。

- VPC サブネットの所有者は、参加者アカウントで Amazon ECS リソース用にサブネットを使用する前に、そのサブネットをそのアカウントに共有する必要があります。
- VPC のデフォルトセキュリティグループは所有者に属しているため、コンテナインスタンスには VPC のデフォルトセキュリティグループを使用できません。また、参加者は、他の参加者または所有者が所有するセキュリティグループを使用してインスタンスを起動することはできません。
- 共有サブネットでは、参加者と所有者がそれぞれのアカウント内のセキュリティグループを個別に管理します。サブネットの所有者は、参加者が作成したセキュリティグループを表示できますが、これらのグループに対してアクションを実行することはできません。サブネットの所有者がこれらのセキュリティグループの削除または変更を希望する場合は、セキュリティグループを作成した参加者がそのアクションを実行する必要があります。
- 共有 VPC の所有者は、参加者が共有サブネット内に作成したクラスターを表示、更新、削除することはできません。これは、アカウントごとに異なるアクセス権を持つ VPC リソースに加えて適用されます。詳細については、「Amazon VPC ユーザーガイド」の「所有者および参加者の責任と権限」を参照してください。

Fargate には共有 VPC を使用できますが、次の点を考慮してください。

- VPC サブネットの所有者は、参加者アカウントで Amazon ECS リソース用にサブネットを使用する前に、そのサブネットをそのアカウントに共有する必要があります。
- VPC のデフォルトセキュリティグループは所有者に属しているため、デフォルトのセキュリティグループを使用してサービスを作成したりタスクを実行したりすることはできません。また、参加者は、他の参加者または所有者が所有するセキュリティグループを使用してサービスを作成したりタスクを実行したりすることはできません。
- 共有サブネットでは、参加者と所有者がそれぞれのアカウント内のセキュリティグループを個別に管理します。サブネットの所有者は、参加者が作成したセキュリティグループを表示できますが、これらのグループに対してアクションを実行することはできません。サブネットの所有者がこれら

のセキュリティグループの削除または変更を希望する場合は、セキュリティグループを作成した参加者がそのアクションを実行する必要があります。

- 共有 VPC の所有者は、参加者が共有サブネット内に作成したクラスターを表示、更新、削除することはできません。これは、アカウントごとに異なるアクセス権を持つ VPC リソースに加えて適用されます。詳細については、「Amazon VPC ユーザーガイド」の「所有者および参加者の責任と権限」を参照してください。

VPC サブネット共有の詳細については、「Amazon VPC ユーザーガイド」の「VPC を他のアカウントと共有する」を参照してください。

ローカルゾーン

Local Zones は、ユーザーに近い場所に位置する AWS リージョンの拡張です。Local Zones は、インターネットへの独自の接続を持ち、Direct Connect をサポートします。Local Zones で作成したリソースは、低いレイテンシーの通信をローカルユーザーに提供できます。詳細については、「AWS ローカルゾーン」を参照してください。

Local Zones を表すには、リージョンコードに続けて場所を示す識別子を使用します (例: `us-west-2-lax-1a`)。

Local Zones を使用するには、まずゾーンにオプトインする必要があります。オプトインしたら、Local Zones に Amazon VPC およびサブネットを作成する必要があります。

Amazon ECS クラスターとタスクに使用する Amazon EC2 インスタンス、Amazon FSx ファイルサーバー、Application Load Balancer を起動できます。

詳細については、AWS ローカルゾーンユーザーガイドの「AWS ローカルゾーンとは」を参照してください。

Wavelength Zone

AWS Wavelength を使用することで、モバイルデバイスおよびエンドユーザー向けに、非常にレイテンシーが低いアプリケーションを構築できます。Wavelength は標準の AWS コンピューティングおよびストレージサービスを通信事業者の 5G ネットワークのエッジにデプロイします。Amazon Virtual Private Cloud は、1 つまたは複数の Wavelength Zone に拡張できます。その後、Amazon EC2 インスタンスなどの AWS リソースを使用して、極めて低いレイテンシーやリージョンの AWS のサービス への接続を必要とするアプリケーションを実行できます。

Wavelength Zone は、Wavelength インフラストラクチャをデプロイする先のキャリアロケーション内の独立したゾーンです。Wavelength Zone は、AWS リージョンに関連付けられています。Wavelength Zone はリージョンの論理的な拡張であり、リージョンの制御プレーンによって管理されます。

Wavelength Zone は、リージョンコードに続けて Wavelength Zone を示す識別子で表されます (例: `us-east-1-wl1-bos-wlz-1`)。

Wavelength Zone を使用するには、まずゾーンにオプトインする必要があります。オプトインしたら、Wavelength Zone に Amazon VPC およびサブネットを作成する必要があります。その後、Amazon ECS クラスターとタスクに使用する Amazon EC2 インスタンスを Zone で起動することができます。

詳細については、AWS Wavelengthデベロッパーガイドの「の使用開始AWS Wavelength」を参照してください。

Wavelength Zone は、すべての AWS リージョンで利用できるわけではありません。Wavelength Zone をサポートするリージョンについてはAWS Wavelength デベロッパーガイドの利用可能な Wavelength Zone をご参照ください。

AWS OutpostsのAmazon Elastic Container Service

AWS Outposts は、オンプレミス施設でネイティブ AWS サービス、インフラストラクチャ、運用モデルを有効にします。AWS Outposts 環境では、AWS で使用するのと同じ AWS クラウド API、ツール、インフラストラクチャを使用できます。

AWS Outposts の Amazon ECS は、オンプレミスのデータやアプリケーションの近くで実行が必要な、低レイテンシーのワークロードに最適です。

AWS Outposts については AWS Outpostsユーザーガイドを参照してください。

考慮事項

AWS Outposts で Amazon ECS を使用する際の考慮事項は以下のとおりです。

- Amazon Elastic Container Registry、AWS Identity and Access Management、Network Load Balancer は、AWS Outposts ではなく AWS リージョンで実行されます。これにより、これらのサービスとコンテナ間のレイテンシーが増加します。
- AWS Fargate は で使用できません。AWS Outposts

以下に、AWS Outposts のネットワーク接続に関する考慮事項を示します。

- AWS Outposts とその AWS リージョン間のネットワーク接続が失われた場合、クラスターは引き続き実行されます。ただし、接続が復元されるまで、新しいクラスターを作成したり、既存のクラスターで新しいアクションを実行したりすることはできません。インスタンスに障害が発生した場合、インスタンスは自動的に置き換えられません。The CloudWatch Logs エージェントは、ログとイベントデータを更新できません。
- AWS OutpostsとAWS リージョン間で、信頼性が高く、可用性が高く、低レイテンシーの接続を提供することをお勧めします。

前提条件

AWS Outposts で Amazon ECS を使用するための前提条件は以下のとおりです。

- オンプレミスのデータセンターに Outpost をインストールして設定しておく必要があります。
- Outpost とその AWS リージョンとの間に、信頼できるネットワーク接続が必要です。

AWS Outposts でのクラスター作成の概要

以下は、設定の概要です。

1. AWS Outposts に対する権限を持つロールとポリシーを作成します。
2. AWS Outposts の権限を持つ IAM インスタンスプロファイルを作成します。
3. VPC を作成、または AWS Outposts と同じリージョンにある既存の VPC を使用します。
4. サブネットを作成、または AWS Outposts に関連付けられている既存のサブネットを使用します。

   これは、コンテナインスタンスが実行されるサブネットです。
5. クラスター内にあるコンテナインスタンスのセキュリティグループを作成します。
6. Amazon ECS クラスターを作成します。
7. クラスターでインスタンスを起動するための、Amazon ECS コンテナエージェント環境変数を定義します。
8. コンテナを実行します。

Amazon ECS を AWS Outposts に統合する方法の詳細については、「Extend Amazon ECS across two AWS Outposts rack」を参照してください。

次の例では、AWS Outposts に Amazon ECS クラスターを作成します。

1. AWS Outposts に対する権限を持つロールとポリシーを作成します。

   role-policy.json ファイルは、リソースに対する効果とアクションを含むポリシードキュメントです。ファイル形式の情報については、「IAM API リファレンス」の「PutRolePolicy」を参照してください。

   ```
   aws iam create-role --role-name ecsRole \
       --assume-role-policy-document file://ecs-policy.json
   aws iam put-role-policy --role-name ecsRole --policy-name ecsRolePolicy \
       --policy-document file://role-policy.json
   ```

2. AWS Outposts の権限を持つ IAM インスタンスプロファイルを作成します。

   ```
   aws iam create-instance-profile --instance-profile-name outpost
   aws iam add-role-to-instance-profile --instance-profile-name outpost \
       --role-name ecsRole
   ```

3. VPC を作成します。

   ```
   aws ec2 create-vpc --cidr-block 10.0.0.0/16
   ```

4. AWS Outposts に関連付けられたサブネットを作成します。

   ```
   aws ec2 create-subnet \
       --cidr-block 10.0.3.0/24 \
       --vpc-id vpc-xxxxxxxx \
       --outpost-arn arn:aws:outposts:us-west-2:123456789012:outpost/op-
   xxxxxxxxxxxxxxxx \
       --availability-zone-id usw2-az1
   ```

5. コンテナインスタンスのセキュリティグループを作成し、AWS Outposts に適切な CIDR 範囲を指定します。(この手順は、AWS Outposts では異なります。)

   ```
   aws ec2 create-security-group --group-name MyOutpostSG
   aws ec2 authorize-security-group-ingress --group-name MyOutpostSG --protocol tcp \
       --port 22 --cidr 10.0.3.0/24
   aws ec2 authorize-security-group-ingress --group-name MyOutpostSG --protocol tcp \
   ```

```
    --port 80 --cidr 10.0.3.0/24
```

6. クラスターを作成します。
7. Amazon ECS コンテナエージェント環境変数を定義して、前のステップで作成したクラスターにインスタンスを起動し、クラスター (例えば、クラスターが Outpost 用であることを示す Outpost など) を識別するのに役立つタグを追加し、定義します。

```
#! /bin/bash
cat << 'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_IMAGE_PULL_BEHAVIOR=prefer-cached
ECS_CONTAINER_INSTANCE_TAGS={"environment": "Outpost"}
EOF
```

> **Note**
>
> リージョン内の Amazon ECR からコンテナイメージをプルすることによる遅延を回避するには、イメージキャッシュを使用します。これを行うには、タスクを実行するたびに、ECS_IMAGE_PULL_BEHAVIOR を prefer-cached に設定して、インスタンス自体でキャッシュされたイメージを使用するように Amazon ECS エージェントをデフォルトに設定します。

8. コンテナインスタンスを作成し、このインスタンスを実行する AWS Outposts の VPC とサブネットを指定して、AWS Outposts で使用できるインスタンスタイプを指定します。(この手順は、AWS Outposts では異なります。)

   userdata.txt ファイルは、インスタンスの起動後にそのインスタンスが一般的な自動設定タスクを実行したり、スクリプトを実行したりするために使用できるユーザーデータが含まれています。API コールのファイルの情報については、「Amazon EC2 ユーザーガイド」の「起動時に Linux インスタンスでコマンドを実行する」を参照してください。

```
aws ec2 run-instances --count 1 --image-id ami-xxxxxxxx --instance-type c5.large \
    --key-name aws-outpost-key --subnet-id subnet-xxxxxxxxxxxxxxxxx \
    --iam-instance-profile Name outpost --security-group-id sg-xxxxxx \
    --associate-public-ip-address --user-data file://userdata.txt
```

> **Note**
>
> このコマンドは、クラスターにインスタンスを追加する場合にも使用されます。クラスターにデプロイされたコンテナは、その特定の AWS Outposts に配置されます。

Amazon ECS コンテナイメージのベストプラクティス

コンテナイメージは、コンテナの構築方法に関する一連の指示です。コンテナイメージには、アプリケーションコードと、アプリケーションコードの実行に必要なすべての依存関係が保持されます。アプリケーションの依存関係には、アプリケーションコードが依存するソースコードパッケージ、インタープリタ型言語の言語ランタイム、および動的にリンクされたコードが依存するバイナリパッケージが含まれます。

コンテナイメージを設計および構築する際には、次のガイドラインに従ってください。

- すべてのアプリケーションの依存関係をコンテナイメージ内に静的ファイルとして保存することで、コンテナイメージを完成させます。

  コンテナイメージ内の何かを変更する場合は、変更内容を反映した新しいコンテナイメージを構築します。
- コンテナ内で単一のアプリケーションプロセスを実行します。

  コンテナの有効期間は、アプリケーションプロセスが実行されている期間です。Amazon ECS は、クラッシュしたプロセスを置き換えると共に、置き換え後のプロセスをどこで起動するかを決定します。適切に完成されたイメージは、デプロイ全体の耐障害性を高めます。
- アプリケーションが `SIGTERM` を処理できるようにします。

  Amazon ECS は、タスクを停止する場合、まず SIGTERM シグナルをそのタスクに送信し、アプリケーションを終了してシャットダウンする必要があることを通知します。その後、Amazon ECS は `SIGKILL` のメッセージを送信します。アプリケーションが `SIGTERM` を無視した場合、Amazon ECS サービスはしばらく待ってからプロセスを終了する `SIGKILL` シグナルを送信する必要があります。

  アプリケーションが作業を完了するまでにかかる時間を特定して、アプリケーションが確実に `SIGTERM` シグナルに対応できるようにしてください。アプリケーションによるシグナル対応においては、アプリケーションがそれ以上新しい作業を取得するのをストップするとともに、進行中の

作業を完了するか、または完了までに長い時間がかかる場合は、タスクの外部のストレージに未完了の作業を保存する必要があります。

- ログを `stdout` および `stderr` に書き込むように、コンテナ化されたアプリケーションを設定します。

  ログ処理をアプリケーションコードから分離することで、ログ処理をインフラストラクチャレベルでより柔軟に調整できるようになります。その一例として、ロギングシステムの変更があります。サービスを変更したり、新しいコンテナイメージを構築してデプロイする代わりに、設定の調整により対応できます。

- タグを使用してコンテナイメージをバージョニングします。

  コンテナイメージはコンテナレジストリに保存されます。レジストリ内の各イメージはタグによって識別されます。`latest` というタグがあります。このタグは、git リポジトリの HEAD と同様に、アプリケーションコンテナイメージの最新バージョンに対するポインターとして機能します。`latest` タグはテストのみに使用することが推奨されます。ベストプラクティスとして、ビルドごとにコンテナイメージに一意のタグを付けます。イメージの構築に使用された git コミットの git SHA を使用してイメージにタグ付けすることをお勧めします。

  コミットごとにコンテナイメージを構築する必要はありません。ただし、特定のコードコミットを本番環境にリリースするたびに、新しいコンテナイメージを構築することをお勧めします。また、イメージに対して、イメージ内のコードの git commit に対応するタグを付けることをお勧めします。イメージに git commit のタグを付けた場合、イメージが実行しているコードのバージョンをより迅速に見つけることができます。

  また、Amazon Elastic Container Registry でイミュータブルなイメージタグをオンにすることをお勧めします。この設定では、タグがポイントするコンテナイメージを変更することはできません。代わりに、Amazon ECR は、新しいイメージを新しいタグにアップロードするよう強制します。詳細については、「Amazon ECR ユーザーガイド」の「イメージタグの変更可能性」を参照してください。

AWS Fargate で実行されるアプリケーションを構築する場合には、複数のコンテナを同一のタスク定義にデプロイするか、あるいは複数のタスク定義に個々のコンテナを別々にデプロイするかのいずれかを選択する必要があります。以下のような要件がある場合には、1 つのタスク定義にコンテナを配置することをお勧めします。

- 各コンテナが同じライフサイクルを共有している (つまり、起動と終了が同時に行われる)。

- 実行基盤となるホストが同じになるようにコンテナを実行する (つまり、あるコンテナが、localhost ポート上の別のコンテナを参照する) 必要がある。
- 各コンテナがリソースを共有している。
- コンテナでデータボリュームを共有している。

上記の要件がない場合は、複数のタスク定義でコンテナを個別にデプロイすることをお勧めします。これにより、各コンテナを個別にスケーリング、プロビジョニング、およびデプロビジョニングできます。

Amazon ECS ネットワーキングのベストプラクティス

最近のアプリケーションは通常、相互に通信する複数の分散コンポーネントから構築されています。例えば、モバイルアプリケーションやウェブアプリケーションは API エンドポイントと通信したり、その API はインターネット経由で通信する複数のマイクロサービスによって動作していたりします。

インターネットへの接続アプリケーションのベストプラクティスについては、「Amazon ECS アプリケーションをインターネットに接続する」を参照してください。

インターネットから Amazon ECS へのインバウンド接続を受信するためのベストプラクティスについては、「インターネットから Amazon ECS へのインバウンド接続を受信するためのベストプラクティス」を参照してください。

Amazon ECS を他の AWS サービスに接続するためのベストプラクティスについては、「Amazon ECS を VPC 内から AWS サービスに接続するためのベストプラクティス」を参照してください。

VPC 内のサービスに接続するためのベストプラクティスについては、「VPC で Amazon ECS サービスを接続するためのベストプラクティス」を参照してください。

AWS アカウント間および VPC 間でのサービスのネットワーキングに関するベストプラクティスについては、「AWS アカウントおよび VPC の間で Amazon ECS サービスをネットワーキングするためのベストプラクティス」を参照してください。

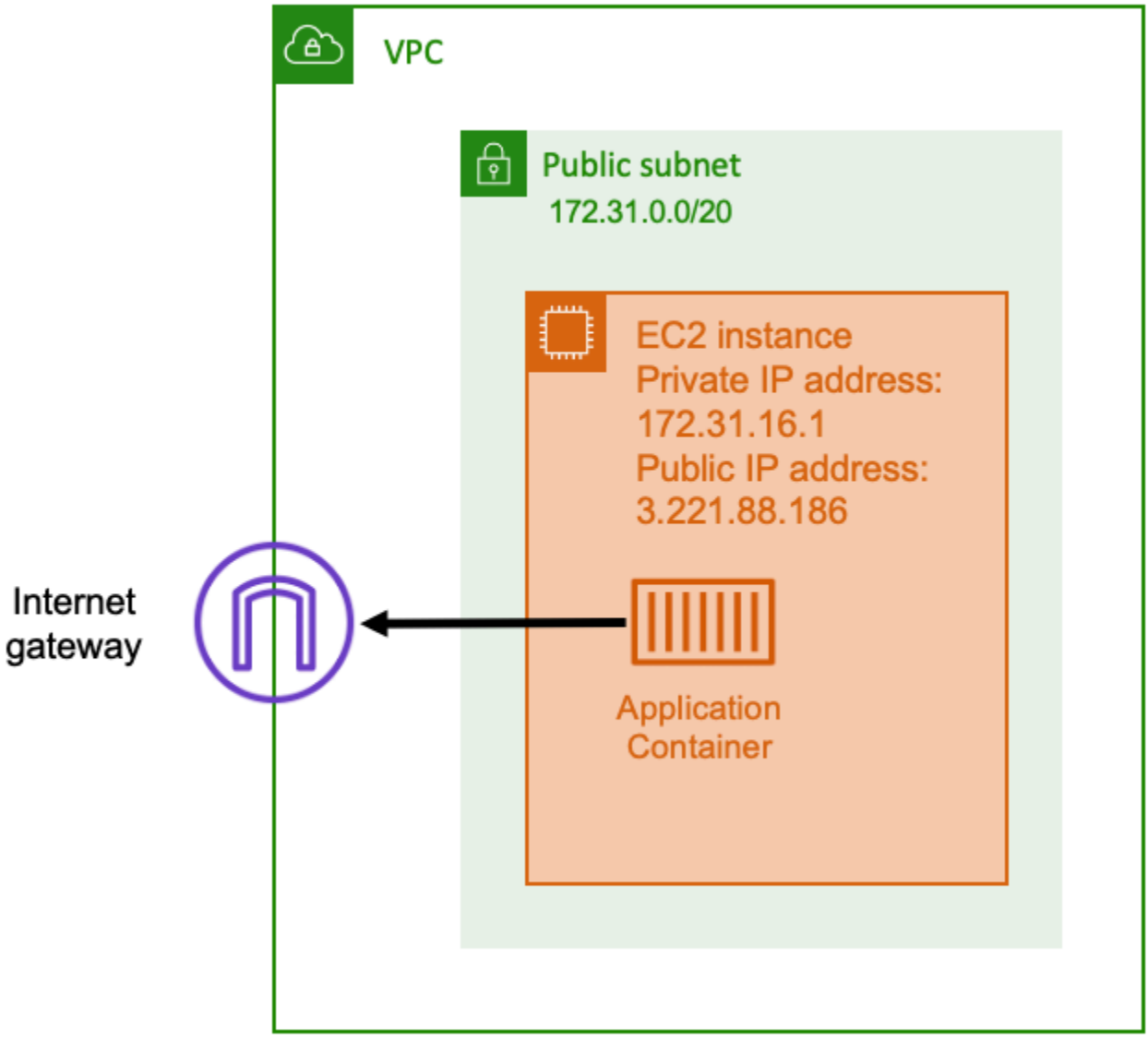
ネットワーキングの問題をトラブルシューティングするサービスに関するベストプラクティスについては、「AWSAmazon ECS ネットワーキングのトラブルシューティングのための サービス」を参照してください。

Amazon ECS アプリケーションをインターネットに接続する

ほとんどのコンテナ化されたアプリケーションには、少なくとも、インターネットへのアウトバウンドアクセスを必要とするいくつかのコンポーネントが含まれています。たとえば、モバイルアプリのバックエンドでは、プッシュ通知へのアウトバウンドアクセスが必要です。

Amazon Virtual Private Cloud には、VPC とインターネット間の通信を円滑にする主な方法が 2 つあります。

パブリックサブネットとインターネットゲートウェイ



インターネットゲートウェイへのルートがあるパブリックサブネットを使用すると、コンテナ化されたアプリケーションは、パブリックサブネットの VPC 内のホストで実行できます。コンテナを実行するホストには、パブリック IP アドレスが割り当てられます。このパブリック IP アドレスは、インターネットからルーティング可能です。詳細については、Amazon VPC ユーザーガイドの「インターネットゲートウェイ」を参照してください。

このネットワークアーキテクチャにより、アプリケーションを実行するホストとインターネット上の他のホストとの直接通信が容易になります。通信は双方向です。つまり、インターネット上の他のホストへのアウトバウンド接続を確立できるだけでなく、インターネット上の他のホストもそのホストに接続を試みる可能性があります。そのため、セキュリティグループとファイアウォールのルールには細心の注意を払う必要があります。これにより、インターネット上の他のホストは、開くべきでない接続を開くことができなくなります。

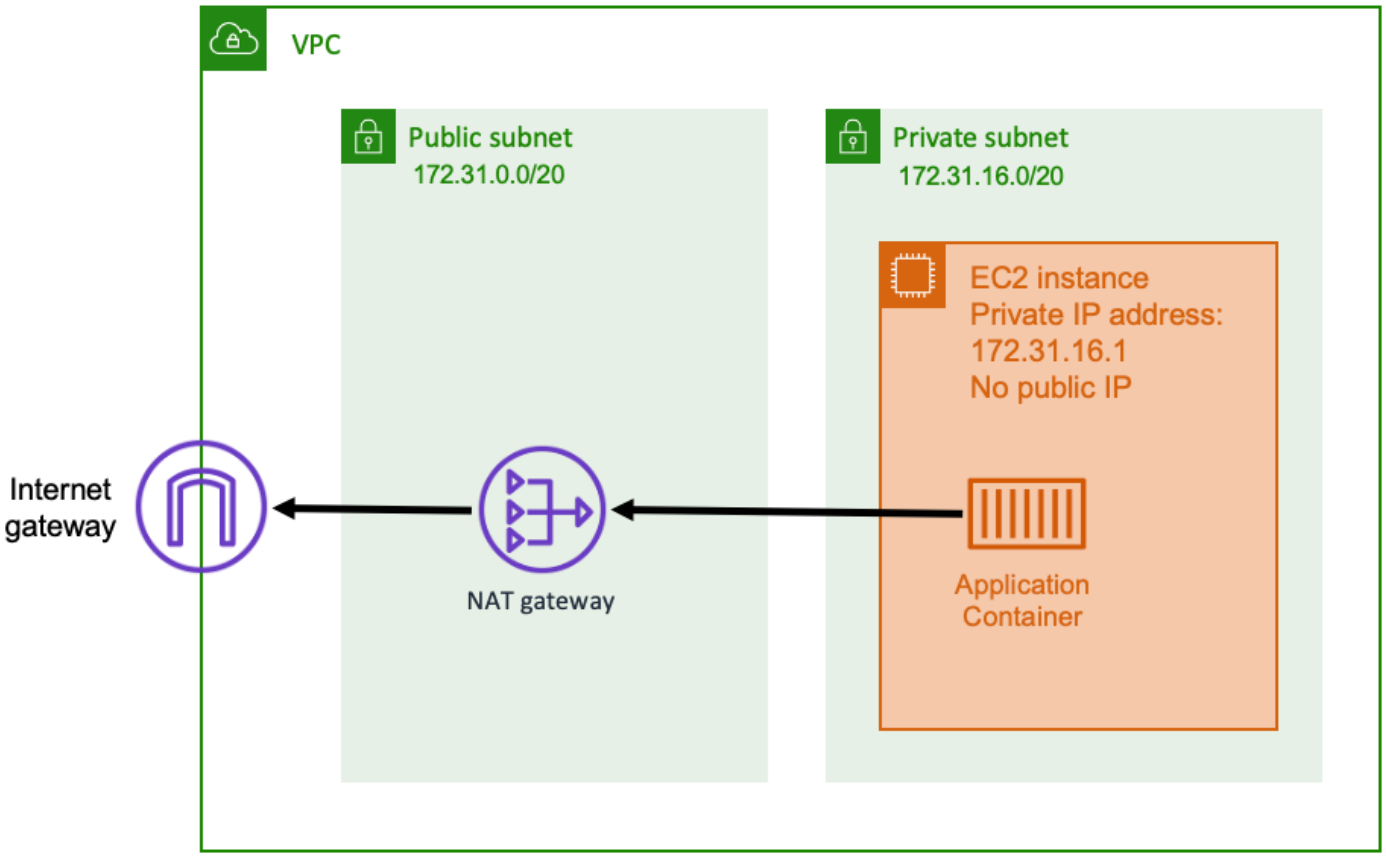
たとえば、アプリケーションが Amazon EC2 で実行されている場合は、SSH アクセス用のポート 22 が開いていないことを確認してください。そうしないと、インスタンスがインターネット上の悪意のあるボットが継続的に SSH 接続を試みる可能性があります。これらのボットはパブリック IP アドレスを総当たり的に調べます。空いている SSH ポートが見つかると、そのようなボットはパスワードをブルートフォース攻撃してインスタンスにアクセスしようとします。このため、多くの組織はパブリックサブネットの使用を制限し、すべてではないにしてもほとんどのリソースをプライベートサブネット内に置くことを好みます。

ネットワークにパブリックサブネットを使用することは、大量の帯域幅や最小のレイテンシーを必要とするパブリックアプリケーションに適しています。該当するユースケースには、ビデオストリーミングやゲームサービスが含まれます。

このネットワークアプローチは、Amazon ECS を Amazon EC2 で使用する場合と、AWS Fargate で使用する場合の両方でサポートされます。

- Amazon EC2 — パブリックサブネットで EC2 インスタンスを起動できます。Amazon ECS はこれらの EC2 インスタンスをクラスター容量として使用し、インスタンス上で実行されるすべてのコンテナは、ホストの基礎となるパブリック IP アドレスをアウトバウンドネットワークに使用できます。これは `host` および `bridge` ネットワークモードの両方に当てはまります。ただし、`awsvpc` ネットワークモードでは、タスク ENI にパブリック IP アドレスが提供されません。そのため、インターネットゲートウェイを直接使用することはできません。
- Fargate — Amazon ECS サービスを作成するときは、サービスのネットワーク設定にパブリックサブネットを指定し、[パブリック IP アドレスの割り当て] オプションを使用してください。各 Fargate タスクはパブリックサブネットでネットワーク化されており、インターネットと直接通信するための独自のパブリック IP アドレスを持っています。

プライベートサブネットと NAT ゲートウェイ



プライベートサブネットと NAT ゲートウェイを使用すると、プライベートサブネット内のホストでコンテナ化されたアプリケーションを実行できます。そのため、このホストのプライベート IP アドレスは VPC 内ではルーティング可能ですが、インターネットからはルーティングできません。つまり、VPC 内の他のホストはプライベート IP アドレスを使用してホストに接続できますが、インターネット上の他のホストはそのホストとのインバウンド通信を行うことができません。

プライベートサブネットでは、ネットワークアドレス変換 (NAT) ゲートウェイを使用して、プライベートサブネット内のホストがインターネットに接続できるようにします。インターネット上のホストは、パブリックサブネット内の NAT ゲートウェイのパブリック IP アドレスから送信されているように見えるインバウンド接続を受信します。NAT ゲートウェイは、インターネットとプライベートサブネットの間のブリッジとして機能します。この構成では、VPC がインターネット上の攻撃者による直接アクセスから保護されるため、多くの場合セキュリティ上望ましいものです。詳細については、「Amazon VPC ユーザーガイド」の「NAT ゲートウェイ」を参照してください。

このプライベートネットワークアプローチは、外部からの直接アクセスからコンテナを保護するシナリオに適しています。該当するシナリオには、支払い処理システムや、ユーザーデータとパスワード

を格納するコンテナなどがあります。アカウントで NAT ゲートウェイを作成して使用するには料金がかかります。NAT ゲートウェイの時間単位の使用料金とデータ処理料金も適用されます。冗長性を確保するために、アベイラビリティーゾーンごとに NAT ゲートウェイが必要です。こうすることで、1 つのアベイラビリティーゾーンの可用性が失われても、アウトバウンド接続が損なわれなくなります。このため、ワークロードが少ない場合は、プライベートサブネットと NAT ゲートウェイを使用する方が費用対効果が高い可能性があります。

このネットワークアプローチは、Amazon ECS を Amazon EC2 で使用する場合と、AWS Fargate で使用する場合の両方でサポートされます。

- Amazon EC2 — プライベートサブネットで EC2 インスタンスを起動できます。これらの EC2 ホストで実行されるコンテナは、基盤となるホストのネットワークを使用し、アウトバウンドリクエストは NAT ゲートウェイを経由します。
- Fargate — Amazon ECS サービスを作成するときは、サービスのネットワーク設定にプライベートサブネットを指定してください。[パブリック IP アドレスの割り当て] オプションは使用しないでください。各 Fargate タスクはプライベートサブネットでホストされます。そのアウトバウンドトラフィックは、そのプライベートサブネットに関連付けた NAT ゲートウェイを経由してルーティングされます。

インターネットから Amazon ECS へのインバウンド接続を受信するためのベストプラクティス

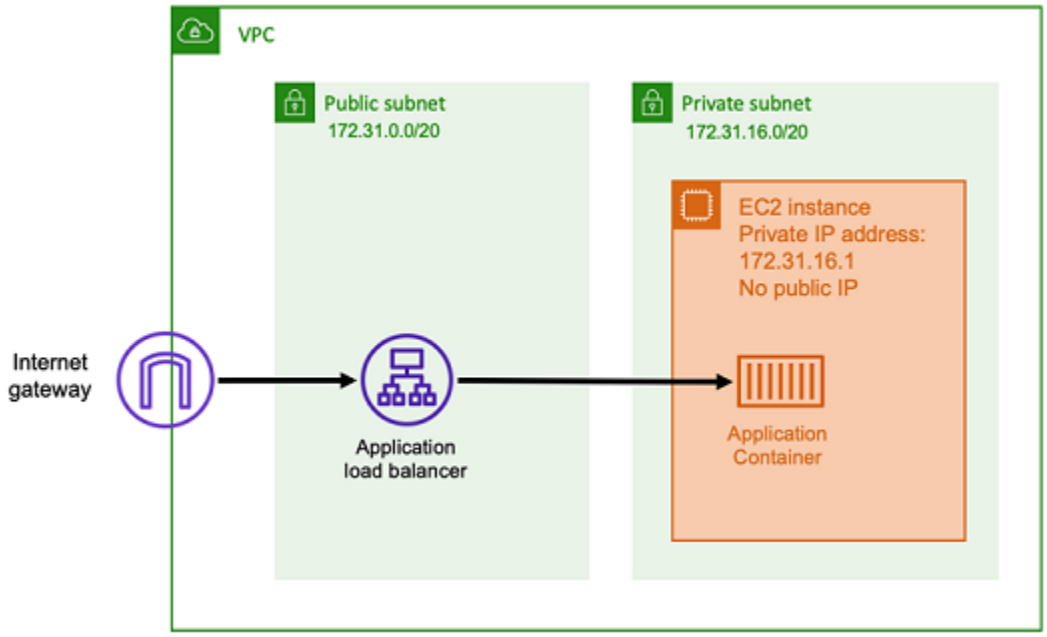
パブリックサービスを運営している場合は、インターネットからのインバウンドトラフィックを受け入れる必要があります。たとえば、公開 Web サイトはブラウザからのインバウンド HTTP リクエストを受け入れる必要があります。このような場合、インターネット上の他のホストもアプリケーションのホストへのインバウンド接続を開始する必要があります。

この問題に対する 1 つの方法は、パブリック IP アドレスを持つパブリックサブネット内のホスト上でコンテナを起動することです。ただし、大規模なアプリケーションにこれを行うことは推奨しません。このような場合は、インターネットとアプリケーションの間にスケーラブルな入力層を設けるのがより適切なアプローチです。このアプローチでは、このセクションに記載されている AWS サービスのいずれかを入力として使用できます。

Application Load Balancer

Application Load Balancer はアプリケーション層で機能します。これは、開放型システム間相互接続 (OSI) モデルの第 7 層です。これにより、Application Load Balancer はパブリック HTTP サービ

スに適しています。ウェブサイトまたは HTTP REST API を使用している場合は、Application Load Balancer がこのワークロードに適したロードバランサーです。詳細については、「Application Load Balancers ユーザーガイド」の「Application Load Balancer とは」を参照してください。



このアーキテクチャでは、パブリックサブネットに Application Load Balancer を作成します。これにより、パブリック IP アドレスが割り当てられ、インターネットからのインバウンド接続を受信できるようになります。Application Load Balancer は、インバウンド接続、より具体的には HTTP リクエストを受信すると、プライベート IP アドレスを使用してアプリケーションへの接続を開きます。次に、リクエストを内部接続経由で転送します。

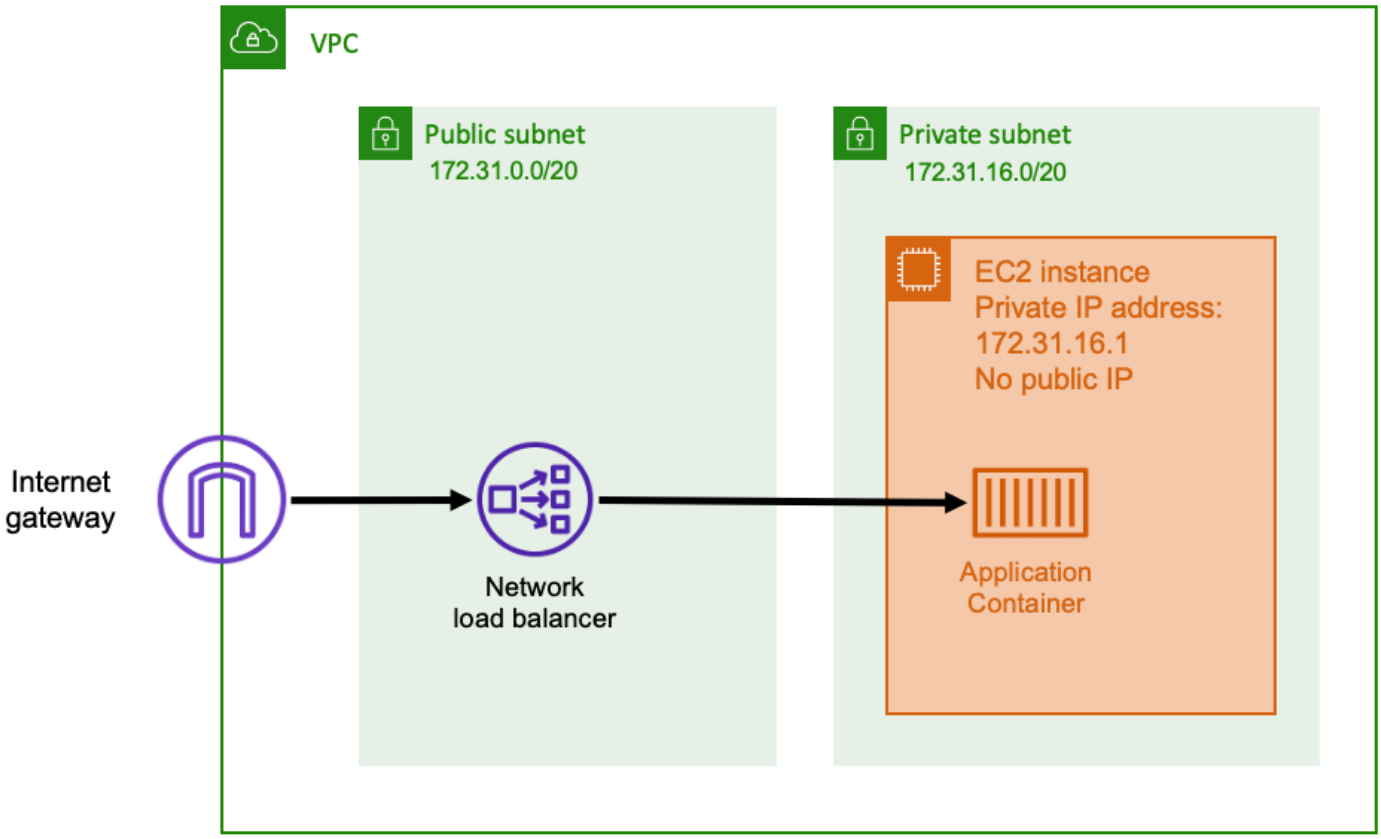
Application Load Balancer には以下の利点があります。

- SSL/TLS ターミネーション — Application Load Balancer は、クライアントとの通信用の安全な HTTPS 通信と証明書を維持できます。必要に応じて SSL 接続をロードバランサーレベルで終了できるため、独自のアプリケーションで証明書を処理する必要がありません。
- 高度なルーティング — Application Load Balancer は複数の DNS ホスト名を持つことができます。また、ホスト名やリクエストのパスなどのメトリックに基づいて、受信した HTTP リクエストをさまざまな宛先に送信する高度なルーティング機能も備えています。つまり、単一の Application Load Balancer をさまざまな内部サービス、さらには REST API のさまざまなパスにあるマイクロサービスの入力として使用できます。
- gRPC サポートとウェブソケット — Application Load Balancer は HTTP 以外も処理できます。また、HTTP/2 サポートにより、gRPC およびウェブソケットベースのサービスの負荷分散も可能です。

- セキュリティ — Application Load Balancer は、悪意のあるトラフィックからアプリケーションを保護するのに役立ちます。HTTP 同期解除対策などの機能が含まれており、AWS Web Application Firewall (AWS WAF) と統合されています。AWS WAF は、SQL インジェクションやクロスサイトスクリプティングなどの攻撃パターンを含む可能性のある悪意のあるトラフィックをさらに効果的に除外できます。

Network Load Balancer

Network Load Balancer は、開放型システム間相互接続 (OSI) モデルの第 4 層で機能します。HTTP 以外のプロトコルや、エンドツーエンドの暗号化が必要なシナリオに適していますが、Application Load Balancer と同じ HTTP 固有の機能はありません。そのため、Network Load Balancer は HTTP を使用しないアプリケーションに最適です。詳細については、「Network Load Balancers ユーザーガイド」の「Network Load Balancer とは」を参照してください。



Network Load Balancer を入力として使用すると、Application Load Balancer と同様に機能します。これは、パブリックサブネットで作成されており、インターネット上でアクセスできるパブリック IP アドレスがあるためです。次に、Network Load Balancer は、コンテナを実行しているホストの

プライベート IP アドレスへの接続を開き、パブリック側からプライベート側にパケットを送信します。

Network Load Balancer の機能

Network Load Balancer はネットワークスタックの下位レベルで動作するため、機能セットは Application Load Balancer と同じではありません。ただし、以下の重要な機能を備えています。

- エンドツーエンドの暗号化 — Network Load Balancer は OSI モデルの第 4 層で動作するため、パケットの内容を読み取ることはありません。これにより、エンドツーエンドの暗号化を必要とする負荷分散通信に適しています。
- TLS 暗号化 — エンドツーエンドの暗号化に加えて、Network Load Balancer は TLS 接続を終了することもできます。これにより、バックエンドアプリケーションが独自の TLS を実装する必要がなくなります。
- UDP サポート — Network Load Balancer は OSI モデルの第 4 層で動作するため、HTTP 以外のワークロードや TCP 以外のプロトコルに適しています。

接続を終了する

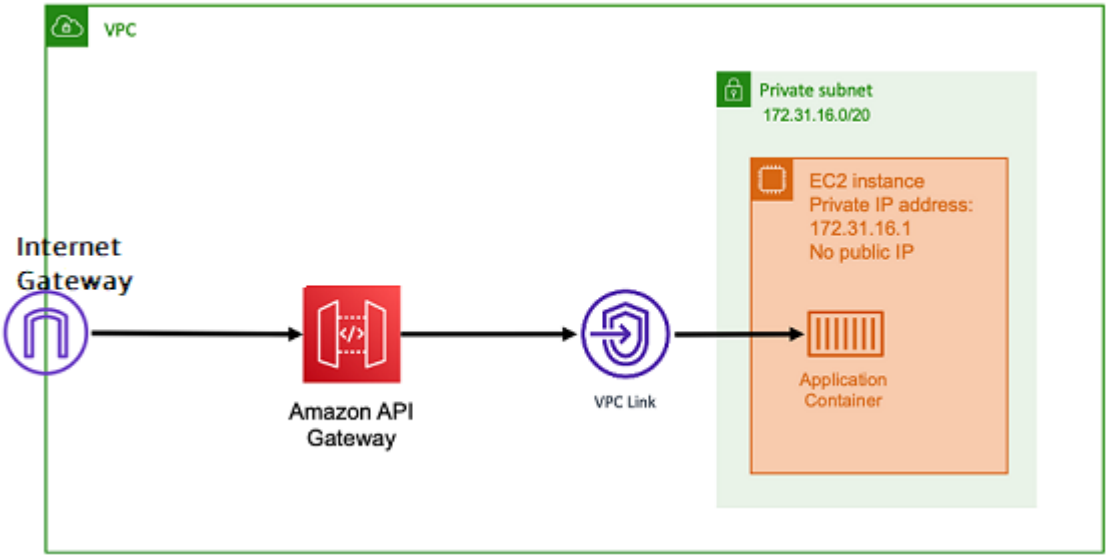
Network Load Balancer は OSI モデルの上位層ではアプリケーションプロトコルを監視しないため、それらのプロトコルのクライアントにクロージャメッセージを送信することはできません。Application Load Balancer とは異なり、これらの接続はアプリケーションによって閉じられる必要があります。または、タスクが停止または置換されたときに、第 4 レイヤーの接続を閉じるように Network Load Balancer を設定することもできます。Network Load Balancer のドキュメントで、Network Load Balancer のターゲットグループの接続終了設定を参照してください。

Network Load Balancer に第 4 層で接続を終了させると、クライアントがそれを処理しない場合、クライアントが望ましくないエラーメッセージを表示する恐れがあります。推奨されるクライアント設定の詳細については、「こちらの」 Builders Library を参照してください。

接続を閉じる方法はアプリケーションによって異なりますが、1 つの方法は、Network Load Balancer のターゲット登録解除の遅延時間をクライアントの接続タイムアウトよりも長くすることです。クライアントは最初にタイムアウトし、Network Load Balancer を介して次のタスクに正常に再接続し、同時に古いタスクではすべてのクライアントがゆっくりとドレインされます。Network Load Balancer のターゲット登録解除を遅延させる方法の詳細については、Network Load Balancer のドキュメントを参照してください。

Amazon API Gateway HTTP API

Amazon API Gateway は、リクエスト量が突然急増する、またはリクエスト量が少ない HTTP アプリケーションに適しています。詳細については、「API Gateway デベロッパーガイド」の「Amazon API Gateway とは」を参照してください。



Application Load Balancer および Network Load Balancer の料金モデルには、ロードバランサーがインバウンド接続を常時受け付けられるようにするための時間単位の料金が含まれています。これとは対照的に、API Gateway ではリクエストごとに個別に請求されます。これには、リクエストを受信しない限り課金されないという効果があります。トラフィック負荷が高い場合、Application Load Balancer または Network Load Balancer は、API Gateway よりも安いリクエストあたりの料金で大量のリクエストを処理できます。ただし、全体的にリクエスト数が少ない場合やトラフィックが少ない期間がある場合は、使用率が低いロードバランサーを維持するために時間単位の料金を支払うよりも、API Gateway を使用する場合の累積料金の方が費用対効果が高くなるでしょう。API Gateway は API レスポンスをキャッシュすることもできるため、バックエンドのリクエスト率が軽減される可能性があります。

API Gateway は、AWS マネージドサービスがプライベート IP アドレスを使用して VPC のプライベートサブネット内のホストに接続できるようにする VPC リンクを使用する機能です。Amazon ECS Service Discovery によって管理されている AWS Cloud Map サービス検出レコードを調べることで、これらのプライベート IP アドレスを検出できます。

API Gateway は、次の機能をサポートしています。

- API Gateway の動作はロードバランサーに似ていますが、API 管理に固有の追加機能があります。

- API Gateway は、クライアントの承認、使用レベル、リクエスト/レスポンスの変更に関する追加機能を備えています。詳細については、「Amazon API Gateway の機能」を参照してください。
- API Gateway は、エッジ、リージョン、プライベート API ゲートウェイのエンドポイントをサポートできます。エッジエンドポイントは、マネージド CloudFront ディストリビューションを通じて利用できます。リージョナルエンドポイントとプライベートエンドポイントはどちらもリージョンに対してローカルです。
- SSL/TLS ターミネーション
- 異なる HTTP パスを別々のバックエンドマイクロサービスにルーティングする

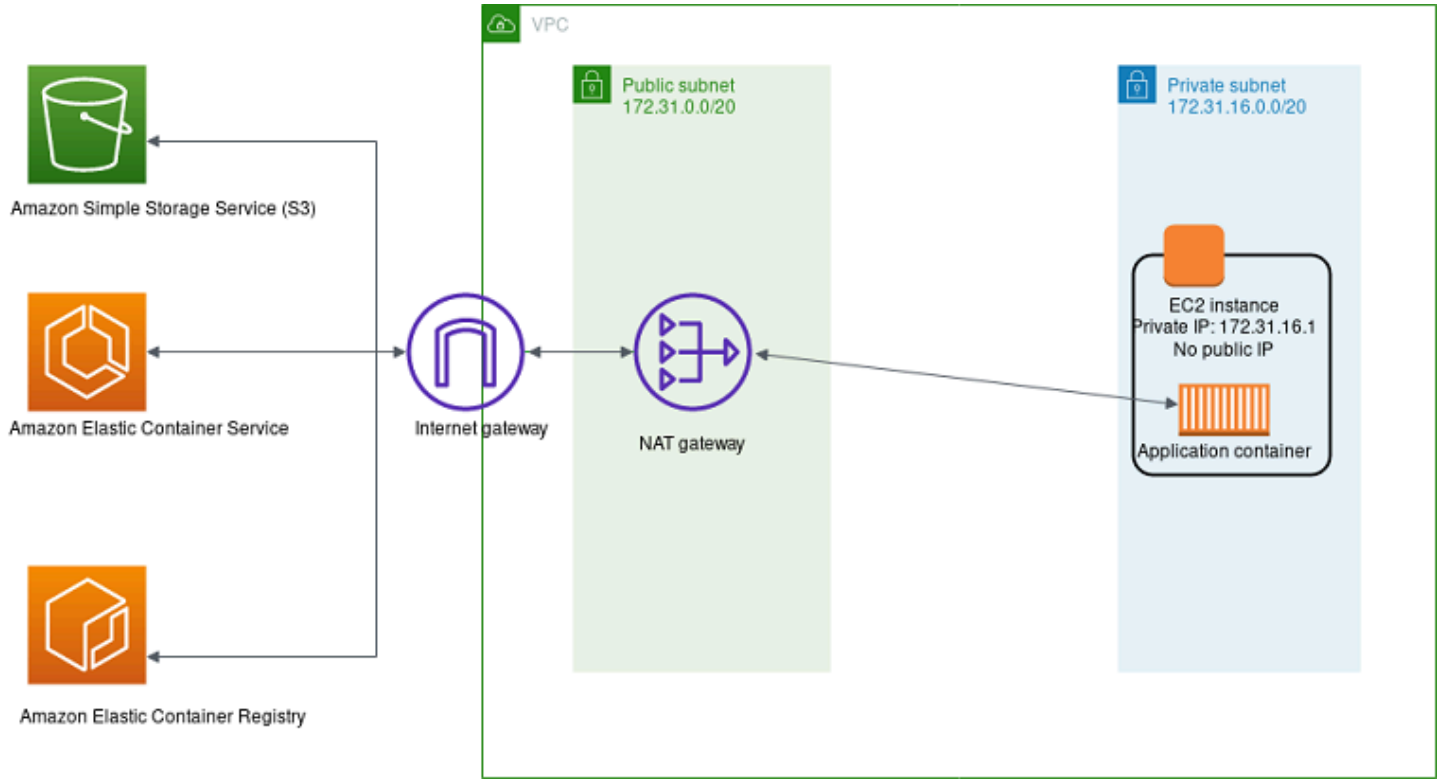
API Gateway は、前述の機能に加えて、API を不正使用から保護するために使用できるカスタム Lambda オーソライザーの使用もサポートしています。詳細については、「フィールドノート:Amazon ECS と Amazon API Gateway を使用したサーバーレスコンテナベースの API」を参照してください。

Amazon ECS を VPC 内から AWS サービスに接続するためのベストプラクティス

Amazon ECS が正しく機能するためには、各ホストで実行される Amazon ECS コンテナエージェントは Amazon ECS コントロールプレーンと通信する必要があります。コンテナイメージを Amazon ECR に保存している場合、Amazon EC2 ホストは Amazon ECR サービスエンドポイントと、およびイメージレイヤーが保存されている Amazon S3 と通信する必要があります。DynamoDB に保存されているデータの保持など、コンテナ化されたアプリケーションに他の AWS サービスを使用する場合は、必要なネットワーキングのサポートもこのサービスにあることを再確認してください。

NAT ゲートウェイ

NAT ゲートウェイの使用は、Amazon ECS タスクが他の AWS サービスにアクセスできるようになる最も簡単な方法です。この方法の詳細については、「プライベートサブネットと NAT ゲートウェイ」を参照してください。



この方法の利用における欠点は次のとおりです。

- NAT ゲートウェイが通信できる宛先を制限することはできません。また、VPC からのすべてのアウトバウンド通信を中断することなく、バックエンド層が通信できる宛先を制限することはできません。
- NAT ゲートウェイは、通過するデータの GB ごとに課金されます。NAT ゲートウェイを次のいずれかの操作に使用すると、帯域幅の GB ごとに課金されます。
  - Amazon S3 からの大きなファイルのダウンロード
  - DynamoDB に対する大量のデータベースクエリの実行
  - Amazon ECR からのイメージのプル

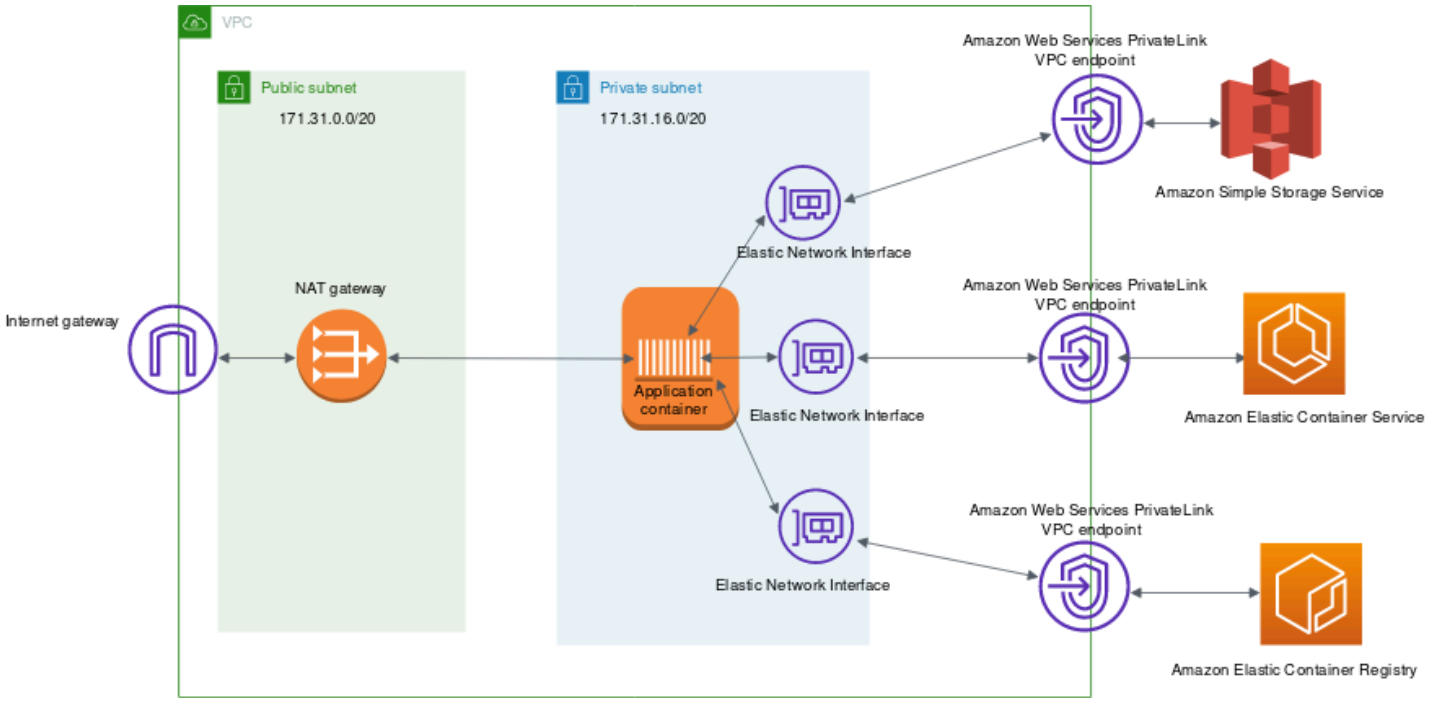
  さらに、NAT ゲートウェイは 5 Gbps の帯域幅をサポートしており、45 Gbps まで自動的にスケールアップします。1 つの NAT ゲートウェイを介してルーティングする場合、非常に大きな帯域幅の接続を必要とするアプリケーションでは、ネットワーク上の制約が発生する可能性があります。回避策として、ワークロードを複数のサブネットに分割し、各サブネットに独自の NAT ゲートウェイを与えることができます。

AWS PrivateLink

AWS PrivateLink は、トラフィックをパブリックインターネットに公開することのない、VPC、AWS サービス、オンプレミスネットワークの間のプライベート接続を提供します。

VPC エンドポイントにより、VPC とサポートされている AWS サービスおよび VPC エンドポイントサービスとの間のプライベート接続が可能になります。VPC と他のサービス間のトラフィックは、Amazon ネットワークを離れることはありません。VPC エンドポイントは、インターネットゲートウェイ、仮想プライベートゲートウェイ、NAT デバイス、VPN 接続、または Direct Connect 接続を必要としません。VPC の Amazon EC2 インスタンスでは、サービスのリソースと通信するのにパブリック IP アドレスを必要としません。

次の図は、インターネットゲートウェイではなく VPC エンドポイントを使用している場合に、AWS サービスへの通信がどのように機能するかを示しています。AWS PrivateLink はサブネット内で Elastic Network Interface (ENI) をプロビジョニングし、ENI 経由でサービスホスト名への通信を宛先の AWS サービスに直接送信するのに、VPC ルーティングルールが使用されています。このトラフィックは、NAT ゲートウェイまたはインターネットゲートウェイを使用する必要がなくなっています。



以下は、Amazon ECS サービスで使用される一般的な VPC エンドポイントの一部です。

- Amazon S3 のゲートウェイエンドポイント
- DynamoDB VPC エンドポイント

- Amazon ECS VPC エンドポイント
- Amazon ECR VPC エンドポイント

他の多くの AWS サービスは VPC エンドポイントをサポートしています。いずれかの AWS サービスを大量に使用する場合は、そのサービスの特定のドキュメントと、そのトラフィックにおいて VPC エンドポイントを作成する方法を調べる必要があります。

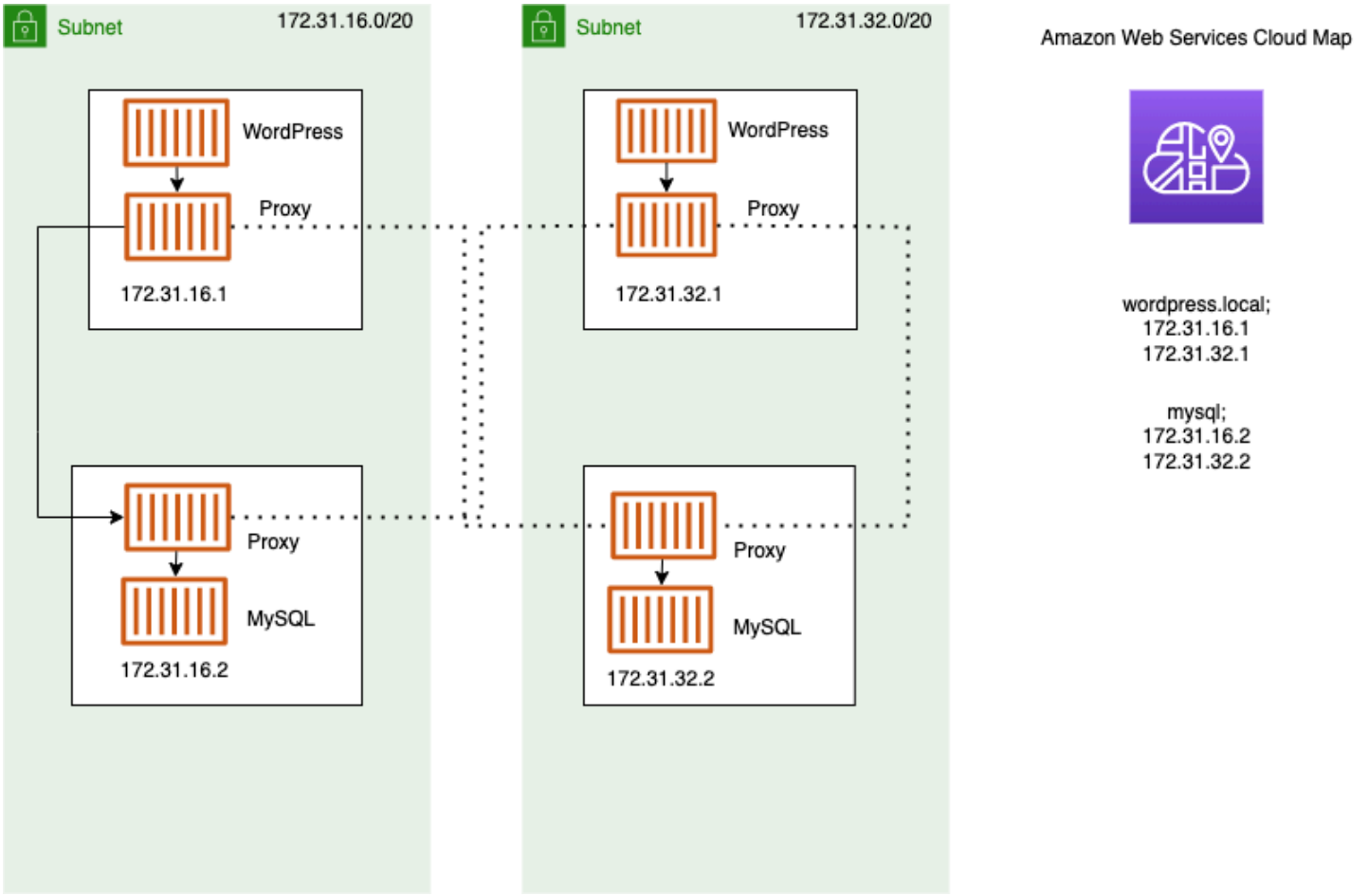
VPC で Amazon ECS サービスを接続するためのベストプラクティス

VPC で Amazon ECS タスクを使用すると、モノリシックアプリケーションを、安全な環境で個別にデプロイおよびスケーリングできる別々の部分に分割できます。このアーキテクチャは、サービス指向アーキテクチャ (SOA) またはマイクロサービスといいます。ただし、VPC の内外両方で、これらのすべての部分が相互に通信できるようにするのが難しい場合があります。通信を促進する方法は複数ありますが、すべてにさまざまな利点と欠点があります。

Service Connect の使用

サービス検出、接続、トラフィックモニタリング用の Amazon ECS 設定を提供する Service Connect をお勧めします。Service Connect を使用すると、アプリケーションは短縮名と標準ポートを使用して、同じクラスターや他のクラスター (同じリージョンの VPC 間を含む) 内のサービスに接続できます。詳細については、「Amazon ECS Service Connect」を参照してください。

Service Connect を使用すると、Amazon ECS がサービス検出のすべての部分を管理します。つまり、検出可能な名前の作成、タスクの開始と終了時に各タスクのエントリを動的に管理すること、名前を検出するように設定された各タスクでのエージェントの実行などです。アプリケーションは DNS 名の標準機能を使用して接続することで名前を検索できます。アプリケーションがすでにこれを実行している場合、Service Connect を使用する際に、アプリケーションを変更する必要はありません。



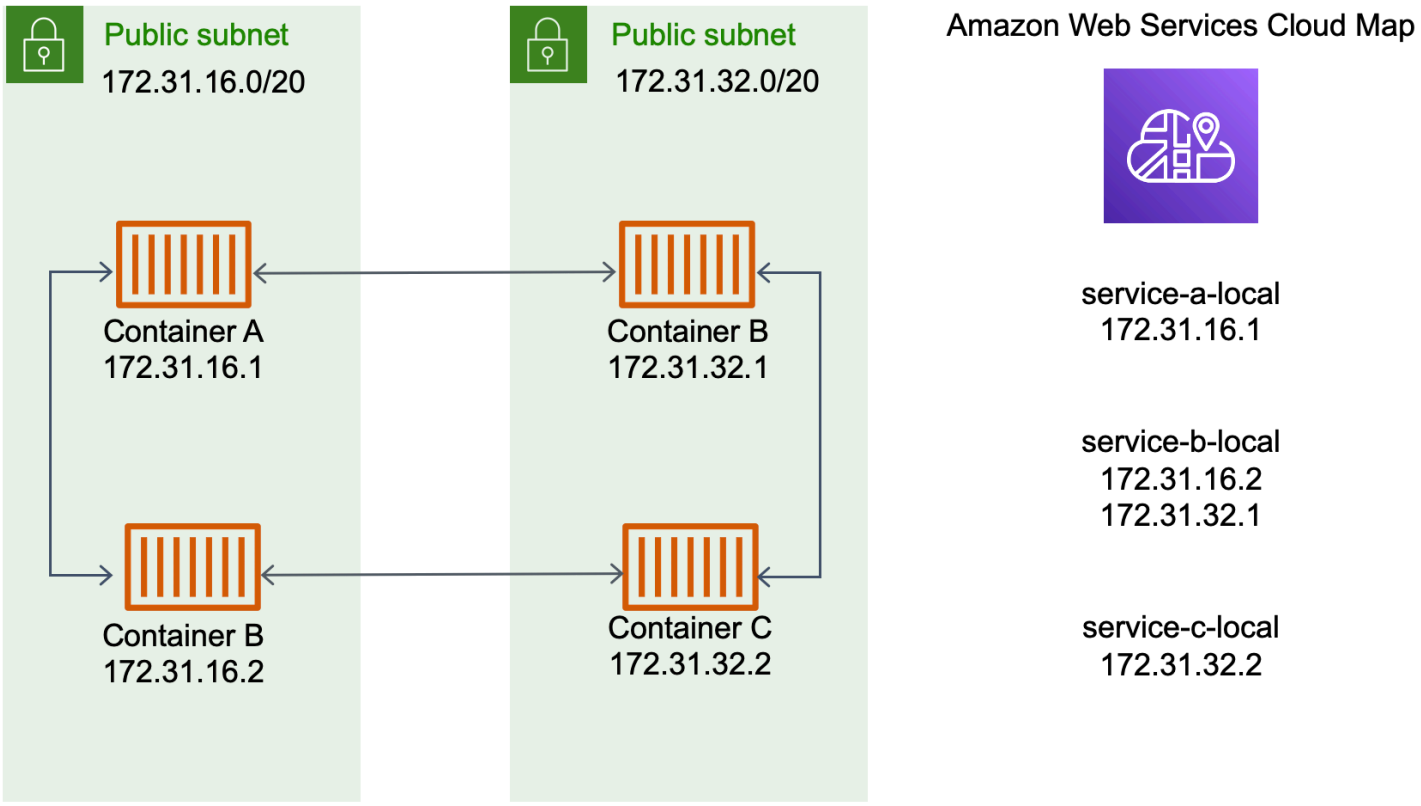
変更はデプロイ中にのみ発生します

各サービスとタスク定義内で完全な設定を行います。Amazon ECS は、デプロイ内のすべてのタスクが同じように動作するように、サービスデプロイごとにこの設定の変更を管理します。たとえば、サービス検出の DNS でよくある問題は、移行の制御です。新しい IP アドレスを指すように DNS 名を変更すると、すべてのクライアントが新しいサービスの使用を開始するまでに最大 TTL 時間がかかることがあります。Service Connect では、クライアントデプロイメントがクライアントタスクを置き換えて設定を更新します。他のデプロイと同様に、Service Connect の変更に影響するように、デプロイサーキットブレーカーやその他のデプロイ設定を設定できます。

サービス検出の使用

サービス間の通信のもう 1 つのアプローチは、サービス検出を使用する直接的な通信です。このアプローチでは、Amazon ECS との AWS Cloud Map サービス検出の統合を使用できます。Amazon ECS はサービス検出を使用して、起動されたタスクのリストを AWS Cloud Map に同期します。このホスト名は特定のサービスの 1 つ以上のタスクの内部 IP アドレスに解決される DNS ホスト名を保持します。Amazon VPC の他のサービスは、この DNS ホスト名を使用して、内部 IP アドレスを

使用してトラフィックを別のコンテナに直接送信できます。詳細については、「サービス検出」を参照してください。



前出の図では、サービスが 3 つあります。`service-a-local` はコンテナが 1 つあり、コンテナが 2 つある `service-b-local` と通信します。`service-b-local` は、コンテナが 1 つある `service-c-local` とも通信する必要があります。これら 3 つのサービスすべての各コンテナは、AWS Cloud Map の内部 DNS 名を使用して、通信する必要があるダウンストリームサービスからコンテナの内部 IP アドレスを見つけることができます。

このサービス間通信のアプローチでは、レイテンシが低くなります。一見すると、コンテナ間に余分なコンポーネントがないためシンプルでもあります。トラフィックは 1 つのコンテナから別のコンテナに直接移動します。

この方法は、各タスクに固有の IP アドレスが割り当てられる `awsvpc` ネットワークモードを使用する場合に適しています。ほとんどのソフトウェアは、IP アドレスに直接変換される DNS `A` レコードの使用のみをサポートしています。`awsvpc` ネットワークモードを使用する場合、各タスクの IP アドレスは `A` レコードになります。ただし、`bridge` ネットワークモードを使用している場合は、複数のコンテナが同じ IP アドレスを共有している可能性があります。さらに、動的ポートマッピングでは、その 1 つの IP アドレスのポート番号がコンテナにランダムに割り当てられます。この時点では、`A` レコードだけではサービス検出には不十分です。`SRV` レコードも使用する必要があります。

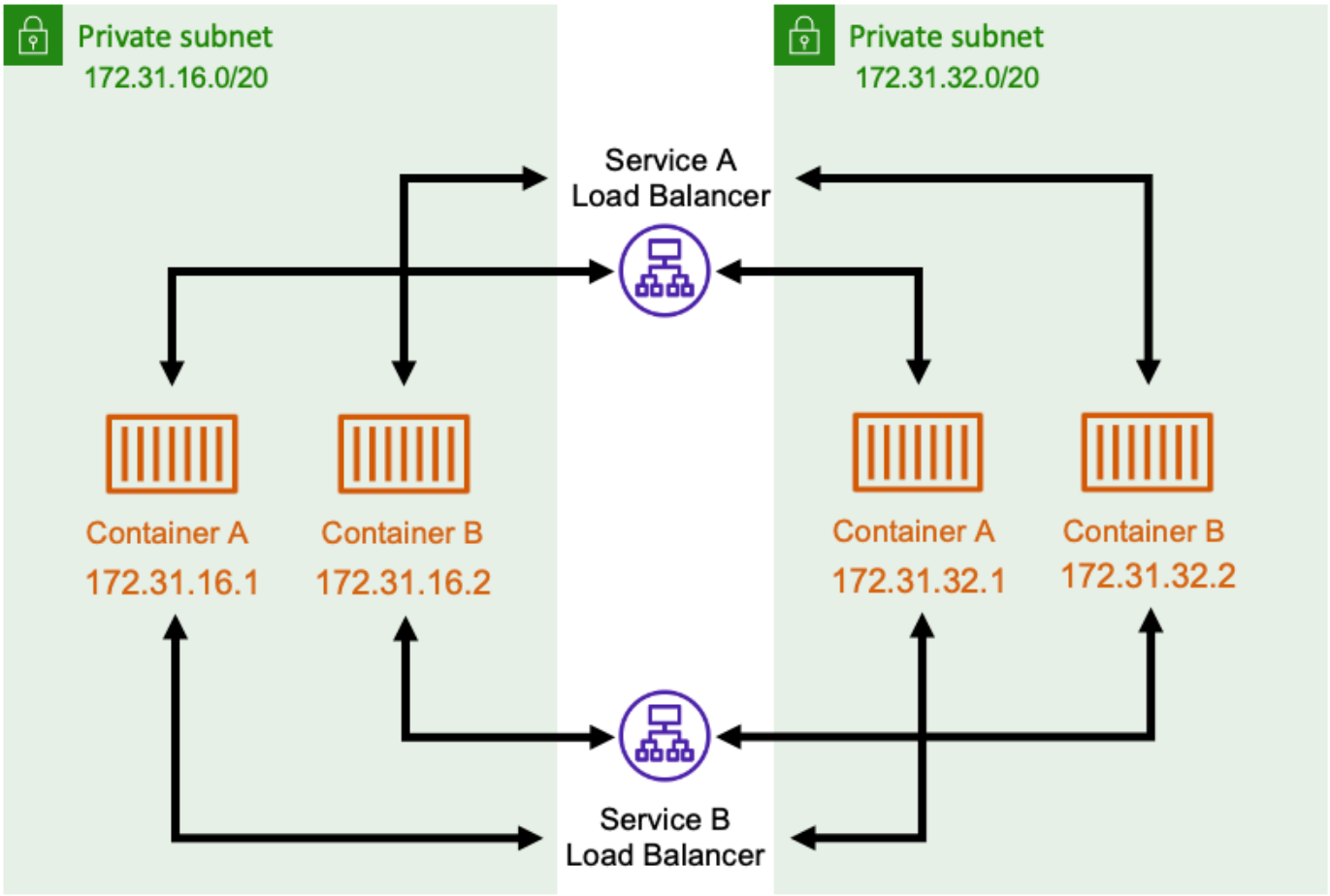
このタイプのレコードは IP アドレスとポート番号の両方を記録できますが、アプリケーションを適切に設定する必要があります。使用するビルド済みアプリケーションの中には、SRV レコードをサポートしていないものもあります。

awsvpc ネットワークモードのもう 1 つの利点は、サービスごとに固有のセキュリティグループがあることです。このセキュリティ グループを設定して、そのサービスと通信する必要がある特定のアップストリームサービスからの受信接続のみを許可することができます。

サービス検出を使用するサービス間直接的な通信の主な欠点は、再試行や接続障害に対処するためのロジックを追加する必要があることです。DNS レコードには、キャッシュされる時間を制御する有効期限 (TTL) があります。DNS レコードが更新されてキャッシュが期限切れになり、アプリケーションが最新バージョンの DNS レコードを取得できるようになるまでには、ある程度の時間がかかります。そのため、アプリケーションが DNS レコードを解決して、もう存在しない別のコンテナを指すようになってしまう可能性があります。アプリケーションには再試行を処理し、不正なバックエンドを無視するロジックが必要です。

内部ロードバランサーの使用

サービス間通信のもう 1 つの方法は、内部ロードバランサーを使用することです。内部ロードバランサーは完全に VPC 内に存在し、VPC 内のサービスにのみアクセスできます。



ロードバランサーは、冗長リソースを各サブネットにデプロイすることで、高可用性を維持します。serviceA のコンテナが serviceB のコンテナと通信する必要がある場合、ロードバランサーへの接続が開きます。次に、ロードバランサーは service B からコンテナへの接続を開きます。ロードバランサーは、各サービス間のすべての接続を管理するための一元的な場所として機能します。

serviceB のコンテナが停止すると、ロードバランサーはそのコンテナをプールから削除できます。また、ロードバランサーはそのプール内の各ダウンストリームターゲットに対してヘルスチェックを行い、再び正常になるまでプールから不正なターゲットを自動的に削除できます。アプリケーションは、ダウンストリームコンテナの数を認識する必要がなくなりました。ロードバランサーへの接続を開くだけです。

この方法は、すべてのネットワークモードで有益です。ロードバランサーは、awsvpc ネットワークモードを使用する場合のタスク IP アドレスと、bridge ネットワークモードを使用する場合の IP アドレスとポートのより高度な組み合わせを追跡できます。また、実際に複数のコンテナが異なるポー

トにおいて同じ Amazon EC2 インスタンスでホストされている場合でも、すべての IP アドレスとポートの組み合わせにトラフィックを均等に分散します。

この方法の欠点の 1 つはコストです。可用性を高めるために、ロードバランサーには各アベイラビリティーゾーンにリソースが必要です。この結果、ロードバランサーとロードバランサーを通過するトラフィック量に対して支払うオーバーヘッドにより、追加コストが追加されます。

ただし、複数のサービスでロードバランサーを共有することで、オーバーヘッドコストを削減できます。これは、Application Load Balancer を使用する REST サービスに特に適しています。ユーザーは、異なるサービスにトラフィックをルーティングするパスベースのルーティングルールを作成できます。例えば、`/api/user/*` は `user` サービスの一部であるコンテナにルーティングし、`/api/order/*` は関連付けられた `order` サービスにルーティングできます。この方法では、1 つの Application Load Balancer に対してのみ料金が発生し、API の URL が一貫した 1 つになります。ただし、トラフィックをバックエンドのさまざまなマイクロサービスに分割することはできます。

AWS アカウントおよび VPC の間で Amazon ECS サービスをネットワーキングするためのベストプラクティス

複数のチームや部門を有する組織に属している場合は、おそらく共有 AWS アカウント内の個別の VPC または複数の個別の AWS アカウントに関連付けられている VPC に、サービスを個別にデプロイするでしょう。どの方法でサービスをデプロイしても、VPC 間でトラフィックをルーティングできるようにネットワークコンポーネントを補完することをお勧めします。そのためには、複数の AWS サービスを使用して、既存のネットワークコンポーネントを補完することができます。

- AWS Transit Gateway – このネットワーキングサービスを最初に検討する必要があります。このサービスは、Amazon VPC、AWS アカウントおよびオンプレミスネットワーク間の接続をルーティングするための中央ハブとして機能します。詳細については、「Amazon VPC Transit Gateway ガイド」の「Transit Gateway とは」を参照してください。
- Amazon VPC および VPN サポート – このサービスを使用して、オンプレミスネットワークを VPC に接続するためのサイト間 VPN 接続を作成できます。詳細については、AWS Site-to-Site VPN ユーザーガイド の「What is AWS Site-to-Site VPN? (とは ?)」 を参照してください。
- Amazon VPC – 同じアカウント内または複数のアカウント間で複数の VPC を接続できる、Amazon VPC ピアリングを使用できます。詳細については、Amazon VPC Peering Guideの「VPC ピア機能とは」を参照してください。
- 共有 VPC – 複数の AWS アカウントで VPC と VPC サブネットを使用できます。詳細については、「Amazon VPC ユーザーガイド」の「共有 VPC の使用」を参照してください。

AWSAmazon ECS ネットワーキングのトラブルシューティングのためのサービス

以下のサービスと機能は、ネットワークとサービスの設定に関するインサイトを得るのに役立ちます。この情報を使用することで、ネットワーキング問題のトラブルシューティングを行い、サービスをより適切に最適化できます。

CloudWatch コンテナインサイト

CloudWatch Container Insights は、コンテナ化されたアプリケーションとマイクロサービスのメトリクスとログを収集、集約、要約します。メトリクスには、CPU、メモリ、ディスク、ネットワークなどのリソースの使用率が含まれています。メトリクスは CloudWatch 自動ダッシュボードで使用できます。詳細については、「Amazon CloudWatch ユーザーガイド」の「Amazon ECS での Container Insights のセットアップ」を参照してください。

AWS X-Ray

AWS X-Ray は、アプリケーションが行うネットワークリクエストに関する情報を収集するのに使用できるトレースサービスです。SDK を使用することで、アプリケーションをインストルメント化し、サービス間、およびサービスと AWS サービスエンドポイントとの間のトラフィックのタイミングとレスポンスコードをキャプチャできます。詳細については、「AWS X-Ray デベロッパーガイド」の「AWS X-Ray とは」を参照してください。

また、サービスがどのように相互にネットワーキングされているかについての AWS X-Ray グラフを見ることもできます。または、これらを使用して、各サービス間リンクのパフォーマンスに関する総合統計情報を見ることができます。最後に、特定のトランザクションを詳しく調べることで、ネットワーク呼び出しを表すセグメントがその特定のトランザクションにどのように関連付けられているかを確認できます。

これらの機能を使用することで、ネットワーキングのボトルネックがあるかどうか、またはネットワーク内の特定のサービスが期待どおりに動作していないかどうかを特定できます。

VPC Flow Logs

Amazon VPC フローログを使用して、ネットワークパフォーマンスを分析したり、接続の問題をデバッグしたりできます。VPC フローログを有効にすると、VPC 内のすべての接続のログをキャプチャできます。この接続には、Elastic Load Balancing、Amazon RDS、NAT ゲートウェイ、および使用している可能性があるその他の主要な AWS サービスに関連付けられているネットワークインターフェイスへの接続が含まれます。詳細については、「Amazon VPC ユーザーガイド」の「VPC フローログを使用した IP トラフィックのログ記録」を参照してください。

ネットワーク調整のヒント

ネットワーキングを改善するために微調整できる設定がいくつかあります。

nofile ulimit

アプリケーションのトラフィックが多く、多数の同時接続を処理することが予想される場合は、許可されるファイル数に対しシステムクォータを考慮する必要があります。多数のネットワークソケットが開いている場合は、それぞれをファイル記述子で表す必要があります。ファイル記述子のクォータが低すぎると、ネットワークソケットが制限されます。この結果、接続が失敗したり、エラーが発生したりします。コンテナ固有のクォータは、Amazon ECS タスク定義内のファイル数に合わせて更新できます。(AWS Fargate ではなく) Amazon EC2 で実行している場合は、基盤となる Amazon EC2 インスタンスでこれらのクォータを調整する必要もあります。

sysctl ネット

もう 1 つのカテゴリの調整可能な設定は、`sysctl` ネット設定です。選択した Linux ディストリビューションの特定の設定を参照する必要があります。この設定の多くでは、読み取りバッファと書き込みバッファのサイズが調整されます。これは、コンテナが多数ある大規模な Amazon EC2 インスタンスを実行する場合に役立つことがあります。

Amazon ECS タスクの起動時間を最適化する

タスク起動をスピードアップするために、以下の推奨事項を考慮してください。

- コンテナイメージと binpack インスタンスをキャッシュします。

  EC2 を使用する場合、Amazon ECS コンテナエージェントのプル動作を `ECS_IMAGE_PULL_BEHAVIOR: prefer-cached` のように設定できます。キャッシュされたイメージがない場合に、リモートでイメージがプルされます。それ以外の場合は、インスタンスにキャッシュされたイメージが使用されます。キャッシュされたイメージが削除されないように、そのコンテナの自動イメージクリーンアップはオフになっています。これにより、次回起動時のイメージのプルタイムが短縮されます。コンテナインスタンスのタスク密度が高い場合、キャッシュの効果はさらに大きくなります。コンテナインスタンスは `binpack` 配置戦略を使用して設定できます。コンテナイメージのキャッシュは、通常はコンテナイメージのサイズが大きい Windows ベースのワークロード (数十 GB) に特に役立ちます。`binpack` 配置戦略を使用する場合は、Elastic Network Interface (ENI) トランキングを使用して、各コンテナインスタンスに `awsvpc` ネットワークモードでより多くのタスクを配置することを検討することもできます。ENI トランキングを使用すると、`awsvpc` モードで実行できるタスク数が増加します。たとえば、同時に 2 つの

タスクしか実行できない c5.large インスタンスの場合、ENI トランキングでは最大 10 のタスクを実行できます。

- 最適なネットワークモードを選択する

  awsvpc ネットワークモードが理想的な例は多数ありますが、このネットワークモードは本質的にタスク起動レイテンシーを増加させる可能性があります。これは、awsvpc モード内の各タスクについて、Amazon EC2 API を呼び出して ENI をプロビジョニングしてアタッチする必要があるため、タスク起動に数秒のオーバーヘッドが追加されるためです。対照的に、awsvpc ネットワークモードを使用する主な利点は、各タスクにトラフィックを許可または拒否するセキュリティグループがあることです。つまり、タスクとサービス間の通信をよりきめ細かく制御できる柔軟性が増します。デプロイ速度を優先する場合は、bridge モードを使用してタスクの起動をスピードアップすることを検討してください。詳細については、「the section called “AWSVPC ネットワークモード”」を参照してください。

- タスク起動のライフサイクルを追跡して、最適化の機会を見つけます。

  多くの場合、アプリケーションが起動するまでにかかる時間を知ることは困難です。アプリケーションの起動時に、コンテナイメージの起動、起動スクリプトの実行、その他の設定を行うと、驚くほど時間がかかることがあります。タスクメタデータエンドポイントを使用してメトリクスを投稿することで、コンテナメタデータレスポンスの StartedAt からタスクまたはサービスまでの StartedAt 時間を追跡できます。このデータにより、アプリケーションが起動時間全体にどのような影響を与えているかを把握し、アプリケーション固有の不要なオーバーヘッドを減らし、コンテナイメージを最適化できる領域を見つけることができます。詳細については、「Amazon ECS 自動スケーリングとキャパシティ管理におけるベストプラクティス」を参照してください。

- 最適なインスタンスタイプを選択する (EC2 の場合)

  適切なインスタンスタイプは、タスクに設定したリソース予約 (CPU、メモリなど) に基づいて選択します。したがって、インスタンスのサイズを決定する際には、1 つのインスタンスに配置できるタスクの数を計算できます。適切に配置されたタスクの簡単な例は、m5.large インスタンス (2 vCPU と 8 GB のメモリをサポート) で 0.5 vCPU と 2 GB のメモリ予約を必要とする 4 つのタスクをホストすることです。このタスク定義の予約は、インスタンスのリソースを最大限に活用します。

Amazon ECS を大規模に運用する

Amazon ECS を大規模に運用し始める際には、Amazon ECS および Amazon ECS と統合する AWS のサービス のサービスクォータと API スロットリングがどのように影響するかを検討します。

トピック

- Amazon ECS Service Quotas と API スロットリング制限
- Amazon ECS のスロットリングに関する問題を処理する

Amazon ECS Service Quotas と API スロットリング制限

Amazon ECS は、Elastic Load Balancing、AWS Cloud Map、Amazon EC2 など、いくつかの AWS のサービス と統合されています。この緊密な統合により、Amazon ECS ではサービスのロードバランシング、サービス検出、タスクネットワーク、クラスターの自動スケーリングなどのいくつかの機能を利用できます。Amazon ECS および、統合されているその他の AWS のサービス はすべて、一貫したパフォーマンスと利用率を確保するために、サービスクォータと API レート制限を維持します。また、これらのサービスクォータは、必要以上のリソースを誤ってプロビジョニングすることを防ぎ、請求額を増やす可能性のある悪意のある行為からも保護します。

サービスクォータと AWS API のレート制限をよく理解しておけば、予期しないパフォーマンスの低下を心配することなく、ワークロードのスケーリングを計画できます。詳細については、「Amazon ECS の Service Quotas」「Request throttling for the Amazon ECS API」を参照してください。

Amazon ECS でワークロードをスケーリングする場合は、次のサービスクォータを検討します。サービスクォータの引き上げをリクエストする方法については、「AWS マネジメントコンソール での Amazon ECS と AWS Fargate サービスクォータの管理」を参照してください。

- AWS Fargate には、各 のタスクで同時に実行できるタスクの数を制限するクォータがあります。AWS リージョンAmazon ECS では、オンデマンドタスクと Fargate Spot タスクの両方にクォータが設定されています。各サービスクォータには、Fargate で実行するすべての Amazon EKS ポッドも含まれます。Fargate クォータの詳細については、「AWS Fargate の Amazon Elastic Container Service ユーザーガイド」にある「AWS Fargate サービスクォータ」を参照してください。
- Amazon EC2 インスタンスで実行されるタスクの場合、各クラスターに登録できる Amazon EC2 インスタンスの最大数は 5,000 です。Auto Scaling グループの容量プロバイダーで Amazon ECS クラスターの自動スケーリングを使用する場合、またはクラスターの Amazon EC2 インスタンスを自分で管理する場合、このクォータがデプロイのボトルネックになる可能性があります。さらに容量が必要な場合は、クラスターをさらに作成するか、サービスクォータの引き上げをリクエストすることができます。
- Auto Scaling グループのキャパシティプロバイダーで Amazon ECS クラスターの自動スケーリングを使用する場合は、サービスをスケーリングする際に `Tasks in the PROVISIONING state`

`per cluster` クォータを考慮してください。このクォータは、キャパシティプロバイダーがキャパシティを増やすことができる各クラスターの `PROVISIONING` 状態にあるタスクの最大数です。多数のタスクを同時に起動すると、すぐにこのクォータに達してしまいます。たとえば、それぞれ数百のタスクを含む数十のサービスを同時にデプロイする場合です。この場合、クラスターのキャパシティが不足すると、キャパシティープロバイダーは新しいコンテナインスタンスを起動してタスクを配置する必要があります。キャパシティープロバイダーが追加の Amazon EC2 インスタンスを起動している間、Amazon ECS サービススケジューラーは引き続きタスクを並行して起動する可能性があります。ただし、クラスターのキャパシティが不十分なため、このアクティビティが抑制される可能性があります。Amazon ECS サービススケジューラは、新しいコンテナインスタンスが起動したときにタスク配置を再試行するためのバックオフおよびエクスポネンシャルスロットリング戦略を実装しています。その結果、デプロイやスケールアウトに時間がかかる可能性があります。このような状況を回避するために、以下のいずれかの方法でサービスのデプロイを計画できます。クラスターの容量を増やす必要のないタスクを大量に展開するか、新しいタスクの起動に備えて余剰のクラスター容量を確保しておくかのどちらかです。

ワークロードをスケーリングする際には Amazon ECS サービスクォータを考慮することに加えて、Amazon ECS と統合されている他の AWS のサービス 用のサービスクォータも考慮してください。次のセクションでは、各サービスの主要なレート制限について詳しく説明し、スロットリングの潜在的な問題に対処するための推奨事項を提供します。

Elastic Load Balancing

Fargate の Amazon ECS サービスは、Elastic Load Balancing を使用してタスク間でトラフィックを均等に分散するように設定できます。ロードバランサーを選択する方法の詳細については、「ロードバランサーを使用して Amazon ECS サービストラフィックを分散する」を参照してください。

Elastic Load Balancing のサービスクォータ

ワークロードをスケーリングするときは、以下の Elastic Load Balancing のサービスクォータを考慮してください。Elastic Load Balancing のサービスクォータのうち、ほとんどは調整可能で、Service Quotas コンソールでリクエストできます。

Application Load Balancer:

Application Load Balancer を使用する際、ユースケースによっては、以下のクォータ増加をリクエストする必要がある場合があります。

- Application Load Balancer の背後にあるターゲットの数である `Targets per Application Load Balancer` クォータ。

- ターゲットグループの背後にあるターゲットの数である `Targets per Target Group per Region` クォータ。

詳細については、「Quotas for your Application Load Balancers」を参照してください。

Network Load Balancer

Network Load Balancer に登録できるターゲットの数には、より厳しい制限があります。Network Load Balancer を使用する場合、クロスゾーンサポートを有効にする必要があることがよくあります。これには、各 Network Load Balancer のアベイラビリティーゾーンあたりの `Targets per Availability Zone Per Network Load Balancer` 最大ターゲット数に対する追加のスケーリング制限が伴います。詳細については、「Network Load Balancer のクォータ」を参照してください。

Elastic Load Balancing API のスロットリング

ロードバランサーを使用するように Amazon ECS サービスを設定する場合、サービスが正常であると見なされるには、ターゲットグループのヘルスチェックに合格する必要があります。これらのヘルスチェックを実行するために、Amazon ECS はユーザーに代わって Elastic Load Balancing API 操作を呼び出します。アカウントにロードバランサーが設定されたサービスが多数ある場合、特に `RegisterTarget`、`DeregisterTarget`、`DescribeTargetHealth` Elastic Load Balancing API 操作に関してスロットリングが発生する可能性があるため、サービスのデプロイが遅くなる可能性があります。スロットリングが発生すると、Amazon ECS サービスのイベントメッセージにスロットリングエラーが発生します。

AWS Cloud Map API でスロットリングが発生した場合は、AWS Cloud Map API のスロットリング制限を引き上げる方法について サポート までお問い合わせください。スロットリングエラーの監視とトラブルシューティングの詳細については、「Handling throttling issues」を参照してください。

Elastic Network Interface

タスクで `awsvpc` ネットワークモードを使用する場合、Amazon ECS はタスクごとに独自の Elastic Network Interface (ENI) をプロビジョニングします。Amazon ECS サービスが Elastic Load Balancing ロードバランサーを使用する場合、これらのネットワークインターフェイスは、サービスで定義された適切なターゲットグループのターゲットとしても登録されます。

Elastic Network Interface のサービスクォータ

`awsvpc` ネットワークモードを使用するタスクを実行すると、固有のエラスティックネットワークインターフェイスが各タスクにアタッチされます。インターネット経由でこれらのタスクにアクセス

する必要がある場合は、タスクのエラスティックネットワークインターフェイスにパブリック IP アドレスを割り当てます。Amazon ECS ワークロードをスケーリングするときは、次の 2 つの重要なクォータを考慮してください。

- アカウントの AWS リージョン におけるネットワークインターフェイスの最大数である `Network interfaces per Region` クォータ。
- AWS リージョン に含まれる Elastic IP アドレスの最大数である `Elastic IP addresses per Region` クォータ。

これらのサービスクォータは両方とも調整可能で、Service Quotas コンソールから増加をリクエストできます。詳細については、「Amazon VPC サービスクォータ」を参照してください。

Amazon EC2 インスタンスでホストされている Amazon ECS ワークロードの場合、`awsvpc` ネットワークモードを使用するタスクを実行するときは、各 Amazon EC2 インスタンスの最大ネットワークインスタンスの数である `Maximum network interfaces` サービスクォータを考慮してください。このクォータは、1 つのインスタンスに配置できるタスクの数を制限します。クォータを調整することはできず、Service Quotas コンソールでは使用できません。詳細については、「Amazon EC2 ユーザーガイド」の「各インスタンスタイプのネットワークインターフェイスあたりの IP アドレス」を参照してください。

Amazon EC2 インスタンスにアタッチできるネットワークインターフェイスの数は変更できませんが、エラスティックネットワークインターフェイスのトランキング機能を使用して利用可能なネットワークインターフェイスの数を増やすことができます。たとえば `c5.large` インスタンスにはデフォルトでネットワークインターフェイスを最大 3 つアタッチできます。このインスタンスのプライマリネットワークインターフェイスも、1 個としてカウントされます。そのため、このインスタンスに追加で 2 個のネットワークインターフェイスをアタッチできます。`awsvpc` ネットワークモードを使用する各タスクにはネットワークインターフェイスが必要なため、通常このインスタンスタイプでは、これらのタスクを 2 つのみ実行できます。これにより、クラスターの容量が十分に活用されない可能性があります。エラスティックネットワークインターフェイスでトランキングを有効にすると、ネットワークインターフェイスの密度を上げて、各インスタンスにより多くのタスクを配置できます。トランキングをオンにすると、1 つの `c5.large` インスタンスに最大 12 のネットワークインターフェースを設定できます。インスタンスはプライマリネットワークインターフェイスを持ち、Amazon ECS は「トランク」ネットワークインターフェイスを作成し、アタッチします。その結果、この構成では、デフォルトの 2 つのタスクではなく、10 のタスクをインスタンスで実行できます。詳細については、「Elastic network interface trunking」を参照してください。

エラスティックネットワークインターフェイス API のスロットリング

awsvpc ネットワークモードを使用するタスクを実行する場合、Amazon ECS は次の Amazon EC2 API に依存します。これらの API にはそれぞれ異なる API スロットリングがあります。詳細については、「Amazon EC2 API のリクエストスロットリング」を参照してください。

- CreateNetworkInterface
- AttachNetworkInterface
- DetachNetworkInterface
- DeleteNetworkInterface
- DescribeNetworkInterfaces
- DescribeVpcs
- DescribeSubnets
- DescribeSecurityGroups
- DescribeInstances

エラスティックネットワークインターフェイスのプロビジョニングワークフロー中に Amazon EC2 API コールがスロットリングされた場合、Amazon ECS サービススケジューラは自動的にエクスポネンシャルバックオフを行って再試行します。これらの自動廃止により、タスクの起動が遅れ、デプロイ速度が遅くなることがあります。API スロットリングが発生すると、サービスイベントメッセージにメッセージ `Operations are being throttled. Will try again later.` が表示されます。Amazon EC2 API でスロットリングが継続的に発生する場合は、API のスロットリング制限を引き上げる方法について サポート までお問い合わせください。スロットリングエラーの監視とトラブルシューティングの詳細については、「スロットリング問題の処理」を参照してください。

AWS Cloud Map

Amazon ECS サービス検出では、AWS Cloud Map API を使用して Amazon ECS サービスの名前空間を管理します。サービスに多数のタスクがある場合は、以下の推奨事項を考慮してください。詳細については、「Amazon ECS サービス検出に関する考慮事項」を参照してください。

AWS Cloud Map サービスクォータ

Amazon ECS サービスがサービス検出を使用するように設定されている場合、サービスの最大タスク数である `Tasks per service` クォータは、そのサービスの最大インスタンス数である AWS Cloud Map `Instances per service` サービスクォータの影響を受けます。特に、AWS Cloud

Map サービスクォータにより、実行できるタスクの数は、サービスあたり最大 1,000 タスクに減少します。AWS Cloud Map のクォータは変更できません。詳細については、「AWS Cloud Map サービスクォータ」を参照してください。

AWS Cloud Map API スロットリング

Amazon ECS はユーザーに代わってListInstances、GetInstancesHealthStatus、RegisterInstance、およびDeregisterInstance AWS Cloud Map API を呼び出します。これらはサービス検出を支援し、タスクを起動するとヘルスチェックを実行します。多数のタスクを伴うサービス検出を使用する複数のサービスを同時にデプロイすると、AWS Cloud Map API のスロットリング制限を超える可能性があります。この場合、Amazon ECS サービスのイベントメッセージに次のようなメッセージが表示される場合があります: Amazon ECS サービスイベントで Operations are being throttled. Will try again later が発生しました。デプロイとタスクの起動速度が遅くなっています。AWS Cloud Map には、これらの API のスロットリング制限は記録されていません。これらのスロットリングが発生した場合は、API のスロットリング制限を引き上げる方法について サポート までお問い合わせください。このようなスロットリングエラーの監視とトラブルシューティングの詳細については、「Handling throttling issues」を参照してください。

Amazon ECS のスロットリングに関する問題を処理する

スロットリングエラーは、同期スロットリングと非同期スロットリングの 2 つの主要なカテゴリに分類されます。

同期スロットリング

同期スロットリングが発生すると、すぐに Amazon ECS からエラーレスポンスが届きます。このカテゴリは通常、タスクの実行中またはサービスの作成中に Amazon ECS API を呼び出すと発生します。影響を受けるスロットリングと関連するスロットリング制限の詳細については、「Amazon ECS API リクエストのスロットリング」を参照してください。

アプリケーションが、AWS CLI、AWS SDK などを使用して API リクエストを開始すると、API スロットリングを修正できます。そのためには、エラーを処理するようにアプリケーションを設計するか、API コールの再試行ロジックを使用してエクスポネンシャルバックオフとジッター戦略を実装します。詳細については、「タイムアウト、リトライ、ジッターによるバックオフ」を参照してください。

AWS SDK を使用する場合、自動再試行ロジックは組み込まれており、設定可能です。

非同期スロットリング

非同期スロットリングは、Amazon ECS または CloudFormation がユーザーに代わって API を呼び出してリソースをプロビジョニングする非同期ワークフローが原因で発生する場合があります。Amazon ECS がユーザーに代わって呼び出す AWS API がどれなのかを知ることは重要です。たとえば、`CreateNetworkInterface` API は `awsvpc` ネットワークモードを使用するタスクに対して呼び出され、`DescribeTargetHealth` API はロードバランサーに登録されたタスクのヘルスチェックを実行するときに呼び出されます。

ワークロードがかなり大きくなると、これらの API オペレーションがスロットリングされる可能性があります。つまり、Amazon ECS や呼び出される AWS のサービス による制限を超えるほどスロットリングされる可能性があります。たとえば、数百のサービスをデプロイし、それぞれが `awsvpc` ネットワークモードを使用する数百のタスクを同時に実行する場合、Amazon ECS は `CreateNetworkInterface` などの EC2 API 操作および `RegisterTarget`、`DescribeTargetHealth` などの Elastic Load Balancing API 操作を呼び出し、それぞれエラスティックネットワーク、ロードバランサーを登録します。これらの API コールが API の制限を超えると、スロットリングエラーが発生する可能性があります。以下は、サービスイベントメッセージに含まれる Elastic Load Balancing スロットリングエラーの例です。

```
{
   "userIdentity":{
      "arn":"arn:aws:sts::111122223333:assumed-role/AWSServiceRoleForECS/ecs-service-
scheduler",
      "eventTime":"2022-03-21T08:11:24Z",
      "eventSource":"elasticloadbalancing.amazonaws.com",
      "eventName":" DescribeTargetHealth ",
      "awsRegion":"us-east-1",
      "sourceIPAddress":"ecs.amazonaws.com",
      "userAgent":"ecs.amazonaws.com",
      "errorCode":"ThrottlingException",
      "errorMessage":"Rate exceeded",
      "eventID":"0aeb38fc-229b-4912-8b0d-2e8315193e9c"
   }
}
```

これらの API コールがアカウント内の他の API トラフィックと制限を共有している場合、サービスイベントとして送信されても監視が難しい場合があります。

スロットリングを監視する

どの API リクエストがスロットリングされ、誰がリクエストを発行したかを特定することが重要です。AWS CloudTrail を使用すると、スロットリングを監視し、CloudWatch、Amazon Athena、Amazon EventBridge と統合できます。CloudWatch Logs にイベントを送信するように CloudTrail を設定することができます。CloudWatch Logs のログインサイトはイベントを解析して分析します。これにより、呼び出しを行ったユーザーや IAM ロール、行われた API コールの数など、スロットリングイベントの詳細が特定されます。詳細については、「Amazon CloudWatch Logs で CloudTrail ログファイルを監視する」を参照してください。

CloudWatch Logs Insights の詳細および、ログファイルのクエリ方法については、「CloudWatch Logs Insights を使用したログデータの分析」を参照してください。

Amazon Athena では、標準 SQL を使用してクエリを作成し、データを分析できます。たとえば、Athena テーブルを作成して CloudTrail イベントを解析することができます。詳細については、「CloudTrail コンソールを使用して CloudTrail ログ用 Athena テーブルを作成する」を参照してください。

Athena テーブルを作成すると、次のような SQL クエリを使用して `ThrottlingException` エラーを調査できます。

`user-input` を独自の値に置き換えます。

```
select eventname, errorcode,eventsource,awsregion, useragent,COUNT(*) count
FROM cloudtrail_table-name
where errorcode = 'ThrottlingException'
AND eventtime between '2024-09-24T00:00:08Z' and '2024-09-23T23:15:08Z'
group by errorcode, awsregion, eventsource, useragent, eventname
order by count desc;
```

Amazon ECS は、Amazon EventBridge にもイベント通知を送信します。リソース状態変更イベントおよびサービスアクションイベントがあります。これらには、`ECS_OPERATION_THROTTLED`、`SERVICE_DISCOVERY_OPERATION_THROTTLED` などの API スロットリングイベントが含まれます。詳細については、「Amazon ECS サービスアクションイベント」を参照してください。

これらのイベントは、応答アクションを実行する目的で、AWS Lambda などのサービスによって利用される可能性があります。詳細については、「Amazon ECS イベントの処理」を参照してください。

スタンドアロンタスクを実行する場合、RunTask などの一部の API 操作は非同期となり、再試行操作は自動的に実行されません。このような場合は、EventBridge と統合している AWS Step Functions などのサービスを使用して、スロットリングされた操作や失敗した操作を再試行できます。詳細については、「コンテナタスクの管理 (Amazon ECS、Amazon SNS)」を参照してください。

CloudWatch を使用してスロットリングを監視する

CloudWatch では、[AWS リソース別] で Usage ネームスペースの API 使用状況をモニタリングできます。これらのメトリクスは タイプ [API]、メトリック名 [CallCount] で記録されます。これらのメトリクスが特定のしきい値に達するたびに開始するアラームを作成できます。詳細については、「サービスクォータの視覚化とアラームの設定」を参照してください。

CloudWatch には異常検出機能もあります。この機能は、機械学習を使用して、有効にしたメトリクスの特定の動作に基づいて分析し、ベースラインを確立します。異常な API アクティビティが発生した場合は、この機能を CloudWatch アラームと合わせて使用できます。詳細については、「CloudWatch の異常検出の使用方法」を参照してください。

スロットリングエラーをプロアクティブにモニタリングすることで、サポート に連絡することで、関連するスロットリング制限を引き上げたり、アプリケーション固有のニーズに関するガイダンスを受けたりできます。

Amazon ECS 自動スケーリングとキャパシティ管理におけるベストプラクティス

Amazon ECS では、全サイズのコンテナ化されたアプリケーションワークロードを実行できます。これには、最小限のテスト環境と、グローバル規模で運用される大規模な本番環境が含まれます。

Amazon ECS と併せて、すべての AWS サービスと同様に、使用した分だけお支払いいただきます。アプリケーションを適切に設計することで、必要なときに必要なリソースのみを消費することでコストを削減できます。

以下の推奨事項は、Amazon ECS ワークロードを実行してサービスレベルの目標を達成しつつ、費用対効果の高い運用を行う方法を示しています。

トピック

- Amazon ECS のタスクサイズの決定

- Amazon ECS サービスの自動スケーリングの最適化
- Amazon ECS のキャパシティとアベイラビリティ
- Amazon ECS クラスターのキャパシティ
- Amazon ECS の Fargate タスクサイズの選択
- Amazon EC2 でキャパシティプロバイダーを使用して Amazon ECS クラスターのキャパシティプロビジョニングを高速化する

Amazon ECS のタスクサイズの決定

Amazon ECS にコンテナをデプロイする際に、最も重要な選択項目の 1 つとなるのがコンテナとタスクのサイズです。コンテナサイズとタスクサイズは、スケーリングとキャパシティプランニングに不可欠な要素です。

Amazon ECS では、CPU とメモリの 2 種類のリソースメトリクスがキャパシティに使用されます。Amazon ECS では、CPU 量 は、フル vCPU の 1024 分の 1 の単位で測定されます (1024 ユニットは 1 つの全 vCPU に相当します)。Amazon ECS はメモリをメガバイト単位で測定します。

タスク定義では、リソースの予約と制限を宣言できます。

予約を宣言する際は、タスクに最低限必要な量のリソースを宣言します。タスクはリクエストに必要となる最低限の量のリソースを受け取ります。アプリケーションは、宣言された予約量よりも多くの CPU またはメモリを使用できる可能性があります。ただし、これには宣言した制限も適用されます。

予約量を超えるリソースを使用することをバーストと呼びます。バーストとは、アプリケーションが予約したよりも多くのリソースを使用するが、宣言された制限内にとどまることを意味します。Amazon ECS は、予約量を保証します。たとえば、Amazon EC2 インスタンスを使用してキャパシティを提供する場合、Amazon ECS は予約量を満たせないインスタンスにはタスクを課しません。

制限は、コンテナまたはタスクが使用できる CPU ユニットまたはメモリの最大量です。コンテナがこの制限よりも多くの CPU を使用しようとすると、Amazon ECS によってスロットリングされます。コンテナがこの制限よりも多くのメモリを使用しようとすると、Amazon ECS はコンテナを停止します。

これらの値の決定は簡単ではありません。アプリケーションに最も適した値は、そのアプリケーションのリソース要件に大きく依存します。

アプリケーションの負荷テストは、リソース要件をうまく計画する上で重要です。負荷テストは、アプリケーションの要件をよりよく理解するのに役立ちます。

ステートレスアプリケーション

ロードバランサーの背後にあるアプリケーションなど、水平にスケーリングするタイプのステートレスアプリケーションでは、まずそのアプリケーションがリクエストを処理するときに消費するメモリの量がどの程度になるかを確認することをお勧めします。

そのためには、`ps` や `top` などの従来のツールを使用できます。CloudWatch Container Insights などのモニタリングソリューションを使用することもできます。

CPU 予約を決定する際には、ビジネス要件を満たすためにアプリケーションをどのようにスケーリングすべきかを考慮します。

256 CPU ユニット (つまりフル vCPU の 4 分の 1) などの小さめの CPU 予約を行うことで、コストを最小限に抑えながらきめ細かくスケールアウトできます。ただし、需要が急増した時にスケーリング速度が追い付かない可能性があります。

より大きな CPU 予約量を使用すると、より迅速にスケールインおよびスケールアウトできます。これにより、需要の急増に迅速に対応できます。ただし、CPU の予約量が多いほどコストが高くなります。

その他のアプリケーション

シングルトンワーカーやデータベースサーバーなど、水平にスケーリングしないタイプのアプリケーションでは、利用可能な容量とコストが最も重要な考慮事項となります。

トラフィックを処理し、サービスレベル目標を達成するためには、負荷テストの結果を確認したうえで必要なメモリと CPU の量を決定します。Amazon ECS は、アプリケーションが十分な容量のあるホストに確実に配置されるようにします。

Amazon ECS サービスの自動スケーリングの最適化

Amazon ECS サービスは管理されたタスクのコレクションです。各サービスには、関連するタスク定義、必要なタスク数、オプションの配置戦略があります。

Amazon ECS サービスの自動スケーリングは、アプリケーション自動スケーリングサービスを通して行われます。Application Auto Scaling は CloudWatch メトリクスをスケーリングメトリクスの

ソースとして使用します。また、CloudWatch アラームを使用して、サービスをスケールインまたはスケールアウトするタイミングのしきい値を設定します。

スケーリングのしきい値を指定します。メトリクスターゲットを設定できます。これはターゲット追跡スケーリングと呼ばれています。しきい値を指定することもできます。これはステップスケーリングと呼ばれています。

アプリケーション自動スケーリングを設定した後、サービスに必要となる適切なタスク数が継続的に計算されます。また、必要なタスク数を変更する必要がある場合、スケールアウトまたはスケールインによって Amazon ECS に通知します。

サービスの自動スケーリングを効果的に使用するには、適切なスケーリングメトリクスを選択する必要があります。メトリクスの選択方法については、以下のセクションで説明します。

アプリケーションの特徴付け

アプリケーションを適切にスケーリングするには、アプリケーションをスケールインする条件とスケールアウトするタイミングを知る必要があります。

基本的に、需要がキャパシティを上回ると予測される場合は、アプリケーションをスケールアウトする必要があります。逆に、リソースが需要を上回る場合は、アプリケーションをスケールインすることでコストを節約できます。

使用率メトリクスの特定

効果的にスケーリングするには、使用率または飽和度を示すメトリクスを特定する必要があります。このメトリクスがスケーリングに役立つためには、以下の特性を備えている必要があります。

- メトリクスは需要と相関している必要があります。リソースを一定に保っていても需要が変化する場合は、メトリクスの値も変化させる必要があります。需要が増減すると、メトリクスも増減します。
- メトリクスの値はキャパシティに比例してスケールインする必要があります。需要が一定であれば、リソースを追加すると、メトリクスの値も比例して変化する必要があります。そのため、タスクの数を 2 倍にすると、指標は 50% 減少するはずです。

使用率メトリクスを特定する最良の方法は、ステージング環境などの実稼働前の環境で負荷テストを行うことです。商用およびオープンソースの負荷テストソリューションは広く利用可能です。これらのソリューションは通常、合成負荷を生成することも、実際のユーザートラフィックをシミュレートすることもできます。

負荷テストのプロセスを開始するには、アプリケーションの使用率メトリクスを表示するダッシュボードを最初に構築する必要があります。これらのメトリクスには、CPU 使用率、メモリ使用率、I/O 操作、I/O キューの深さ、ネットワークスループットが含まれます。これらのメトリクスは CloudWatch Container Insights などのサービスを使用して収集できます。また、Amazon Managed Grafana と一緒に Amazon Managed Service for Prometheus を使用して収集することもできます。このプロセスでは、アプリケーションの応答時間や作業完了率に関するメトリクスを必ず収集してプロットしてください。

負荷テストを実施する際は、まず小さなリクエストまたはジョブ挿入率から開始します。アプリケーションがウォームアップできるように、このレートを数分間一定に保ってください。その後、速度をゆっくりと上げて、数分間一定に保ちます。アプリケーションの応答時間または完了時間が遅すぎてサービスレベル目標 (SLO) を達成できなくなるまで、このサイクルを繰り返し、その都度レートを上げていきます。

負荷テストを行う際には、各使用率メトリクスを調べます。負荷とともに増加するメトリクスは、最適な使用率メトリクスとして役立つ最有力候補です。

次に、飽和状態に達しているリソースを特定します。同時に、使用率メトリクスを調べて、最初に高いレベルで横ばいになったものを調べます。または、ピークに達して最初にアプリケーションをクラッシュさせるものを調べます。たとえば、負荷を追加するにつれて CPU 使用率が 0% から 70 ~ 80% に増加し、さらに多くの負荷を追加してもそのレベルにとどまる場合は、CPU が飽和状態であると言えます。CPU アーキテクチャによっては、100% に達しない可能性があります。たとえば、負荷を追加するとメモリ使用率が増加し、タスクまたは Amazon EC2 インスタンスのメモリ制限に達するとアプリケーションが突然クラッシュしたとします。このような状況では、メモリが完全に消費された可能性があります。アプリケーションが複数のリソースを消費する可能性があります。そのため、最初に枯渇したリソースを表すメトリクスを選択します。

最後に、タスクまたは Amazon EC2 インスタンスの数を 2 倍にしてから、負荷テストを再試行します。キーメトリクスが以前の半分の速度で、増加または減少すると仮定します。この場合、メトリクスはキャパシティに比例します。これは自動スケーリングに適した使用率メトリクスです。

次に、この架空のシナリオを考えてみましょう。アプリケーションの負荷テストを行い、1 秒あたり 100 リクエストで CPU 使用率が最終的に 80% に達したとします。負荷を追加しても、CPU 使用率は上昇しなくなります。ただし、アプリケーションの応答は遅くなります。次に、負荷テストを再度実行して、タスクの数を 2 倍にしますが、速度は以前のピーク値に保たれます。CPU の平均使用率が約 40% に低下した場合、CPU 平均使用率がスケーリングメトリクスの候補として適しています。一方、タスク数を増やしても CPU 使用率が 80% のままであれば、平均 CPU 使用率は適切なスケーリング指標ではありません。その場合は、適切なメトリクスを見つけるためにさらに調査する必要があります。

一般的なアプリケーションモデルおよびスケーリングプロパティ

あらゆる種類のソフトウェアを AWS で実行できます。多くのワークロードは自社開発ですが、他のワークロードは一般的なオープンソースソフトウェアをベースにしています。その出所に関係なく、サービスの一般的なデザインパターンがいくつか確認されています。どの方法で効果的にスケーリングできるかは、そのパターンに大きく依存します。

効率的な CPU バウンドサーバー

効率的な CPU バウンドサーバーは、CPU とネットワークスループット以外のリソースをほとんど使用しません。各リクエストはアプリケーションだけで処理できます。リクエストはデータベースなど他のサービスには依存しません。アプリケーションは何十万もの同時リクエストを処理でき、そのために複数の CPU を効率的に活用できます。各リクエストは、メモリーオーバーヘッドの少ない専用スレッドによって処理されるか、リクエストを処理する各 CPU で実行される非同期イベントループによって処理されます。アプリケーションの各レプリカは、同じようにリクエストを処理できます。CPU が枯渇する前に枯渇する可能性のあるリソースは、ネットワーク帯域幅のみです。CPU バウンドサービスでは、ピークスループットでもメモリ使用率は、利用可能なリソースのほんの一部です。

このタイプのアプリケーションには、CPU ベースの自動スケーリングを使用できます。このアプリケーションはスケーリングに関して最大限の柔軟性を備えています。より大きな Amazon EC2 インスタンスまたは Fargate vCPU を提供することで、垂直方向にスケーリングできます。また、レプリカをさらに追加することで、水平方向にスケールすることもできます。レプリカを追加したり、インスタンスサイズを 2 倍にしたりすると、容量に対する平均 CPU 使用率が半分に削減されます。

このアプリケーションに Amazon EC2 の容量を使用している場合は、c5 または c6g ファミリーなどのコンピューティング最適化インスタンスに配置することを検討してください。

効率的なメモリバウンドサーバー

効率的なメモリバウンドサーバーは、リクエストごとに大量のメモリを割り当てます。同時実行性が最大になると、必ずしもスループットが高くなるわけではありませんが、CPU リソースが使い果たされる前にメモリが使い果たされます。リクエストに関連するメモリは、リクエストが終了すると解放されます。メモリに余裕がある限り、追加のリクエストを受け付けることができます。

このタイプのアプリケーションには、メモリベースの自動スケーリングを使用できます。このアプリケーションはスケーリングに関して最大限の柔軟性を備えています。より大きな Amazon EC2 または Fargate メモリリソースを提供することで、両方を垂直方向にスケーリングできます。また、レプリカをさらに追加することで、水平方向にスケールすることもできます。レプリカを追加したり、イ

ンスタンスサイズを 2 倍にしたりすると、容量に対する平均メモリ使用率を半分に減らすことができます。

このアプリケーションに Amazon EC2 キャパシティを使用している場合は、r5 または r6g ファミリーなどのメモリ最適化インスタンスにキャパシティを割り当てることを検討してください。

メモリに制約のあるアプリケーションの中には、同時実行数を減らしても使用されるメモリが減らないように、リクエストの終了時にそのリクエストに関連するメモリを解放しないものがあります。このため、メモリベースのスケーリングを使用することはお勧めしません。

ワーカーベースのサーバー

ワーカーベースのサーバーは、個々のワーカースレッドごとに 1 つのリクエストを次々に処理します。ワーカースレッドは POSIX スレッドなどの軽量スレッドでもかまいません。UNIX プロセスなど、より負荷の大きいスレッドの場合もあります。どのスレッドであっても、アプリケーションがサポートできる同時実行数は常に最大です。通常、同時実行数の制限は、使用可能なメモリリソースに比例して設定されます。同時実行数の上限に達すると、アプリケーションにより、追加のリクエストはバックログキューに入れられます。バックログキューがオーバーフローすると、アプリケーションはそれ以降の受信リクエストを直ちに拒否します。このパターンに当てはまる一般的なアプリケーションには、Apache Web サーバーや Gunicorn などがあります。

このアプリケーションを拡張するには、通常、リクエストの同時実行性が最適なメトリクスです。各レプリカには同時実行数の制限があるため、平均制限に達する前にスケールアウトすることが重要です。

リクエストの同時実行メトリクスを取得する最良の方法は、アプリケーションに CloudWatch にレポートさせることです。アプリケーションの各レプリカは、同時リクエストの数をカスタムメトリクスとして高い頻度で公開できます。この頻度は少なくとも 1 分に 1 回に設定することをおすすめします。複数のレポートを収集したら、平均同時実行数をスケーリングメトリクスとして使用できます。このメトリクスは、同時実行数の合計をレプリカ数で割って算出します。たとえば、同時実行数の合計が 1000 で、レプリカの数が 10 の場合、平均同時実行数は 100 です。

アプリケーションが Application Load Balancer の背後にある場合は、ロードバランサーの `ActiveConnectionCount` メトリクスをスケーリングメトリクスの要素として使用することもできます。平均値を求めるには、`ActiveConnectionCount` メトリクスをレプリカの数で割る必要があります。スケーリングには未加工のカウント値ではなく、平均値を使用する必要があります。

この設計を最適に機能させるには、リクエストレートが低い場合でも応答レイテンシの標準偏差を小さくする必要があります。需要が少ない時期には、ほとんどのリクエストに短時間で回答し、平均

応答時間よりも大幅に時間がかかるリクエストは多くないことをお勧めします。平均応答時間は 95 パーセンタイル応答時間に近くなります。そうしないと、結果としてキューオーバーフローが発生する可能性があります。これはエラーにつながります。オーバーフローのリスクを軽減するために、必要に応じて追加のレプリカを提供することをお勧めします。

待機中のサーバー

待機中のサーバーはリクエストごとに何らかの処理を行いますが、機能するには 1 つ以上のダウンストリームサービスに大きく依存しています。コンテナアプリケーションは多くの場合、データベースやその他の API サービスなどのダウンストリームサービスを多用します。特に大容量または同時実行の多いシナリオでは、これらのサービスが応答するまでに時間がかかることがあります。これは、これらのアプリケーションが CPU リソースをほとんど使用せず、使用可能なメモリの観点から最大限の同時実行性を利用する傾向があるためです。

待機サービスは、アプリケーションの設計方法に応じて、メモリーバウンドのサーバーパターンとワーカーベースのサーバーパターンのどちらにも適しています。アプリケーションの同時実行がメモリによってのみ制限される場合は、平均メモリ使用率をスケーリングメトリクスとして使用する必要があります。アプリケーションの同時実行がワーカーの制限に基づいている場合は、平均同時実行数をスケーリングメトリクスとして使用する必要があります。

Java ベースのサーバー

Java ベースのサーバーが CPU に依存し、CPU リソースに比例してスケーリングする場合は、効率的な CPU バウンドのサーバーパターンに適している場合があります。その場合は、平均 CPU 使用率がスケーリングメトリクスとして適切である場合があります。ただし、多くの Java アプリケーションは CPU に依存していないため、スケーリングが困難です。

最高のパフォーマンスを得るには、Java 仮想マシン (JVM) ヒープにできるだけ多くのメモリーを割り当てることをお勧めします。Java 8 update 191 以降を含む最近のバージョンの JVM では、コンテナに収まるようにヒープサイズをできるだけ大きく自動的に設定しています。つまり、Java では、メモリー使用率がアプリケーション使用率に比例することはほとんどありません。要求率と同時実行数が増加しても、メモリ使用率は一定に保たれます。このため、メモリー使用率に基づいて Java ベースのサーバーをスケーリングすることはお勧めしません。代わりに、通常は CPU 使用率に基づいてスケーリングすることをおすすめします。

Java ベースのサーバーでは、CPU を使い果たす前にヒープが枯渇することがあります。同時実行数が多いとアプリケーションがヒープを使い果たしやすい場合は、平均接続数が最適なスケーリングメトリクスです。アプリケーションが高スループットでヒープを使い果たしやすい場合は、平均リクエストレートが最適なスケーリングメトリクスです。

ガベージコレクションされた他のランタイムを使用するサーバー

多くのサーバーアプリケーションは、.NET や Ruby などのガベージコレクションを実行するランタイムをベースにしています。これらのサーバーアプリケーションは、前述のパターンのいずれかに当てはまります。ただし、Java の場合と同様に、これらのアプリケーションの平均メモリ使用率はスループットや同時実行性と相関関係がないことが多いため、メモリに基づいてアプリケーションをスケーリングすることはお勧めしません。

このようなアプリケーションでは、アプリケーションが CPU に制約されている場合は CPU 使用率に基づいてスケールすることをおすすめします。それ以外の場合は、負荷テストの結果に基づいて、平均スループットまたは平均同時実行性を基準にスケーリングすることをお勧めします。

ジョブプロセッサ

多くのワークロードには非同期のジョブ処理が含まれます。その中には、リクエストをリアルタイムで受信せず、代わりにワークキューにサブスクライブしてジョブを受け取るアプリケーションが含まれます。この種のアプリケーションでは、ほとんどの場合、適切なスケーリング指標はキューの深さです。キューの増加は、保留中の作業が処理能力を上回っていることを示しているのに対し、キューが空の場合は、実行すべき作業よりも容量が多いことを示します。

AWSAmazon SQS や Amazon Kinesis Data Streams などの メッセージングサービスは、スケーリングに使用できる CloudWatch メトリクスを提供します。Amazon SQS では、`ApproximateNumberOfMessagesVisible` が最適なメトリクスです。Kinesis Data Streams では、Kinesis Client Library (KCL) が公開している `MillisBehindLatest` メトリクスの使用を検討してください。このメトリクスは、スケーリングに使用する前に、すべてのコンシューマで平均化する必要があります。

Amazon ECS のキャパシティとアベイラビリティ

アプリケーションの可用性は、エラーのないエクスペリエンスを提供し、アプリケーションのレイテンシーを最小限に抑えるために不可欠です。可用性を確保するには、アクセス可能で需要を満たすのに十分なキャパシティーを持つリソースが必要です。AWS では、可用性を管理するためのいくつかのメカニズムが用意されています。Amazon ECS でホストされるアプリケーションの場合、自動スケーリングやアベイラビリティーゾーン (AZ) などがあります。自動スケーリングは、定義したメトリクスに基づいてタスクやインスタンスの数を管理します。一方、アベイラビリティーゾーンを使用すると、分離されているが地理的に近い場所でアプリケーションをホストできます。

タスクサイズと同様に、キャパシティーと可用性には考慮する必要がある特定のトレードオフが存在します。理想は、キャパシティーと需要が完全に一致することです。低レイテンシーやエラー率など

のサービスレベル目標 (SLO) を満たすために、リクエストを処理し、ジョブを処理するのに十分なキャパシティーが常に確保されます。キャパシティーが大きすぎて過度のコストがかかったり、キャパシティーが小さすぎてレイテンシーや エラー率が高くなったりすることはありません。

自動スケーリングは潜在的なプロセスです。まず、CloudWatch はリアルタイムメトリクスを受信する必要があります。次に、CloudWatch は、メトリクスを分析するためにこれらを集計する必要があります。メトリクスの詳細度によっては数分かかる場合があります。CloudWatch は、メトリクスをアラームのしきい値と比較して、リソースの不足や超過を特定します。不安定な状態を回避するために、設定されたしきい値を超えた状態が数分間続いたらアラームが作動するようにアラームを設定する必要があります。また、新しいタスクをプロビジョニングしたり、不要になったタスクを終了したりするのにも時間がかかります。

システムでこのような遅延が発生する可能性があるため、オーバープロビジョニングしてある程度の余裕を維持する必要があります。オーバープロビジョニングは、短期的な需要の急増に対応するのに役立ちます。また、アプリケーションを飽和状態に達させずに追加のリクエストを処理するのにも役立ちます。スケーリングのターゲットは、使用率の 60 ~ 80 % に設定することをお勧めします。これにより、追加のキャパシティーがプロビジョニングされている間も、アプリケーションは追加需要の急増をより適切に処理できるようになります。

オーバープロビジョニングが推奨されるもう 1 つの理由は、アベイラビリティーゾーンの障害に迅速に対応できるようになることです。AWS では、本番ワークロードを複数のアベイラビリティーゾーンから提供することをお勧めします。これは、1 つのアベイラビリティーゾーンに障害が発生した場合でも、残りのアベイラビリティーゾーンで実行されているタスクが引き続き需要を処理できるためです。アプリケーションが 2 つのアベイラビリティーゾーンで実行されている場合は、通常のタスク数を 2 倍にする必要があります。これは、潜在的な障害の発生時に即時にキャパシティーを提供できるようにするためです。アプリケーションが 3 つのアベイラビリティーゾーンで実行されている場合は、通常のタスク数の 1.5 倍を実行することをお勧めします。つまり、通常のサービス提供に必要な 2 つのタスクごとに 3 つのタスクを実行します。

スケーリング速度の最大化

自動スケーリングは事後対応型プロセスであるため、効果が現れるまでに時間がかかります。ただし、スケールアウトに必要な時間を最小限に抑えるための方法がいくつかあります。

イメージサイズを最小限に抑えます。イメージが大きいほど、イメージリポジトリからのダウンロードと解凍にかかる時間が長くなります。したがって、イメージサイズを小さくすることで、コンテナの起動に必要な時間が短縮されます。イメージサイズを小さくするには、以下の特定の推奨事項に従ってください。

- 静的バイナリを構築できるか、Golang を使用する場合は、イメージを`FROM`ゼロから構築し、作成されたイメージにバイナリアプリケーションのみを含めます。
- Amazon Linux や Ubuntu など、アップストリームのディストリビューションベンダーが提供する最小限のベースイメージを使用します。
- 最終イメージにはビルドアーティファクトを含めないでください。マルチステージビルドを使用することで、これを実現できます。
- 可能な限り、`RUN` ステージをコンパクトにします。各 `RUN` ステージで新しいイメージレイヤーが作成され、そのレイヤーをダウンロードするための追加のラウンドトリップが発生します。`&&` によって複数のコマンドが結合された単一の `RUN` ステージでは、複数の `RUN` ステージの場合よりレイヤーが少なくなります。
- 最終イメージに ML 推論データなどのデータを含める場合は、起動とトラフィック処理の開始に必要なデータのみを含めます。サービスに影響を与えずに Amazon S3 または他のストレージからオンデマンドでデータを取得する場合は、代わりにそれらの場所にデータを保存します。

イメージを近くに保持します。ネットワークレイテンシーが高いほど、イメージのダウンロードにかかる時間が長くなります。ワークロードと同じ AWS リージョンのリポジトリでイメージをホストします。Amazon ECR は、Amazon ECS を使用可能なすべてのリージョンで使用できる高性能のイメージリポジトリです。インターネットや VPN リンクを経由してコンテナイメージをダウンロードすることは避けてください。同じリージョンでイメージをホストすることで、全体的な信頼性が向上します。これにより、別のリージョンでネットワーク接続の問題や可用性の問題が発生するリスクを軽減できます。または、Amazon ECR クロスリージョンレプリケーションを実装して、これを実現することもできます。

ロードバランサーのヘルスチェックのしきい値を下げます。ロードバランサーは、アプリケーションにトラフィックを送信する前にヘルスチェックを実行します。ターゲットグループのデフォルトのヘルスチェック設定には、90 秒以上かかる場合があります。この間に、ロードバランサーはヘルスステータスをチェックし、リクエストを受信します。ヘルスチェックの間隔としきい値のカウントを下げることで、アプリケーションがトラフィックをより迅速に受け入れ、他のタスクの負荷を減らすことができます。

コールドスタートのパフォーマンスを考慮します。一部のアプリケーションでは、Java などのランタイムを使用してジャストインタイム (JIT) コンパイルを実行します。コンパイルプロセスは、少なくとも開始時に、アプリケーションのパフォーマンスを示すことができます。回避策は、レイテンシーが重要となるワークロードの部分を、コールドスタート時にパフォーマンスの低下を引き起こさない言語で書き換えることです。

ターゲット追跡スケーリングポリシーではなく、ステップスケーリングポリシーを使用します。Amazon ECS タスクには、いくつかの Application Auto Scaling オプションがあります。ターゲットトラッキングは最も使いやすいモードです。これにより、CPU 平均使用率などのメトリクスの目標値を設定するだけです。次に、オートスケーラーは、その値を達成するために必要なタスクの数を自動的に管理します。ステップスケーリングを使用すると、スケーリングメトリクスの特定のしきい値と、しきい値を超えたときに追加または削除するタスクの数を定義できるため、需要の変化に迅速に対応できます。さらに重要なことは、しきい値アラームが超過する時間を最小限に抑えることで、需要の変化に非常に迅速に対応できることです。詳細については、Amazon Elastic Container Service デベロッパーガイドの「サービスのオートスケーリング」を参照してください。

Amazon EC2 インスタンスを使用してクラスターキャパシティーを提供する場合は、以下の推奨事項を考慮してください。

より大きい Amazon EC2 インスタンスと、より高速な Amazon EBS ボリュームを使用します。より大きい Amazon EC2 インスタンスとより高速な Amazon EBS ボリュームを使用することで、イメージのダウンロードと準備の速度を向上させることができます。特定の Amazon EC2 インスタンスファミリー内では、インスタンスサイズが大きくなると、ネットワークと Amazon EBS の最大スループットが増加します (例えば、`m5.xlarge` から `m5.2xlarge`)。さらに、Amazon EBS ボリュームをカスタマイズして、そのスループットと IOPS を向上させることもできます。例えば、gp2 ボリュームを使用する場合は、より大きいボリュームを使用すると、ベースラインスループットが高くなります。gp3 ボリュームを使用する場合は、ボリュームの作成時にスループットと IOPS を指定します。

Amazon EC2 インスタンスで実行されるタスクには、ブリッジネットワークモードを使用します。Amazon EC2 で `bridge` ネットワークモードを使用するタスクは、`awsvpc` ネットワークモードを使用するタスクよりも速く開始されます。`awsvpc` ネットワークモードを使用すると、Amazon ECS はタスクを起動する前に Elastic Network Interface (ENI) をインスタンスにアタッチします。これにより、レイテンシーが大きくなります。ただし、ブリッジネットワークの使用には、いくつかのトレードオフがあります。これらのタスクは独自のセキュリティグループを取得しないため、ロードバランシングにいくつかの影響を及ぼします。詳細については、「Elastic Load Balancing ユーザーガイド」の「Load balancer target groups」を参照してください。

需要ショックへの対処

一部のアプリケーションでは、需要が急激に増大することがあります。これは、ニュースイベント、大セール、メディアイベント、急速に拡散されるその他のイベント (トラフィックが非常に短期間で急速かつ大幅に増加する原因となる) など、さまざまな理由で発生します。計画外の場合、これによって使用可能なリソースを需要が急速に上回る可能性があります。

需要ショックに対処する最善の方法は、それらを予測して適切に計画することです。自動スケーリングには時間がかかる場合があるため、需要ショックが始まる前にアプリケーションをスケールアウトすることをお勧めします。最良の結果を得るために、共有カレンダーを使用するチーム間の緊密なコラボレーションを含むビジネス計画を立てることをお勧めします。イベントを計画しているチームは、事前にアプリケーションを担当するチームと緊密に連携する必要があります。これにより、そのチームは明確なスケジューリング計画を立てるのに十分な時間を確保できます。イベント前にスケールアウトし、イベント後にスケールインするキャパシティーをスケジュールできます。詳細については、「Application Auto Scaling ユーザーガイド」の「スケジュールされたスケーリング」を参照してください。

エンタープライズサポートプランをご利用の場合は、テクニカルアカウントマネージャー (TAM) とも連携してください。TAM はサービスクォータを検証し、イベントの開始前に必要なクォータが引き上げられるようにします。これにより、誤ってサービスクォータに達することがなくなります。また、ロードバランサーなどのサービスを事前準備して、イベントがスムーズに進行するように支援することもできます。

予定外の需要ショックに対処することは、より難しい問題です。予定外のショックが発生し、その振幅が非常に大きい場合、需要がすぐにキャパシティーを超える可能性があります。また、自動スケーリングが反応する能力を上回る可能性もあります。予定外のショックに備える最善の方法は、リソースをオーバープロビジョニングすることです。予想される最大トラフィック需要をいつでも処理できる十分なリソースを確保しておく必要があります。

予定外の需要ショックに備えて最大キャパシティーを維持すると、コストがかかる可能性があります。コストへの影響を軽減するには、大きな需要ショックが差し迫っていることを予測する先行指標となるメトリクスやイベントを見つけます。メトリクスやイベントによって十分な事前通知が確実に提供される場合、イベントが発生したとき、またはメトリクスが設定した特定のしきい値を超えたときには、すぐにスケールアウトプロセスを開始してください。

アプリケーションが突然の予定外の需要ショックを受けやすい場合は、高性能モードをアプリケーションに追加することを検討してください。このモードでは、重要ではない機能を犠牲にして、顧客にとって重要な機能を保持します。例えば、ご使用のアプリケーションでは、コストがかかる高価なカスタマイズされたレスポンスの生成から静的レスポンスページの提供に切り替えることができるとします。このシナリオでは、アプリケーションをまったくスケーリングせずに、スループットを大幅に向上させることができます。

最後に、需要ショックにより適切に対処するために、モノリシックサービスを分割することを検討できます。ご使用のアプリケーションが、実行にコストがかかり、スケーリングに時間がかかるモノリシックサービスである場合、パフォーマンスが重要となる部分を抽出または書き換えて、別のサービスとして実行できる可能性があります。これらの新しいサービスは、重要度の低いコンポーネントか

ら独立してスケーリングできます。パフォーマンスが重要となる機能をアプリケーションの他の部分とは別にスケールアウトする柔軟性を持つことで、キャパシティーの追加にかかる時間の短縮とコストの削減の両方を実現できます。

Amazon ECS クラスターのキャパシティ

Amazon ECS クラスターに容量を提供するには、いくつかの方法があります。例えば、Amazon EC2 インスタンスを起動し、Amazon ECS コンテナエージェントを使用して起動時にクラスターに登録できます。ただし、スケーリングを自分で管理する必要があるため、この方法は難しい場合があります。そのため、Amazon ECS キャパシティプロバイダーを使用することをお勧めします。キャパシティプロバイダーがリソースのスケーリングを管理します。Amazon EC2、Fargate、Fargate Spot の 3 種類のキャパシティプロバイダーがあります。Fargate キャパシティプロバイダーの詳細については、「Amazon ECS clusters for Fargate workloads」を参照し、EC2 ワークロードについては、「Amazon ECS clusters for EC2 workloads」を参照してください。

Fargate および Fargate Spot のキャパシティプロバイダーは、Fargate タスクのライフサイクルをユーザーに代わって処理します。Fargate はオンデマンド容量を提供し、Fargate Spot はスポット容量を提供します。タスクを起動すると、Amazon ECS はユーザーに代わって Fargate リソースをプロビジョニングします。この Fargate リソースには、タスク定義で宣言したタスクレベルの制限に直接対応するメモリと CPU ユニットが付属しています。各タスクは独自の Fargate リソースを受け取り、タスクとコンピューティングリソースの間に 1:1 の関係が作成されます。

Fargate Spot で実行されるタスクは中断される可能性があります。中断は 2 分間の警告の後に発生します。これらは、需要が高い時間帯に発生します。Fargate Spot は、バッチジョブ、開発環境、ステージング環境などの中断に対して耐性のあるワークロードに最適です。また、高いアベイラビリティと低い待機時間が要件ではない他のすべてのシナリオにも適しています。

Fargate Spot タスクは、Fargate オンデマンドタスクと一緒に実行できます。これらを一緒に使用することで、プロビジョンの「バースト」キャパシティを低コストで受け取ることができます。

Amazon ECS では、タスクの Amazon EC2 インスタンス容量を管理することもできます。各 Amazon EC2 キャパシティプロバイダーは、指定した Amazon EC2 Auto Scaling グループに関連付けられます。Amazon EC2 キャパシティプロバイダーを使用する場合、クラスターの自動スケーリングは Amazon EC2 Auto Scaling グループのサイズを維持して、スケジュールされたすべてのタスクを配置できるようにします。

Amazon ECS の Fargate タスクサイズの選択

AWS Fargate タスク定義では、タスクレベルで CPU とメモリを指定する必要があり、オーバーヘッドを考慮する必要はありません。また、コンテナレベルで Fargate タスクの CPU とメモリを指定す

ることもできます。ただし、この設定は必須ではありません。リソース制限は、宣言した予約量以上の値である必要があります。ほとんどの場合、タスク定義で宣言されている各コンテナの予約量の合計に設定できます。次に、その数値を最も近い Fargate サイズに切り上げます。使用可能なサイズの詳細については、「タスク CPU とメモリ」を参照してください。

Amazon EC2 でキャパシティプロバイダーを使用して Amazon ECS クラスターのキャパシティプロビジョニングを高速化する

Amazon EC2 で Amazon ECS を実行するお客様は、Amazon ECS Cluster Auto Scaling (CAS) を利用して Amazon EC2 Auto Scaling グループのスケーリングを管理できます。CAS を使用すると、ASG を自動的にスケールするように Amazon ECS を設定できるため、タスクの実行に集中できます。Amazon ECS により、追加の操作を必要とせずに、必要に応じて ASG をスケールインおよびスケールアウトできるようになります。Amazon ECS キャパシティープロバイダーは、アプリケーションの需要を満たすのに十分なコンテナインスタンスを確保することで、クラスター内のインフラストラクチャを管理するために使用されます。Amazon ECS CAS の仕組みについては、「Amazon ECS クラスターの Auto Scaling を深く探る」を参照してください。

CAS は、クラスターの容量を調節するにあたり、ASG との CloudWatch ベースの統合に依存するため、CloudWatch メトリクスの公開、メトリクス `CapacityProviderReservation` が CloudWatch アラーム (高と低の両方) に違反するまでの時間、および新しく起動された Amazon EC2 インスタンスがウォームアップするまでの時間に関連する固有のレイテンシーが存在します。以下のアクションを実行することで、CAS の応答性を高め、デプロイを高速化できます。

キャパシティープロバイダーのステップスケーリングサイズ

Amazon ECS キャパシティープロバイダーは、最終的に、アプリケーションの需要に合わせてコンテナインスタンスを拡大/縮小します。Amazon ECS が起動するインスタンスの最小数は、デフォルトでは 1 に設定されています。そのため、保留中のタスクを配置するために複数のインスタンスが必要な場合は、デプロイにさらに時間がかかることがあります。Amazon ECS API を使用して `minimumScalingStepSize` を増やすことで、Amazon ECS が一度にスケールインまたはスケールアウトするインスタンスの最小数を増やすことができます。`maximumScalingStepSize` が小さすぎると、一度にスケールインまたはスケールアウトされるコンテナインスタンスの数が制限され、デプロイが遅くなる可能性があります。

> **Note**
>
> この設定は現在、`CreateCapacityProvider` または `UpdateCapacityProvider` API を使用することでのみ利用可能です。

インスタンスのウォームアップ期間

インスタンスのウォームアップ期間は、新たに起動された Amazon EC2 インスタンスが Auto Scaling グループの CloudWatch メトリックスに反映されるまでの時間です。指定されたウォームアップ期間が終了すると、そのインスタンスは ASG の集計メトリクスにカウントされ、CAS は、必要なインスタンス数を推定するための次の計算ループへと進みます。

`instanceWarmupPeriod` のデフォルト値は 300 秒です。この値は `CreateCapacityProvider` API または `UpdateCapacityProvider` API を使用して小さい値に設定することで、スケーリングの応答性を向上させることができます。

予備のキャパシティー

キャパシティープロバイダーにタスクを配置するために使用できるコンテナインスタンスがない場合は、Amazon EC2 インスタンスをその場で起動してクラスターキャパシティーを増やし (スケールアウトし)、それらのインスタンスが起動するまで待ってからコンテナを起動する必要があります。これにより、タスクの起動レートが大幅に低下する可能性があります。ここでは 2 つのオプションがあります。

この場合、予備の Amazon EC2 キャパシティーを事前に起動してタスクを実行する準備をしておくことで、実質的なタスク起動レートを上げることができます。`Target Capacity` 設定を使用して、クラスターで予備のキャパシティーを保持するかどうかを指定できます。例えば、`Target Capacity` を 80 % に設定することで、クラスターに常に 20 % の予備のキャパシティーが必要であることを示します。この予備のキャパシティーにより、スタンドアロンタスクをすぐに起動できるようになり、タスクの起動がスロットリングされなくなります。このアプローチのトレードオフは、予備のクラスターキャパシティーを保持するコストが増加する可能性があることです。

検討できる代替アプローチは、キャパシティープロバイダーではなくサービスにヘッドルームを追加することです。つまり、予備のキャパシティーを起動するための `Target Capacity` 設定を小さくする代わりに、ターゲット追跡スケーリングメトリクス、またはサービス自動スケーリングのステップスケーリングのしきい値を変更することで、サービス内のレプリカの数を増やすことができます。このアプローチが有用なのはワークロードの急増に対してのみであり、新しいサービスをデプロイし、初めて 0 から N タスクに移行する場合には効果がないことに注意してください。関連するスケーリングポリシーの詳細については、「Target Tracking Scaling Policies」または「Step Scaling Policies」を参照してください。

アカウント設定による Amazon ECS 機能へのアクセス

Amazon ECS アカウント設定に移動して、特定の機能をオプトインまたはオプトアウトできます。各 AWS リージョンについて、アカウントレベルまたは特定のユーザーまたはロールで、各アカウント設定をオプトインまたはオプトアウトすることができます。

次のいずれかの条件が関係する場合は、特定の機能をオプトインまたはオプトアウトすることが考えられます。

- ユーザーまたはロールは、個々のアカウントのために特定のアカウント設定をオプトインまたはオプトアウトできます。
- ユーザーまたはロールはアカウントのすべてのユーザーに対してデフォルトのオプトインまたはオプトアウト設定を定義できます。
- ルートユーザーまたは管理者権限を持つユーザーは、アカウントの任意の種類のロールまたはユーザーについて、オプトインまたはオプトアウトすることができます。ルートユーザーのアカウント設定が変更されると、個別にアカウント設定が選択されなかったすべてのユーザーとロールに対してデフォルト値が設定されます。

> **Note**
>
> フェデレーションユーザーは、ルートユーザーのアカウント設定を想定しており、明示的にアカウント設定を個別に設定することはできません。

以下のアカウント設定を使用できます。アカウント設定ごとに、オプトインおよびオプトアウトする必要があります。

リソース名	詳細情報
`containerInsights`	Container Insights

`serviceLongArnFormat`
`taskLongArnFormat`
`containerInstanceLongArnFormat`
	Amazon リソースネーム (ARN) と ID

`tagResourceAuthorization`	タグ付け認可

リソース名	詳細情報
fargateFIPSMode	AWS Fargate 連邦情報処理標準 (FIPS-140) コンプライアンス
fargateTaskRetirementWaitPeriod	AWS Fargate タスク廃止の待機時間
fargateEventWindows	AWS FargateEC2 イベントウィンドウを使用した タスクの廃止
guardDutyActivate	Runtime Monitoring (Amazon GuardDuty 統合)
dualStackIPv6	デュアルスタック IPv6 VPC
awsvpcTrunking	Linux コンテナインスタンスのネットワークインターフェイスを増やす
defaultLogDriverMode	デフォルトのログドライバーモード

Amazon リソースネーム (ARN) と ID

Amazon ECS リソースの作成時に、各リソースには一意の Amazon リソースネーム (ARN) と一意のリソース識別子 (ID) が割り当てられます。コマンドラインツールまたは Amazon ECS API を使用して Amazon ECS を操作する場合、特定のコマンドにはリソース ARNまたはリソース ID が必要になります。例えば、タスクを停止するために stop-task AWS CLI コマンドを使用する場合は、そのコマンドでタスクの ARN または ID を指定する必要があります。

Amazon ECS では、Amazon ECS サービス、タスク、コンテナインスタンスの Amazon リソース名 (ARN) とリソース ID の新しい形式が導入されました。各リソースタイプのオプトインステータスによって、リソースが使用する Amazon リソースネーム (ARN) の形式が決まります。そのリソースタイプのリソースタグ付けなどの機能を使用するには、新しい ARN の形式にオプトインする必要があります。

新しい Amazon リソースネーム (ARN) およびリソース ID の形式は、リージョンごとにオプトインおよびオプトアウトできます。現在、新しく作成されたアカウントはデフォルトでオプトインされています。

新形式の Amazon リソースネーム (ARN) およびリソース ID はいつでもオプトインまたはオプトアウトできます。オプトイン後、作成するすべての新しいリソースは新しい形式を使用します。

> **Note**
>
> リソース ID が作成後に変更することはありません。したがって、新しい形式をオプトインまたはオプトアウトしても、既存のリソース ID には影響しません。

以下のセクションでは、ARN およびリソース ID 形式がどのように変更されているかについて説明します。新しい形式への移行の詳細については、「Amazon Elastic Container Service のよくある質問」を参照してください。

Amazon リソースネーム (ARN) 形式

一部のリソースには、production という名前のサービスなど、わかりやすい名前があります。それ以外の場合は、Amazon リソースネーム (ARN) 形式を使用してリソースを指定する必要があります。Amazon ECS のタスク、サービス、およびコンテナインスタンスの新しい ARN 形式には、クラスター名が含まれます。新しい ARN 形式へのオプトインの詳細については、「Amazon ECS アカウント設定の変更」を参照してください。

各リソースタイプでの現在の形式と新しい形式を次の表に示しています。

リソースタイプ	ARN
コンテナインスタンス	現在: arn:aws:ecs: *region*:*aws_account_id* :container-instance/ *container-instance-id* 新: arn:aws:ecs: *region*:*aws_account_id* :container-instance/ *cluster-name* /*container-instance-id*
Amazon ECS サービス	現在: arn:aws:ecs: *region*:*aws_account_id* :service/ *service-name* 新: arn:aws:ecs: *region*:*aws_account_id* :service/ *cluster-name* /*service-name*
Amazon ECS タスク	現在: arn:aws:ecs: *region*:*aws_account_id* :task/*task-id* 新: arn:aws:ecs: *region*:*aws_account_id* :task/*cluster-name* /*task-id*

リソース ID の長さ

リソース ID は文字と数字の一意の組み合わせです。新しいリソース ID 形式には、Amazon ECS タスクおよびコンテナインスタンス用の短い ID が含まれます。現在のリソース ID の形式の長さは 36 文字です。新しい ID は 32 文字の形式で、ハイフンは含まれません。新しい リソース ID 形式をオプトインする方法の詳細については、「Amazon ECS アカウント設定の変更」を参照してください。

デフォルトは `enabled` です。

オプトインした後に起動されたリソースだけが、新形式の ARN とリソース ID を受け取ります。既存のリソースは、いずれも影響を受けません。Amazon ECS サービスとタスクを新しい ARN およびリソース ID の形式に移行するには、サービスまたはタスクを再作成する必要があります。コンテナインスタンスを新しい ARN およびリソース ID 形式に移行するには、そのコンテナインスタンスを空にし、新規コンテナインスタンスを起動してクラスターに登録する必要があります。

> **Note**
>
> Amazon ECS サービスが 2018 年 11 月 16 日以降に作成されていて、そのサービスを作成したユーザーがタスクの新形式をオプトインしている場合、そのサービスによって起動されたタスクだけが新形式の ARN とリソース ID になります。

ARN およびリソース ID 形式のタイムライン

新しい Amazon リソースネーム (ARN) と Amazon ECS リソースのリソース ID 形式のための、オプトインおよびオプトアウト期間のタイムラインは、2021 年 4 月 1 日に終了しました。デフォルトでは、すべてのアカウントが新しい形式にオプトインされています。新しく作成されたすべてのリソースには新しい形式が適用され、オプトアウトできなくなります。

Container Insights

2024 年 12 月 2 日、AWS で Amazon ECS 用にオブザーバビリティが強化された Container Insights がリリースされました。このバージョンでは、Fargate Amazon ECS マネージドインスタンスと EC2 を使用して Amazon ECS クラスター用に強化されたオブザーバビリティがサポートされます。Amazon ECS でオブザーバビリティが強化された Container Insights を設定したら、Container Insights は環境内のクラスターレベルからコンテナレベルまでの詳細なインフラストラクチャテレメトリを自動収集し、さまざまなメトリクスとディメンションを示すダッシュボードにデータを表示します。その後、Container Insights コンソールでこれらのすぐに使えるダッシュボードを使用して、

コンテナの健全性とパフォーマンスをよりよく理解し、異常を特定することで問題を迅速に軽減することができます。

コンテナ環境で詳細な可視性を提供し、解決までの平均時間を短縮するため、Container Insights ではなく、オブザーバビリティが強化された Container Insights を使用することをお勧めします。詳細については、「Amazon CloudWatchユーザーガイド」の「Amazon ECS 用にオブザーバビリティメトリクスが強化された Container Insights」を参照してください。

`containerInsights` アカウントのデフォルトは、`disabled` です。

オブザーバビリティが強化された Container Insights

次のコマンドを使用し、オブザーバビリティが強化された Container Insights を有効にします。

`containerInsights` アカウント設定を `enhanced` に設定します。

```
aws ecs put-account-setting --name containerInsights --value enhanced
```

出力の例

```
{
    "setting": {
        "name": "containerInsights",
        "value": "enhanced",
        "principalArn": "arn:aws:iam::123456789012:johndoe",
         "type": user
    }
}
```

このアカウント設定を行ったら、すべての新しいクラスターはオブザーバビリティが強化された Container Insights を自動的に使用します。`update-cluster-settings` コマンドを使用してオブザーバビリティが強化された Container Insights を既存のクラスターに追加するか、Container InsightsからオブザーバビリティがEnhanced Container Insightsにクラスターをアップグレードします。

```
aws ecs update-cluster-settings --cluster cluster-name --settings
 name=containerInsights,value=enhanced
```

コンソールを使用して、オブザーバビリティが強化された Container Insights を設定することもできます。詳細については、「Amazon ECS アカウント設定の変更」を参照してください。

Container Insights

containerInsights アカウント設定を enabled に設定すると、すべての新しいクラスターはデフォルトで Container Insights が有効になります。update-cluster-settings を使用して既存のクラスターを変更できます。

Container Insights を使用するには、containerInsights アカウント設定を enabled に設定します。次のコマンドを使用して Container Insights を有効にします。

```
aws ecs put-account-setting --name containerInsights --value enabled
```

出力の例

```
{
    "setting": {
        "name": "containerInsights",
        "value": "enabled",
        "principalArn": "arn:aws:iam::123456789012:johndoe",
         "type": user
    }
}
```

containerInsights アカウント設定を enabled に設定すると、すべての新しいクラスターはデフォルトで Container Insights が有効になります。update-cluster-settings コマンドを使用して Container Insights を既存のクラスターに追加します。

```
aws ecs update-cluster-settings --cluster cluster-name --settings
 name=containerInsights,value=enabled
```

コンソールを使用して Container Insights を設定することもできます。詳細については、「Amazon ECS アカウント設定の変更」を参照してください。

AWS Fargate 連邦情報処理標準 (FIPS-140) コンプライアンス

Fargate は、機密情報を保護する暗号モジュールのセキュリティ要件を指定する連邦情報処理標準 (FIPS-140) をサポートしています。これは現在の米国およびカナダの政府標準であり、Federal Information Security Management Act (FISMA) または Federal Risk and Authorization Management Program (FedRAMP) への準拠が要求されるシステムに適用されます。

リソース名は fargateFIPSMode です。

デフォルトは disabled です。

Fargate で連邦情報処理標準 (FIPS-140) コンプライアンスをオンにする必要があります。詳細については、「the section called “AWS Fargate FIPS-140 コンプライアンス”」を参照してください。

> **Important**
>
> fargateFIPSMode アカウント設定は、Amazon ECS API または AWS CLI を使用してのみ変更できます。詳細については、「Amazon ECS アカウント設定の変更」を参照してください。

fargateFIPSMode オプションを enabled に設定して put-account-setting-default を実行します。詳細については、「Amazon Elastic Container Service API リファレンス」の「put-account-setting-default」を参照してください。

- 次のコマンドを使用して、FIPS-140 コンプライアンスをオンにできます。

```
aws ecs put-account-setting-default --name fargateFIPSMode --value enabled
```

出力の例

```
{
    "setting": {
        "name": "fargateFIPSMode",
        "value": "enabled",
        "principalArn": "arn:aws:iam::123456789012:root",
         "type": user
    }
}
```

list-account-settings を実行して、現在の FIPS-140 コンプライアンスステータスを表示できます。effective-settings オプションを使用して、アカウントレベルの設定を表示します。

```
aws ecs list-account-settings --effective-settings
```

タグ付け認可

Amazon ECS は、リソース作成のためのタグ付け認可を導入しています。ecsCreateCluster などのリソースを作成するアクションには、ユーザーがタグ付けのアクセス許可を持っている必要があります。リソースを作成し、そのリソースのタグを指定すると、AWS は、追加の認可を実行して、タグを作成するための許可があることを検証します。したがって、ecs:TagResource アクションを使用するための明示的な許可を付与する必要があります。詳細については、「the section called “リソース作成時のタグ付け”」を参照してください。

タグ付け認可にオプトインするには、tagResourceAuthorization オプションを on に設定して put-account-setting-default を実行します。詳細については、「Amazon Elastic Container Service API リファレンス」の「put-account-setting-default」を参照してください。list-account-settings を実行して、現在のタグ付け認可ステータスを表示できます。

- 次のコマンドを使用して、タグ付け認可を有効にできます。

```
aws ecs put-account-setting-default --name tagResourceAuthorization --value on --
region region
```

出力の例

```
{
    "setting": {
        "name": "tagResourceAuthorization",
        "value": "on",
        "principalArn": "arn:aws:iam::123456789012:root",
        "type": user
    }
}
```

タグ付け認可を有効にしたら、作成時にユーザーがリソースにタグ付けできるように、適切なアクセス許可を設定する必要があります。詳細については、「the section called “リソース作成時のタグ付け”」を参照してください。

list-account-settings を実行して、現在のタグ付け認可ステータスを表示できます。effective-settings オプションを使用して、アカウントレベルの設定を表示します。

```
aws ecs list-account-settings --effective-settings
```

タグ付け認可のタイムライン

`list-account-settings` を実行して `tagResourceAuthorization` 値を表示することで、タグ付け認可がアクティブかどうかを確認できます。値が on である場合、タグ付け認可が使用されています。詳細については、「Amazon Elastic Container Service API リファレンス」の「list-account-settings」を参照してください。

タグ付け認可に関連する重要な日付を次に示します。

- 2023 年 4 月 18 日 – タグ付け認可が導入されました。この機能を使用するには、新規および既存のすべてのアカウントでオプトインする必要があります。タグ付け認可の使用開始をオプトインできます。オプトインすることにより、適切な許可を付与する必要があります。
- 2024 年 2 月 9 日から 2024 年 3 月 6 日 — すべての新しいアカウントと影響を受けていない既存のアカウントでは、タグ付け認可がデフォルトで有効になります。`tagResourceAuthorization` アカウント設定を有効/無効にすると、IAM ポリシーを確認できます。

  AWS は、影響を受けたアカウントに通知しました。

  この機能を無効にするには、`tagResourceAuthorization` オプションを `off` に設定して `put-account-setting-default` を実行します。
- 2024 年 3 月 7 日 — タグ付け認可を有効にしている場合、アカウント設定を無効にすることはできなくなります。

  この日までに IAM ポリシーのテストを完了することをお勧めします。
- 2024 年 3 月 29 日 — すべてのアカウントがタグ付け認可を使用するようになります。アカウントレベルの設定は、Amazon ECS コンソールまたは AWS CLI では使用できなくなります。

AWS Fargate タスク廃止の待機時間

AWS廃止の対象としてマークされたプラットフォームバージョンリビジョン上で実行している Fargate タスクがある場合、 から通知が送信されます。詳細については、「AWS Fargate on Amazon ECS のタスクの廃止とメンテナンス」を参照してください。

AWS Fargate の基盤となるインフラストラクチャに対する、パッチ適用とメンテナンスは、AWS が担当します。Fargate でホストされている Amazon ECS タスクにおいて、セキュリティまたはインフラストラクチャの更新が必要であると AWS が判断した場合、対象のタスクを停止し、それらを置

き換えるための新しいタスクを起動する必要があります。Fargate タスクが廃止されるまでの待機時間を設定できます。タスクをすぐに廃止するか、7 暦日または 14 暦日待つかを選択できます。

この設定はアカウントレベルで適用されます。

Fargate がタスク廃止を開始する時間を設定できます。デフォルトの待機時間は 7 日間です。アップデートをすぐに適用する必要があるワークロードの場合は、即時設定 (0) を選択します。これより長い時間が必要な場合は、7 日または 14 日オプションを設定します。

新しいプラットフォームバージョンのリビジョンを早く取得できるように、待機時間を短くすることをお勧めします。

待機時間を設定するには、ルートユーザーまたは管理者ユーザーで `put-account-setting-default` または `put-account-setting` を実行します。`fargateTaskRetirementWaitPeriod` オプションを `name` に使用し、`value` オプションを次のいずれかの値に設定します。

- `0` - AWS から通知が送信され、影響を受けたタスクが直ちに廃止されます。
- 7 - AWS から通知が送信され、7 日間待機してから、影響を受けたタスクの廃止が開始されます。これがデフォルトです。
- 14 - AWS から通知が送信され、14 日間待機してから、影響を受けたタスクの廃止が開始されます。

詳細については、「Amazon Elastic Container Service API リファレンス」の「put-account-setting-default」と「put-account-setting」を参照してください。

次のコマンドを実行して、待機時間を 14 日間に設定できます。

```
aws ecs put-account-setting-default --name fargateTaskRetirementWaitPeriod --value 14
```

出力の例

```
{
    "setting": {
        "name": "fargateTaskRetirementWaitPeriod",
        "value": "14",
        "principalArn": "arn:aws:iam::123456789012:root",
        "type: user"
    }
```

```
}
```

現在の Fargate タスク廃止の待機時間は、`list-account-settings` を実行して表示できます。`effective-settings` オプションを使用する

```
aws ecs list-account-settings --effective-settings
```

AWS FargateEC2 イベントウィンドウを使用した タスクの廃止

AWS の基盤インフラストラクチャに対するパッチの適用とメンテナンスは の責任です。AWS FargateFargate でホストされている Amazon ECS タスクにおいて、セキュリティまたはインフラストラクチャの更新が必要であると AWS が判断した場合、対象のタスクを停止し、それらを置き換えるための新しいタスクを起動する必要があります。このアカウント設定を有効にすると、AWS Fargate が EC2 イベントウィンドウを使用してタスクの廃止日時を判断します。これはアカウントの 1 回限りの有効化であることに注意してください。

Fargate タスクの廃止に対して EC2 イベントウィンドウの使用を有効にするには、次の CLI コマンドを使用できます。

```
aws ecs put-account-setting-default --name fargateEventWindows --value enabled
```

これは、アカウントレベルの設定です。次のインスタンスタグを使用して、EC2 イベントウィンドウをアカウント、クラスター、サービスレベルで Fargate タスクに関連付けることができます。

- `aws:ecs:serviceArn` : サービスタスク向け
- `aws:ecs:clusterArn` : クラスターに属するタスク向け
- `aws:ecs:fargateTask` : アカウント内のすべての Fargate タスクを対象にするために true に設定

Fargate で実行されている Amazon ECS タスク向けの Amazon EC2 イベントウィンドウの機能方法に関する詳細については、「ステップ 1: タスクの待機時間を設定、または Amazon EC2 イベントウィンドウを使用する」を参照してください。

Linux コンテナインスタンスのネットワークインターフェイスを増やす

awsvpc ネットワークモードを使用する各 Amazon ECS タスクには、独自の Elastic Network Interface (ENI) が割り当てられ、その ENI はそれをホストするコンテナインスタンスにアタッチされ

ます。Amazon EC2 インスタンスにアタッチできるネットワークインターフェイスの数にはデフォルトの制限があり、プライマリネットワークインターフェイスも 1 つとしてカウントされます。例えば、デフォルトでは c5.large インスタンスには最大 3 つの ENI がアタッチされています。このインスタンスのプライマリネットワークインターフェイスも 1 つとしてカウントされるため、このインスタンスに追加でアタッチできる ENI は 2 つです。awsvpc ネットワークモードを使用する各タスクには ENI が必要なため、通常このインスタンスタイプでは、このようなタスクを 2 つだけ実行できます。

Amazon ECS は、サポートされている Amazon EC2 インスタンスタイプを使用して、ENI 密度が高いコンテナインスタンスの起動をサポートしています。これらのインスタンスタイプを使用し、awsvpcTrunking アカウント設定を有効にすると、新しく起動されたコンテナインスタンスで追加の ENI を利用できます。この設定により、各コンテナインスタンスにより多くのタスクを配置できます。

例えば、awsvpcTrunking を持つ c5.large インスタンスでは、ENI の制限が 12 に引き上げられています。コンテナインスタンスはプライマリネットワークインターフェイスを持ち、Amazon ECS はコンテナインスタンスの「トランク」ネットワークインターフェイスを作成およびアタッチします。したがって、この設定では、現在の 2 個ではなく 10 個のタスクをコンテナインスタンスで起動できます。

Runtime Monitoring (Amazon GuardDuty 統合)

Runtime Monitoring は、AWS ログとネットワークアクティビティを継続的に監視して悪意のある動作や不正な動作を特定することで、Fargate および EC2 で実行されているワークロードを保護するインテリジェントな脅威検出サービスです。

guardDutyActivate パラメータは Amazon ECS では読み取り専用で、Amazon ECS アカウントのセキュリティ管理者によってランタイムモニタリングが有効になっているか無効になっているかを示します。GuardDuty は、ユーザーに代わってこのアカウント設定を制御します。詳細については、「Runtime Monitoring で Amazon ECS ワークロードを保護する」を参照してください。

list-account-settings を実行して、現在の GuardDuty 統合設定を表示できます。

```
aws ecs list-account-settings
```

出力の例

```
{
```

```
    "setting": {
        "name": "guardDutyActivate",
        "value": "on",
        "principalArn": "arn:aws:iam::123456789012:doej",
        "type": aws-managed"
    }
}
```

デュアルスタック IPv6 VPC

Amazon ECS は、プライマリプライベート IPv4 アドレスに加えて IPv6 アドレスを備えたタスクの提供をサポートします。

タスクが IPv6 アドレスを受信するには、タスクが awsvpc ネットワークモードを使用し、デュアルスタックモードに設定された VPC で起動し、dualStackIPv6 アカウント設定を有効にする必要があります。その他の要件の詳細については、EC2 のキャパシティについては「デュアルスタックモードでの VPC の使用」、Amazon ECS マネージドインスタンスのキャパシティについては「デュアルスタックモードでの VPC の使用」、Fargate のキャパシティについては「デュアルスタックモードでの VPC の使用」を参照してください。

> **Important**
>
> dualStackIPv6 アカウント設定は、Amazon ECS API または AWS CLI を使用してのみ変更できます。詳細については、「Amazon ECS アカウント設定の変更」を参照してください。

2020 年 10 月 1 日から 2020 年 11 月 2 日の間に、IPv6 が有効なサブネットで awsvpc ネットワークモードを使用してタスクを実行している場合、タスクが実行されていたリージョンの既定の dualStackIPv6 アカウント設定は disabled です。この条件が満たされない場合、リージョンのデフォルト dualStackIPv6 設定は enabled です。

デフォルトは disabled です。

デフォルトのログドライバーモード

Amazon ECS では、コンテナから選択したログドライバーへのログメッセージのデフォルトの配信モードの設定がサポートされています。配信モードは、コンテナからログドライバーへのログのフローが中断されたときのアプリケーションの安定性に影響します。

defaultLogDriverMode 設定では、blocking と non-blocking の 2 つの値がサポートされています。これらの配信モードの詳細については、「Amazon Elastic Containers サービス API リファレンス」の「LogConfiguration」を参照してください。

コンテナ定義の logConfiguration で配信モードを指定しない場合、このアカウント設定を使用して指定したモードがデフォルトとして使用されます。

デフォルトの配信モードは non-blocking です。

> **Note**
>
> 2025 年 6 月 25 日、Amazon ECS はデフォルトのログドライバーモードが blocking から non-blocking に変更され、ログ記録よりもタスクの可用性が優先されるようになりました。この変更以降も blocking モードを引き続き使用するには、次のいずれかを実行してください。
>
> - コンテナ定義の logConfiguration の mode オプションを blocking に設定します。
> - defaultLogDriverMode アカウント設定を blocking に設定します。

デフォルトのログドライバーモードを blocking に設定するには、次のコマンドを実行します。

```
aws ecs put-account-setting-default --name defaultLogDriverMode --value "blocking"
```

コンソールを使用して Amazon ECS のアカウント設定を表示する

コンソールでアカウント設定を表示して、アクセスできる機能を確認します。

> **Important**
>
> dualStackIPv6、fargateFIPSMode、fargateTaskRetirementWaitPeriod、および fargateEventWindows アカウントの設定を表示または変更するために使用できるのは AWS CLI のみです。

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. 上部にあるナビゲーションバーで、アカウント設定を表示するリージョンを選択します。
3. ナビゲーションペインで [Account Settings] (アカウント設定) を選択します。

Amazon ECS アカウント設定の変更

アカウント設定を変更して Amazon ECS 機能にアクセスします。

`guardDutyActivate` パラメータは Amazon ECS では読み取り専用で、Amazon ECS アカウントのセキュリティ管理者によってランタイムモニタリングが有効になっているか無効になっているかを示します。GuardDuty は、ユーザーに代わってこのアカウント設定を制御します。詳細については、「Runtime Monitoring で Amazon ECS ワークロードを保護する」を参照してください。

> **Important**
>
> `dualStackIPv6`、`fargateFIPSMode`、`fargateTaskRetirementWaitPeriod`、および `fargateEventWindows` アカウントの設定を表示または変更するために使用できるのは AWS CLI のみです。

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. 上部にあるナビゲーションバーで、アカウント設定を表示するリージョンを選択します。
3. ナビゲーションペインで [Account Settings] (アカウント設定) を選択します。
4. [更新] を選択します。
5. (オプション) 各 EC2 インスタンスの awsvpc ネットワークモードで実行できるタスク数を増減するには、[AWSVPC トランキング] で [AWSVPC トランキング] を選択します。
6. (オプション) クラスターで CloudWatch Container Insights をデフォルトで使用、または使用を停止するには、CloudWatch Container Insights オブザーバビリティ で、次のいずれかのオプションを選択します。
   - オブザーバビリティが強化された Container Insights を使用するには、[オブザーバビリティが強化された Container Insights] を選択します。
   - Container Insights を使用するには、[Container Insights] を選択します。
   - Container Insights の使用を停止するには、[オフ]を選択します。
7. (オプション) タグ付け認可を有効または無効にするには、[リソースのタグ付け認可] で、[リソースのタグ付け認可] を選択または選択解除します。
8. (オプション) コンテナの `logConfiguration` でログ配信モードが定義されていない場合にデフォルトのログドライバーモードを設定するには、デフォルトのログドライバーモードで、次のいずれかのオプションを選択します。

- デフォルトのログドライバーモードを `blocking` に設定するには、[ブロッキング] を選択します。
- デフォルトのログドライバーモードを `non-blocking` に設定するには、[ノンブロッキング] を選択します。

9. [Save changes] (変更の保存) をクリックします。
10. 確認画面で [Confirm (確認)] を選択すると、選択内容が保存されます。

次のステップ

オブザーバビリティが強化された Container Insights または Container Insights を有効にした場合、オプションで既存のクラスターを更新してその機能を使用できます。詳細については、「Amazon ECS クラスターを更新する」を参照してください。

デフォルトの Amazon ECS アカウント設定の復元

AWS マネジメントコンソール を使用して、Amazon ECS アカウント設定をデフォルトに戻せます。

[Revert to account default] (アカウントのデフォルトに戻す) オプションは、アカウント設定がデフォルト設定ではなくなった場合にのみ使用できます。

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. 上部にあるナビゲーションバーで、アカウント設定を表示するリージョンを選択します。
3. ナビゲーションペインで [Account Settings] (アカウント設定) を選択します。
4. [更新] を選択します。
5. [Revert to account default] (アカウントのデフォルトに戻す) を選択します。
6. 確認画面で [Confirm (確認)] を選択すると、選択内容が保存されます。

AWS CLI を使用した Amazon ECS アカウント設定の管理

アカウント設定は、Amazon ECS API、AWS CLI、または SDK を使用して管理することができます。`dualStackIPv6`、`fargateFIPSMode`、`fargateTaskRetirementWaitPeriod`、および `fargateEventWindows` アカウントの設定を表示または変更するために使用できるのは、これらのツールのみです。

> **Note**
>
> デュアルスタックサービスエンドポイントを使用することで、IPv4 と IPv6 の両方を介して AWS CLI、SDK、および Amazon ECS API から Amazon ECS とやり取りできます。詳細については、「Amazon ECS デュアルスタックエンドポイントの使用」を参照してください。

タスク定義に使用できる API アクションの詳細については、「Amazon Elastic Container Service API リファレンス」の「アカウント設定アクション」を参照してください。

アカウントのすべてのユーザーまたはロールのデフォルトアカウント設定を変更するには、以下のいずれかのコマンドを使用します。これらの変更は、ユーザーまたはロールがこれらの設定を明示的に上書きしない限り、AWS アカウント全体に適用されます。

- put-account-setting-default (AWS CLI)

```
aws ecs put-account-setting-default --name serviceLongArnFormat --value enabled --
region us-east-2
```

  このコマンドを使用して、他のアカウント設定を変更することもできます。そのためには、name パラメータを対応するアカウント設定に置き換えます。

- Write-ECSAccountSetting (AWS Tools for Windows PowerShell)

```
Write-ECSAccountSettingDefault -Name serviceLongArnFormat -Value enabled -Region us-
east-1 -Force
```

ユーザーアカウントのアカウント設定を変更するには (AWS CLI)

自分のユーザーのアカウント設定を変更するには、以下のいずれかのコマンドを使用します。ルートユーザーとしてこれらのコマンドを使用する場合、ユーザーまたはロールが対象の設定を明示的に上書きしない限り、この変更内容が AWS アカウント全体に適用されます。

- put-account-setting (AWS CLI)

```
aws ecs put-account-setting --name serviceLongArnFormat --value enabled --region us-
east-1
```

このコマンドを使用して、他のアカウント設定を変更することもできます。そのためには、name パラメータを対応するアカウント設定に置き換えます。

- Write-ECSAccountSetting (AWS Tools for Windows PowerShell)

```
Write-ECSAccountSetting -Name serviceLongArnFormat -Value enabled -Force
```

特定のユーザーまたはロールのアカウント設定を変更するには (AWS CLI)

特定のユーザーまたはロールのアカウント設定を変更するには、以下のいずれかのコマンドを使用して、リクエストの中でユーザー、ロール、またはルートユーザーの ARN を指定します。

- put-account-setting (AWS CLI)

```
aws ecs put-account-setting --name serviceLongArnFormat --value enabled --principal-
arn arn:aws:iam::aws_account_id:user/principalName --region us-east-1
```

このコマンドを使用して、他のアカウント設定を変更することもできます。そのためには、name パラメータを対応するアカウント設定に置き換えます。

- Write-ECSAccountSetting (AWS Tools for Windows PowerShell)

```
Write-ECSAccountSetting -Name serviceLongArnFormat -Value enabled -PrincipalArn
 arn:aws:iam::aws_account_id:user/principalName -Region us-east-1 -Force
```

Amazon ECS の IAM ロール

IAM ロールは、特定の許可があり、アカウントで作成できるもう 1 つの IAM アイデンティティです。Amazon ECS では、コンテナやサービスなどの Amazon ECS リソースにアクセス許可を付与するロールを作成できます。

Amazon ECS が必要とするロールは、タスク定義の起動タイプと使用する機能によって異なります。次の表を参照して、Amazon ECS に必要な IAM ロールを決定します。

ロール	定義	必要な場合	詳細情報
タスク実行ロール	このロールにより、Amazon ECS がお	タスクは AWS Fargate または外部イ	Amazon ECS タスク実行IAM ロール

ロール	定義	必要な場合	詳細情報
	客様に代わって他の AWS サービスを利用できるようにします。	ンスタンスホストされています。また、 • Amazon ECR プライベートリポジトリからコンテナイメージをプルします。 • タスクを実行するアカウントとは別のアカウントの Amazon ECR プライベートリポジトリからコンテナイメージをプルします。 • `awslogs` ログドライバーを使用して CloudWatch Logs にコンテナログを送信します。 タスクは、AWS Fargate または Amazon EC2 インスタンスでホストされています。また、 • プライベートレジストリの認証を使用します。 • Runtime Monitoring を使用します。	

ロール	定義	必要な場合	詳細情報
		• タスク定義は、Secrets Manager のシークレットまたは AWS Systems Manager Parameter Store のパラメータを使用して機密データを参照します。	
タスクロール	このロールにより、(コンテナ上の) アプリケーションコードが他の AWS サービスを使用できるようになります。	アプリケーションは、Amazon S3 などの他の AWS サービスにアクセスします。	Amazon ECS タスクの IAM ロール
コンテナインスタンスのロール	このロールにより、EC2 インスタンスまたは外部インスタンスをクラスターに登録できます。	タスクは Amazon EC2 インスタンスまたは外部インスタンスでホストされています。	Amazon ECS コンテナインスタンスの IAM ロール
Amazon ECS Anywhere ロール	このロールにより、外部インスタンスが AWS API にアクセスできるようになります。	タスクは外部インスタンスでホストされています。	Amazon ECS Anywhere IAM ロール

ロール	定義	必要な場合	詳細情報
ロードバランサーロール用の Amazon ECS インフラストラクチャ	このロールにより、Amazon ECS はブルー/グリーンデプロイにおいて、ユーザーに代わってクラスター内のロードバランサーリソースを管理できます。	Amazon ECS のブルー/グリーンデプロイを使用できます。	ロードバランサー用の Amazon ECS インフラストラクチャの IAM ロール
Amazon ECS CodeDeploy ロール	このロールにより、CodeDeploy はサービスを更新できます。	CodeDeploy のブルー/グリーンデプロイタイプを使用して、サービスをデプロイします。	Amazon ECS CodeDeploy IAM ロール
Amazon ECS EventBridge ロール	このロールにより、EventBridge はサービスを更新できます。	EventBridge のルールとターゲットを使用してタスクをスケジュールします。	Amazon ECS EventBridge IAM ロール

ロール	定義	必要な場合	詳細情報
Amazon ECS インフラストラクチャロール	このロールにより、Amazon ECS はクラスター内のインフラストラクチャリソースを管理できます。	• 容量に Amazon ECS マネージドインスタンスを使用します。 • Amazon EBS ボリュームを Fargate または EC2 起動タイプの Amazon ECS タスクにアタッチできます。インフラストラクチャロールにより、Amazon ECS はタスクの Amazon EBS ボリュームを管理できます。 • Amazon ECS Service Connect サービス間のトラフィックを暗号化するには、Transport Layer Security (TLS) を使用します。 • VPC Lattice ターゲットグループを作成する場合。	Amazon ECS インフラストラクチャ IAM ロール

ロール	定義	必要な場合	詳細情報
インスタンスプロファイル	このロールにより、Amazon ECS マネージドインスタンスはインフラストラクチャロールを安全に引き受けることができます。	クラスターで Amazon ECS マネージドインスタンスを使用します。	Amazon ECS マネージドインスタンスのインスタンスプロファイル

Amazon ECS マネージドインスタンスの設計

Amazon ECS マネージドインスタンスは、Amazon ECS の完全マネージド型のコンピューティングオプションであり、インフラストラクチャ管理を AWS で負荷軽減しながら、コンテナ化されたワークロードをすべての Amazon EC2 インスタンスタイプで実行できます。Amazon ECS マネージドインスタンスでは、AWS が基盤となるインフラストラクチャの割り当て、スケーリング、パッチ適用、メンテナンスを処理している間に、GPU アクセラレーション、特定の CPU アーキテクチャ、高ネットワークパフォーマンス、および特殊なインスタンスタイプなどの特定のコンピューティング機能にアクセスできます。

Amazon ECS マネージドインスタンスを使用する場合、アプリケーションをコンテナにパッケージ化し、コンピューティング要件を指定します。AWS は、ワークロードの要求を満たす最も低コストの汎用 Amazon EC2 インスタンスタイプを自動的に選択します。または、インスタンスタイプ、CPU メーカー、アクセラレーターなどの必要なインスタンス属性を指定できます。Amazon ECS マネージドインスタンスは、スケーリング、パッチ適用、コスト最適化など、インフラストラクチャのあらゆる側面を完全に管理します。AWS 機能や Amazon EC2 統合へのアクセスは完全に保証されます。

Amazon ECS マネージドインスタンスは、プラットフォーム固有の最適化とセキュリティ設定で Linux コンテナに対応しています。デフォルトでは、Amazon ECS マネージドインスタンスは、複数の小さなタスクを大きなインスタンスに配置することでインフラストラクチャの使用率を最適化し、結果として、コストを削減し、タスクの起動時間を改善します。

このトピックでは、Amazon ECS マネージドインスタンスのタスクおよびサービスのさまざまなコンポーネントについて述べ、Amazon ECS で Amazon ECS マネージドインスタンスを使用する際の特別な考慮事項について説明します。

開始方法

Amazon ECS マネージドインスタンスの使用を開始するには、必要な IAM ロールを作成し、AWS アカウントで Amazon ECS マネージドインスタンスを有効にします。その後、キャパシティプロバイダを作成することで、Amazon ECS マネージドインスタンスキャパシティプロバイダを使用してタスクまたはサービスを起動できます。

開始方法についての詳細な説明については、「スタートガイド」の次の項目を参照してください。

- Amazon ECS マネージドインスタンスのタスクの作成方法について説明します。

- AWS CLI を使用した Amazon ECS マネージドインスタンスのタスクの作成方法について説明します。

キャパシティープロバイダー

Amazon ECS マネージドインスタンスは、キャパシティプロバイダを使用してワークロードのコンピューティングキャパシティを管理します。キャパシティプロバイダーを作成するときは、デフォルトのインスタンス選択を使用するか、`instanceRequirements` を使用してカスタムインスタンス要件を指定できます。

次のキャパシティプロバイダが利用できます。

- デフォルトのキャパシティプロバイダ – ワークロード要件に合わせて、最も低コストの汎用インスタンスタイプを自動的に選択します。
- カスタムキャパシティプロバイダ – vCPU 数、メモリ、CPU メーカー、アクセラレータータイプ、特定のインスタンスタイプなど、属性ベースのインスタンスタイプ選択を使用してインスタンス属性を指定できます。

キャパシティプロバイダ戦略には、次のリストからキャパシティプロバイダタイプを 1 つだけ含めることができます。

- Amazon ECS マネージドインスタンス
- Auto Scaling グループ
- Fargate/Fargate_SPOT

インスタンスの選択と最適化

Amazon ECS は、次のいずれかの方法で、Amazon ECS マネージドインスタンスワークロードのインスタンスタイプを選択します。

- 自動選択 – デフォルトのキャパシティプロバイダを使用する場合、Amazon ECS は、タスク定義で指定された CPU とメモリの要件を満たす、最も低コストの汎用インスタンスタイプを自動的に選択します。
- 属性ベースの選択 – カスタムキャパシティプロバイダを使用する場合、vCPU 数、メモリサイズ、CPU メーカー、アクセラレータータイプ、特定のインスタンスタイプなどのインスタンス属

性を指定できます。Amazon ECS は、指定した属性に一致するすべてのインスタンスタイプから選択します。

Amazon ECS マネージドインスタンスは、インフラストラクチャの使用率とコストを、いくつかのメカニズムを使用して最適化します。

- マルチタスク配置 – デフォルトでは、Amazon ECS は複数の小さなタスクを大きなインスタンスに配置して、使用率を最大化することによってコストを削減します。
- アクティブなワークロード統合 – Amazon ECS は、アプリケーションの可用性やデプロイのパフォーマンスに影響を与える可能性のある早期停止を回避しながら、コンテナインスタンスが完全にアイドル状態になるタイミングを特定します。システムは、サービスに設定されたタスクの最小数と最大数、スタートビフォーストップ動作、およびタスク保護動作を尊重します。
- 適切なサイズ設定 – ワークロード要件が変更されると、Amazon ECS は現時点の要求に合った適切なサイズの代替インスタンスを起動します。

Amazon ECS は Amazon EC2 イベントウィンドウを使用して、都合の良い期間にメンテナンスアクティビティをスケジュールします。イベントウィンドウを使用すると、AWS がインスタンス上でメンテナンスを実施できる定期的な期間を定義できるため、メンテナンスを運用スケジュールに合わせて調整することで、ワークロードの中断を最小限に抑えることができます。詳細については、「Amazon EC2 ユーザーガイド」の「インスタンスの予定されているイベント」を参照してください。

厳重な分離が必要な場合は、Amazon ECS マネージドインスタンスを、VM レベルのセキュリティ分離境界を持つ個別のインスタンスで各タスクを実行するように設定することができます。

タスク定義

Amazon ECS マネージドインスタンスを使用するタスクは、ほとんどの Amazon ECS タスク定義パラメータに対応しています。Amazon ECS マネージドインスタンスは、プラットフォームバージョン 1.4.0 を使用している既存の Fargate タスク定義と互換性があるため、移行が簡単になります。

Amazon ECS マネージドインスタンスを使用するには、`requiresCompatibilities` タスク定義パラメータに `MANAGED_INSTANCES` を含めます。タスク定義では、Fargate と Amazon ECS マネージドインスタンス双方の互換性を指定して、デプロイオプションに柔軟性を持たせることができます。

オペレーティングシステムと CPU アーキテクチャ

以下のオペレーティングシステムがサポートされています。

- Bottlerocket

Amazon ECS タスク定義には、ARM と X86_64 の 2 つのアーキテクチャを使用できます。

Amazon ECS マネージドインスタンスで Linux コンテナを実行する場合、ARM ベースのアプリケーションに X86_64 CPU アーキテクチャ、または ARM64 アーキテクチャを使用できます。

主な特徴

Amazon ECS マネージドインスタンスの主な特徴を以下に挙げます。

- アプリケーションの要件に合わせて特定の EC2 インスタンスタイプを選択することで、GPU アクセラレーテッドコンピューティング、特定の CPU 機能、大きなメモリサイズなどの特殊なハードウェア機能にアクセスできます。
- 独自の分離された環境で各タスクを実行する Fargate とは異なり、デフォルトで 1 つのインスタンスで複数のタスクを実行してリソースの使用率とコストを最適化します。
- セキュリティコンプライアンスを確保し、インスタンスパッチ適用を定期的に実行します。ECS マネージドインスタンスは、14 日後にインスタンスドレイニングを開始し、サービスベースのタスクを新しいインスタンスに自動的に置き換えます。
- CAP_NET_ADMIN、CAP_SYS_ADMIN、CAP_BPF、CAP_PERFMON などの特権 Linux 機能を使用して、コンテナ内での高度なネットワーキングとシステム管理機能を有効にできます。

IAM ロール

Amazon ECS マネージドインスタンスには 2 つの IAM ロールが必要です。

- インフラストラクチャロール: このロールにより、AWS はユーザーに代わって Amazon ECS マネージドインスタンスを管理できます。
- インスタンスプロファイル: インスタンスプロファイルは、IAM ロールを Amazon ECS マネージドインスタンスに渡すために使用します。このプロファイルは、次の目的で使用されます。
  - コンテナワークロードを実行する Amazon ECS マネージドインスタンスの IAM アクセス許可を定義します。

- AWS がユーザーに代わってこれらのインスタンスを管理できるようにします。
- プロファイルで定義されたアクセス許可に従って、インスタンスが AWS サービスにアクセスできるようにします。

セキュリティとコンプライアンス

Amazon ECS マネージドインスタンスは、ワークロードを保護するために複数のセキュリティレイヤーを実装しています。

- 安全な設定 – Amazon ECS マネージドインスタンスは、SSH アクセス禁止、変更不可能なルートファイルシステム、SELinux 経由のカーネルレベルの必須アクセスコントロールなど、AWS セキュリティのベストプラクティスに従います。
- 自動パッチ適用 – AWS は、設定したメンテナンスウィンドウを尊重して、最新のセキュリティパッチで Amazon ECS マネージドインスタンスを定期的に更新します。
- インスタンス存続期間の制限 – ECS が 14 日後にインスタンスドレイニングを自動的に開始して、最新のセキュリティパッチが適用され、適切に構成されたインスタンスでアプリケーションが実行されることを確実にします。
- 特権機能 – ネットワークモニタリングやオブザーバビリティソリューションなど、特権 Linux 機能を必要とするワークロードに対して、それらの機能をオプションで有効にできます。

Amazon ECS マネージドインスタンスは、PCI-DSS、HIPAA、FedRAMP など、Amazon ECS と同じコンプライアンスプログラムに対応しています。サポートされているリージョンでは、キャパシティプロバイダーレベルで FIPS エンドポイントを設定して、FedRAMP コンプライアンスを実現できます。

ネットワーク

Amazon ECS マネージドインスタンスは、`awsvpc` および `host` ネットワークモードに対応しています。`awsvpc` ネットワークモードは、VPC 内で独自の Elastic Network Interface とプライベート IP アドレスを各タスクに提供します。これにより、タスクレベルでのきめ細かなセキュリティグループとネットワーク ACL 制御が可能になります。`host` ネットワークモードでは、タスクはホストの Amazon ECS マネージドインスタンスネットワーク名前空間を共有します。Amazon ECS マネージドインスタンスでのタスクネットワーキングの詳細については、「Amazon ECS マネージドインスタンスの Amazon ECS タスクネットワーキング」を参照してください。

インスタンスストレージ

Amazon ECS マネージドインスタンスは、インスタンスに付加されている Amazon EBS データボリュームのサイズ設定に対応しています。このストレージは、インスタンスで実行されるすべてのタスク間で共有され、バインドマウントに使用できます。このボリュームは、タスク定義内で volumes、mountPoint、および volumesFrom パラメータを使用しているコンテナ間で共有およびマウントが可能です。

ボリュームはインスタンスの作成時に付加されます。storageConfiguration パラメータを使用して Amazon ECS マネージドインスタンスのキャパシティプロバイダを作成するときに、ボリュームのサイズを GiB 単位で指定できます。

```
{
...

    "managedInstancesProvider": {
        "infrastructureRoleArn": "arn:aws:iam::123456789012:role/
ecsInfrastructureRole",
        "instanceLaunchTemplate": {
            "ec2InstanceProfileArn": "arn:aws:iam::123456789012:instance-profile/
ecsInstanceProfile",
            "networkConfiguration": {
                "subnets": [
                    "subnet-abcdef01234567",
                    "subnet-bcdefa98765432"
                ],
                "securityGroups": [
                    "sg-0123456789abcdef"
                ]
            },
            "storageConfiguration": {
                "storageSizeinGiB" : 100

            }
        }
    }
 ...
}
```

このボリュームの最小サイズは 30 GiB、最大サイズは 16,384 GiB です。デフォルトでは、ボリュームのサイズは 80 GiB です。

タスクで、取り込み、圧縮、および非圧縮されたコンテナイメージは、このボリュームに格納されます。タスクがバインドマウントに使用すべきインスタンスストレージの総量を判断するには、タスクに割り当てられているインスタンスのストレージの総量から、コンテナイメージが使用するストレージの容量を差し引く必要があります。

Amazon ECS マネージドインスタンスに付加された Amazon EBS ボリュームのパフォーマンスは、「Amazon EC2 ユーザーガイド」の「Amazon EBS 最適化インスタンス」ドキュメントで説明されているように、対応する Amazon EC2 インスタンスのパフォーマンスと一致します。

ボリュームのスナップショットを作成して、セキュリティ問題の法的分析を実行、もしくはアプリケーションのデバッグができます。Amazon EBS ボリュームのスナップショット作成の詳細については、「Amazon EBS ユーザーガイド」の「Amazon EBS スナップショット」を参照してください。Amazon EBS 暗号化がデフォルトで有効になっている場合、ボリュームはデフォルトで暗号化用に指定された AWS KMS キーで暗号化されます。デフォルトでの EBS 暗号化の詳細については、「Amazon EBS ユーザーガイド」の「Amazon EBS 暗号化をデフォルトで有効にする」を参照してください。

インスタンスに付加されたデータボリュームを使用するだけでなく、Amazon ECS マネージドインスタンス上で実行されるタスクごとにデータボリュームを設定することもできます。タスクレベルストレージの詳細については、「Amazon ECS タスクのストレージオプション」を参照してください。

ローカルストレージ

Amazon EC2 インスタンスストアは Amazon EC2 インスタンス用に一時的なブロックレベルストレージを提供します。Amazon EC2 インスタンスストアが提供するストレージは、ホストに物理的にアタッチされたディスクからアクセスできます。コンテナインスタンスで実行されているタスクのデータボリュームとしてインスタンスストアを使用するように Amazon ECS マネージドインスタンスを設定できます。ローカルストレージを有効にし、インスタンスにインスタンスストアボリュームがある場合、Amazon ECS は Amazon EBS データボリュームをプロビジョニングする代わりにインスタンスストアボリュームを使用します。これにより、ストレージコストを削減し、レイテンシーの影響を受けやすいワークロードの I/O パフォーマンスを向上させることができます。Amazon ECS マネージドインスタンスでのインスタンスストアボリュームの使用には、追加料金はかかりません。

インスタンスに複数のインスタンスストアボリュームがある場合、Amazon ECS はそれらを自動的に 1 つの RAID 0 ボリュームに結合し、タスク用の連続ストレージとして表示します。インスタンスにインスタンスストアボリュームがない場合、またはローカルストレージが無効になっている場合、Amazon ECS は `storageSizeGiB` で指定されたサイズで Amazon EBS データボリュームをプ

ロビジョニングします。インスタンスストアボリュームはエフェメラルです。インスタンスストアボリュームのデータは、インスタンスが停止または終了したとき、またはハードウェアに障害が発生したときに失われます。永続化する必要があるデータには、インスタンスストアを使用しないでください。

ローカルストレージを有効にするには、localStorageConfiguration パラメータを使用して新しいキャパシティプロバイダーを作成し、useLocalStorage を true に設定します。instanceRequirements パラメータを使用して、プロビジョニングされたインスタンスに特定のサイズのローカルストレージが含まれるようにすることもできます。

> **Note**
>
> AWS マネジメントコンソール を使用してクラスターを作成する場合、Amazon ECS が作成するデフォルトのマネージドインスタンスキャパシティプロバイダーには localStorageConfiguration は含まれません。

次の例は、ローカルストレージが有効になっているキャパシティプロバイダー設定と、最小で 50 GiB の SSD ベースのインスタンスストアを指定するインスタンス要件を示しています。

```
{
...
    "managedInstancesProvider": {
        "infrastructureRoleArn": "arn:aws:iam::123456789012:role/
ecsInfrastructureRole",
        "instanceLaunchTemplate": {
            "ec2InstanceProfileArn": "arn:aws:iam::123456789012:instance-profile/
ecsInstanceProfile",
            "networkConfiguration": {
                "subnets": [
                    "subnet-abcdef01234567",
                    "subnet-bcdefa98765432"
                ],
                "securityGroups": [
                    "sg-0123456789abcdef"
                ]
            },
            "storageConfiguration": {
                "storageSizeGiB": 50
            },
            "localStorageConfiguration": {
```

```
                "useLocalStorage": true
            },
            "instanceRequirements": {
                "vCpuCount": {
                    "min": 0
                },
                "memoryMiB": {
                    "min": 0
                },
                "localStorage": "REQUIRED",
                "totalLocalStorageGB": {
                    "min": 50
                },
                "localStorageTypes": ["ssd"]
            }
        }
    }
}
```

instanceRequirements 設定を省略すると、Amazon ECS はデフォルトで localStorage から INCLUDED になります。この場合、Amazon ECS はインスタンスストアボリュームの有無にかかわらず、インスタンスをプロビジョニング用と見なします。インスタンスストアボリュームを持つインスタンスはデータボリュームにインスタンスストアを使用し、インスタンスストアボリュームを持たないインスタンスは Amazon EBS データボリュームを使用します。totalLocalStorageGB が指定されていない場合、Amazon ECS は利用可能なすべてのインスタンスストレージを使用し、最小サイズは適用されません。

storageConfiguration の storageSizeGiB 値は Amazon EBS データボリュームサイズのみを定義し、インスタンスストレージが利用できない場合に使用されます。instanceRequirements の totalLocalStorageGB 値は、プロビジョニングされたインスタンスに必要な最小インスタンスストレージサイズを制御します。すべてのインスタンスでローカルストレージが使用されるようにするには、localStorage を REQUIRED に設定し、totalLocalStorageGB を使用して最小サイズを指定します。

特定のプロビジョニングされたコンテナインスタンスがローカルストレージを使用しているかどうかを確認するには、コンテナインスタンスの ecs.capability.storage.local-storage-enabled 属性を確認します。

サービスの負荷分散

Amazon ECS マネージドインスタンスを使用する Amazon ECS サービスは、Elastic Load Balancing を使用してサービス内のタスク全体にトラフィックを均等に分散するよう設定できます。

Amazon ECS マネージドインスタンス上の Amazon ECS サービスは、Application Load Balancer、Network Load Balancer、および Gateway Load Balancer の各ロードバランサータイプに対応しています。Application Load Balancer は HTTP/HTTPS (レイヤー 7) のトラフィックをルーティングし、Network Load Balancer は TCP または UDP (レイヤー 4) のトラフィックをルーティングします。

また、このようなサービスのターゲットグループを作成する場合は、ターゲットタイプとして `instance` ではなく、`ip` を選択する必要があります。これは、`awsvpc` ネットワークモードを使用するタスクは、直接 Amazon EC2 インスタンスではなく、Elastic Network Interface に関連付けられているためです。

モニタリングとオブザーバビリティ

Amazon ECS マネージドインスタンスは、CloudWatch メトリックおよびオブザーバビリティツールとの統合を通じて包括的なモニタリング機能を提供します。

- CloudWatch メトリック – タスクレベルとインスタンスレベルの両方で CPU、メモリ、ネットワーク、およびストレージの使用率をモニタリングします。
- Container Insights – コンテナ化されたアプリケーションの詳細なパフォーマンスメトリックとログを取得します。
- サードパーティーの統合 – 特権機能を有効にすると、高いレベルの Linux アクセス許可を必要とする高度なモニタリングおよびオブザーバビリティソリューションを実行できます。

料金とコストの最適化

Amazon ECS マネージドインスタンスでは、タスクを実行している Amazon EC2 インスタンス全体に対して課金されます。料金は、ワークロード用に選択されたインスタンスタイプによって異なります。

Amazon ECS マネージドインスタンスでは、いくつかのコスト最適化機能が提供されています。

- マルチタスクの最適化 – 適切なサイズに設定されたインスタンスで複数のタスクを実行することにより、インスタンスの使用率を最大化します。

Compute and Instance Savings Plans は、Amazon ECS マネージドインスタンスワークロードにも適用されます。

Service Quotas

Amazon ECS マネージドインスタンスワークロードには、Amazon EC2 オンデマンドインスタンス Service Quotas が適用されます。Amazon ECS マネージドインスタンスを使用する Amazon ECS サービスには、Amazon ECS Service Quotas が適用されます。

Service Quotas の詳細については、次の資料を参照してください。

- 「Amazon Web Services 全般のリファレンス」の「Amazon EC2 エンドポイントとクオータ」
- Amazon ECS の Service Quotas

移行に関する考慮事項

Amazon ECS マネージドインスタンスへの移行は、ほとんどのワークロードで簡単に行えます。

- Fargate から – キャパシティプロバイダの設定変更と再デプロイのみが必要です。プラットフォームバージョン 1.4.0 を使用している既存のタスク定義には完全な互換性があります。
- EC2 から – Fargate への移行と同様ですが、特定のインスタンスタイプなどの Amazon EC2 機能へのアクセスは保持されます。

移行を計画する際には、次の点を考慮してください。

- アプリケーションは、14 日間のインスタンス存続期間と計画されたメンテナンスウィンドウに対応できる必要があります。
- 長期間実行されるタスク (14 日以上) は、Amazon ECS マネージドインスタンスには適していません。
- カスタム AMI はサポートされていません – Amazon ECS マネージドインスタンスは、AWS が管理するセキュリティ最適化 AMI を使用します。

制約事項と考慮事項

Amazon ECS マネージドインスタンスには、次の制限が適用されます。

- カスタム AMI – AMI は AWS によって所有および管理されます
- インスタンスの存続期間 – セキュリティパッチ適用とコンプライアンスを確実にするため、インスタンスあたりの最大ランタイムは 14 日です。
- SSH アクセス – セキュリティ上の理由から使用できません。Amazon ECS Exec を使用してデバッグとトラブルシューティングを行います。管理オペレーションは Amazon ECS API 経由のみで可能です。

組織コントロール

組織コントロールには、Amazon ECS マネージドインスタンスの正しい動作を妨げる可能性があるものもあります。その場合は、これらのコントロールを更新して、Amazon ECS がユーザーに代わって EC2 インスタンスを管理するために必要な許可を得られるようにする必要があります。

Amazon ECS は、Amazon ECS マネージドインスタンスを支える EC2 インスタンスの起動にインフラストラクチャロールを使用します。このインフラストラクチャロールは、Amazon ECS がユーザーに代わって Amazon ECS マネージドインスタンスを管理できるようにする IAM ロールであり、アカウント内に作成されます。サービスロールで実行されるアクションには、サービスコントロールポリシー (SCP) が常に適用されます。このため、SCP は Amazon ECS マネージドインスタンスの運用を阻止できます。最も一般的なケースは、SCP を使用して起動できる Amazon マシンイメージ (AMI) を制限する場合です。Amazon ECS マネージドインスタンスを機能させるには、Amazon ECS マネージドインスタンス AMI アカウントからの AMI の起動を許可するように SCP を変更します。

EC2 が許可された AMI 機能を使用し、他のアカウントで AMI の可視性を制限することもできます。この機能を使用する場合は、イメージ基準の範囲を拡大して、対象リージョン内の Amazon ECS マネージドインスタンス AMI アカウントも範囲に含まれるようにする必要があります。

Amazon ECS マネージドインスタンス AMI 以外のすべての AMI をブロックする SCP の例

以下の SCP は、AMI が us-west-2 または us-east-1 の Amazon ECS マネージドインスタンス AMI アカウントに属している場合を除き、`ec2:RunInstances` の呼び出しを阻止します。

> **Note**
>
> `ec2:Owner` コンテキストキーを使用しないことが重要です。Amazon は Amazon ECS マネージドインスタンス AMI アカウントを所有しており、このキーの値は常に `amazon` になります。`ec2:Owner` が `amazon` の場合に AMI の起動を許可する SCP を構築すると、Amazon

ECS マネージドインスタンス用の AMI だけでなく、Amazon が所有する任意の AMI も起動できます。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "DenyAMI",
      "Effect": "Deny",
      "Action": "ec2:RunInstances",
      "Resource": "arn:*:ec2:*::image/ami-*",
      "Condition": {
        "StringNotEquals": {
          "aws:ResourceAccount": [
            "187296253231",
            "260073348889"
          ]
        }
      }
    }
  ]
}
```

Amazon ECS マネージドインスタンス AMI アカウント

リージョンによって異なる AWS アカウントが Amazon ECS マネージドインスタンスのパブリック AMI をホストします。

AWS リージョン	アカウント
af-south-1	070957084703
ap-east-1	587573215167
ap-northeast-1	679336465495
ap-northeast-2	309903600357
ap-northeast-3	384570461223

AWS リージョン	アカウント
ap-south-1	062344138989
ap-south-2	624198668379
ap-southeast-1	832199679391
ap-southeast-2	552073033681
ap-southeast-3	368903466070
ap-southeast-4	696793786439
ap-southeast-5	003457290689
ap-southeast-6	465836752572
ap-southeast-7	622515864387
ca-central-1	853167153192
ca-west-1	899469777611
eu-central-1	832570432258
eu-central-2	041659148495
eu-north-1	851563870067
eu-south-1	766433696616
eu-south-2	003380494496
eu-west-1	986619735082
eu-west-2	591706807364
eu-west-3	108582616801
il-central-1	009537862704

AWS リージョン	アカウント
me-central-1	540883425316
me-south-1	181438624895
mx-central-1	210749644920
sa-east-1	591338347621
us-east-1	260073348889
us-east-2	292185169523
us-west-1	187296253231
us-west-2	491085424538

Amazon ECS マネージドインスタンスのインスタンスタイプ

Amazon ECS マネージドインスタンスキャパシティプロバイダーを作成する際に、Amazon ECS がタスクの EC2 インスタンスタイプを選択する方法を選択します。Amazon ECS で自動的に選択したり、正確なインスタンスタイプやファミリーを指定したり、インスタンスが満たす必要があるハードウェア属性を定義したりできます。

インスタンスの選択方法

Amazon ECS マネージドインスタンスには、インスタンスタイプを選択するための 2 つの方法があります。

- 特定のインスタンスタイプまたはファミリー: ワイルドカード (`c6i.*` や `m6*.*` など) を使用して、正確なインスタンスタイプ (`m6i.xlarge` など) またはインスタンスファミリーを指定します。`allowedInstanceTypes` を使用して特定のタイプを含めるか、不要なタイプを除外する場合は `excludedInstanceTypes` を使用します。
- 属性ベースの選択: ハードウェア要件 (vCPU 数、メモリ、CPU 製造元、アクセラレータータイプなど) を定義し、Amazon ECS は指定したすべての属性に一致するインスタンスタイプを選択します。

インスタンスタイプや属性を指定しない場合、Amazon ECS はタスク定義の要件に基づいて、最もコスト効率の良いインスタンスタイプを自動的に選択します。

特定のインスタンスタイプの選択

特定のインスタンスタイプを選択すると、Amazon ECS マネージドインスタンスが使用できる EC2 インスタンスタイプを正確に制御できます。個々のインスタンスタイプ、インスタンスファミリー全体、またはその両方を組み合わせて指定できます。

含める型を指定するには `allowedInstanceTypes` フィールドを使用し、含めない型を除外するには `excludedInstanceTypes`フィールドを使用します。どちらのフィールドもワイルドカードをサポートしています。

- `m6i.xlarge` — 特定のインスタンスタイプ
- `c6i.*` — c6i ファミリーのすべてのサイズ
- `m6*.*` — すべての m6 世代ファミリー (m6i、m6a、m6g など)

属性ベースのインスタンスタイプの選択

属性ベースのインスタンスタイプを選択すると、ハードウェア要件を定義し、Amazon ECS はそれらの要件を満たすインスタンスタイプを選択します。これにより柔軟性が確保され、特定のインスタンスタイプが利用できない場合でも、タスクが確実に配置されるようになります。

複数の属性を指定すると、Amazon ECS が指定したすべての属性を満たすインスタンスタイプを取得します。属性に複数の値を指定すると、Amazon ECS は指定された値のいずれかを満たすインスタンスタイプを取得します。

以下のカテゴリの属性でフィルターできます。

- CPU とメモリ — vCPU 数、メモリ (MiB)、vCPU あたりのメモリ比率、CPU 製造元 (Intel、AMD、AWS Graviton)、インスタンス生成 (現在または以前) 。
- ローカルストレージ — ローカルストレージを含める、除外する、または要求します。ストレージタイプ (HDD または SSD) と合計ストレージ容量でフィルタリングします。
- ネットワークとパフォーマンス — ネットワーク帯域幅 (Gbps)、ネットワークインターフェイス数、ベースライン EBS 帯域幅 (Mbps)、バーストパフォーマンスのサポート、ベアメタルインスタンス。
- アクセラレーター — アクセラレーターの数、製造元、チップ名、アクセラレータータイプ、およびアクセラレータの総メモリ容量。

- 価格保護 — 要件に一致する最低価格のインスタンスタイプに対する、スポット価格およびオンデマンド価格の上限をパーセンテージで示します。

属性と有効な値の完全なリストについては、Amazon ECS API Reference の InstanceRequirementsRequest を参照してください。

請求と購入のオプション

容量プロバイダーを作成する際に、プロビジョニングモデルを選択します。プロビジョニングモデルは、Amazon ECS マネージドインスタンスがタスクの EC2 インスタンスを取得する方法を決定します。

オンデマンド

コンピューティング性能に対して時間単位で支払いでき、長期契約や前払いは必要ありません。これがデフォルトです。

Spot

予備の EC2 容量を割引価格で利用できます。スポットインスタンスは、EC2 が容量を解放する必要が生じた場合に中断される可能性があります。キャパシティープロバイダーの作成時に `capacityOptionType` を `Spot` に設定します。

キャパシティ予約

特定のアベイラビリティーゾーンで EC2 の容量を予約します。`capacityOptionType` を `Reserved` に設定し、キャパシティ予約グループ を指定します。予約設定として、`reservations-only`、`reservations-first`、または `reservations-excluded` を指定することもできます。

Amazon ECS マネージドインスタンスは、タスクで使用するインスタンスタイプ用に購入した AWS Savings Plans やリザーブドインスタンスのメリットも自動的に享受できます。追加の設定は必要ありません。

Amazon ECS マネージド インスタンスでサポートされているインスタンスタイプ

Amazon ECS マネージドインスタンスは、次のインスタンスタイプをサポートしています。

- コンピューティング最適化:C5 | C5a | C5ad | C5d | C5n | C6a | C6g | C6gd | C6i | C6id | C6in | C7a | C7g | C7gd | C7gn | C7i | C7i-flex | C8a | C8g | C8gb | C8gd | C8gn | C8i | C8ib | C8id | C8i-flex | C8in | C8ine
- 汎用:M5 | M5a | M5ad | M5d | M5dn | M5n | M5zn | M6a | M6g | M6gd | M6i | M6id | M6idn | M6in | M7a | M7g | M7gd | M7i | M7i-flex | M8a | M8azn | M8g | M8gb | M8gd | M8gn | M8i | M8ib | M8id | M8idb | M8idn | M8i-flex | M8in | M8ine | M9g | M9gd | T3 | T3a | T4g
- メモリ最適化: R5 | R5a | R5ad | R5b | R5d | R5dn | R5n | R6a | R6g | R6gd | R6i | R6id | R6idn | R6in | R7a | R7g | R7gd | R7i | R7iz | R8a | R8g | R8gb | R8gd | R8gn | U-3tb1 | U7i-6tb | U7i-8tb | U7i-12tb | U7in-24tb | U7in-32tb | X2gd | X2idn | X2iedn | X2iezn | X8g | Z1d
- ストレージ最適化: D3 | D3en | I3en | I4g | I4i | I7i | I7ie | I8g | I8ge | Im4gn | Is4gen
- アクセラレーテッドコンピューティング: DL1 | G4ad | G4dn | G5 | G5g | G6 | G6e | G6f | G7e | Gr6 | Gr6f | Inf1 | Inf2 | P3dn | P4d | P4de | P5 | P5en | P6-B200 | P6-B300 | Trn1
- ハイパフォーマンスコンピューティング: Hpc6a | Hpc6id | Hpc7a | Hpc7g

さらに、Amazon ECS マネージドインスタンスは、以下の要件を満たすインスタンスのみを作成します。

- 1 個以上の vCPU
- インスタンスサイズがナノ、もしくはマイクロではない

詳細については、「Amazon EC2 インスタンスタイプの命名規則」を参照してください。

Amazon ECS マネージドインスタンスにおけるインスタンス選択のベストプラクティス

Amazon ECS マネージドインスタンスのワークロードに適したインスタンス設定を選択することは、パフォーマンス、コスト、リソース使用率を最適化するために不可欠です。Amazon ECS には、アプリケーション要件とコスト効率のバランスを取るための柔軟なインスタンス選択オプションが用意されています。次のベストプラクティスは、コンテナ化されたワークロードのインスタンス選択の際に、情報に基づいた意思決定を行うのに役立ちます。

1. デフォルトのインスタンス選択を使用する

   `instanceRequirements` を指定しないと、Amazon ECS は、次に示すタスク定義とサービスパラメータ要件を満たす最も費用対効果の高いインスタンスを選択します。

タスク定義

- operatingSystemFamily
- cpuArchitecture
- cpu
- メモリ

サービスの定義

- placementConstraints
- placementStrategy

2. ほとんどのワークロードで属性ベースの選択を使用して柔軟性を提供し、配置の成功率を向上させる

   属性ベースのインスタンス選択の場合、Amazon ECS は指定した要件を満たす幅広いインスタンスタイプから選択することができます。このアプローチにより、Amazon ECS が起動時に利用可能な最も費用対効果の高いインスタンスを選択できるようになるため、タスク配置が成功する可能性が高まり、コスト最適性が向上します。

3. アプリケーションに特定のハードウェア要件がある場合にのみ、特定のインスタンスタイプを使用する

   GPU アクセラレーション、高クロック周波数プロセッサ、特殊なネットワーク機能など、特定のハードウェア機能を必要とするワークロード用に、特定のインスタンスタイプの選択を予約します。汎用アプリケーションの場合には、属性ベースの選択により、通常、柔軟性とコスト最適性が向上します。

4. 過剰な割り当てや不要なコストを避けるために、バランスの取れたリソースを選択する

   アプリケーションの CPU およびメモリ要件に最も近いインスタンス設定を選択します。コストが高くなり、効率が低下するため、リソースの著しく過剰な割り当ては避けてください。モニタリングデータを使用して実際のリソース使用率のパターンを理解し、それに応じてインスタンスの選択を調整します。

5. ワークロードが異なるアプリケーションのインスタンスタイプを混在させて、パフォーマンスとコストのバランスをとる

   さまざまなパフォーマンス要件やワークロードパターンを持つアプリケーションの場合は、インスタンス設定が異なる複数のキャパシティプロバイダを使用することを検討してください。この

アプローチにより、必要に応じてパフォーマンスを維持しながら、アプリケーションのさまざまなコンポーネントに適切なインスタンスタイプを使用してコストを最適化できます。

6. capacityOptionType=Reserved が設定された Amazon ECS マネージドインスタンスキャパシティプロバイダーを使用するときは、ECS サービスが minimumHealthyPercent=100% と maximumPercent=200% のデフォルトデプロイ設定を使用することに注意してください。これは、ECS デプロイが古いタスクを停止する前に新しいタスクを開始しようと試み、定常状態キャパシティの最大 200% が一時的に必要になることを意味します。サービスが定常状態の EC2 キャパシティ予約にある使用可能なキャパシティのすべてを消費すると、デプロイプロセス中に新しいタスクを起動するための追加のキャパシティがなくなるため、デプロイが失敗します。この状況を回避するため、minimumHealthyPercent を 100% 未満 (75% など) に設定するとともに、maximumPercent を 100% に設定することを検討して、サービスがタスクを停止してから新しいタスクを開始するようにしてください。そうすることで、代替タスクの起動前にキャパシティが解放され、デプロイを正常に完了できるようになります。また、デプロイに対応し、トラフィックの急増に対処するための余裕を予約内に維持できるように、キャパシティ使用率を定期的に監視することも検討してください。

Amazon ECS マネージドインスタンスのコンテナイメージの取り込み動作

コンテナのスタートアップ時間は、基盤となるコンテナイメージのサイズによって異なります。Debian、Ubuntu、Amazon Linux 2 のフルバージョンなどの大きなイメージは、slim 対応 (Debian-slim、Ubuntu-slim、Amazon-slim) や Alpine などの最小限のベースイメージよりも多くのサービスが含まれているため、起動に時間がかかります。

Amazon ECS が新しい Amazon ECS マネージドインスタンスコンテナインスタンスを起動すると、ローカルイメージキャッシュは空になります。そのコンテナインスタンスで実行される最初のタスクは、コンテナリポジトリからローカルイメージキャッシュにコンテナイメージをプルします。コンテナインスタンスで同じイメージを使用する後続のタスクでは、キャッシュされたイメージを再度プルするのではなく、キャッシュされたイメージを使用します。

Amazon ECS マネージドインスタンスの Amazon ECS エージェントは、各コンテナインスタンスのディスク容量を自動的に管理します。タスクが停止し、他の実行中のタスクが同じコンテナイメージを使用しない場合、Amazon ECS はそのイメージの 15 分間のクリーンアップタイマーを作動させます。イメージを使用する新しいタスクが 15 分以内にそのコンテナインスタンスで開始されない場合、イメージはローカルイメージキャッシュから削除されます。

コンテナイメージの解決

Amazon ECS はコンテナイメージタグをイメージダイジェストに解決し、サービス内のすべてのタスクが同じイメージバージョンを実行するようにします。この動作は、Amazon ECS マネージドインスタンスのビンパッキングとインスタンスの選択に影響します。Amazon ECS は、残りのタスクをスケジュールする前にコンテナイメージダイジェストを取得するために、まずサービスから単一のタスクをスケジュールするためです。詳細については、「コンテナイメージの解決」を参照してください。

Amazon ECS マネージドインスタンスでのパッチ適用

Amazon ECS マネージドインスタンスでは、パッチ適用は重要なメンテナンスプロセスで、その中で AWS は制御された設定可能なプロセスを通じてワークロードの可用性を維持しながら、マネージドコンテナインスタンスのソフトウェアを自動的に管理および更新することで、セキュリティとコンプライアンスを確実にします。

インスタンスのライフサイクル

デフォルトでは、Amazon ECS マネージドコンテナインスタンスは標準化された 14～21 日のライフサイクルで動作します。Amazon ECS は、インスタンスの起動から 14 日目に段階的なワークロードドレイニングを開始し、21 日目までに停止します。Amazon ECS は、特定の状況下での早期ドレイニングに対応します。

- ソフトウェアのセキュリティ脆弱性の検出
- ハードウェア状態の劣化
- 顧客が設定したイベントウィンドウを優先するため

このアプローチは、顧客が定義したメンテナンス要件を尊重しながら、システムのコンプライアンスを維持します。

イベントウィンドウとメンテナンススケジューリング

AWS は、ノードの作成タイムスタンプとメンテナンススケジュールをモニタリングする自動バックグラウンドプロセスを通じて、マネージドコンテナインスタンスのライフサイクルを管理します。インスタンスの起動時に、AWS はデフォルトの 14 日のドレイニングスケジュールを設定し、顧客が設定したイベントウィンドウを評価します。

イベントウィンドウを設定することで、Amazon ECS マネージドインスタンスのメンテナンスアクティビティをスケジュールできます。インスタンス ID またはインスタンスタグを使用して、1 つ以上の Amazon ECS マネージドインスタンスをイベントウィンドウに関連付けることができます。イベントウィンドウに特定の値がタグ付けされると、Amazon ECS はこれらのタグを該当するクラスターの該当する Amazon ECS マネージドインスタンスにマッピングし、最大限の努力で定義された期間中にインスタンスのメンテナンスをスケジュールします。

イベントウィンドウの詳細については、「Amazon EC2 ユーザーガイド」の「Amazon EC2 インスタンスに影響のあるスケジュール済のイベントのカスタムイベントウィンドウを作成する」を参照してください。

イベントウィンドウが存在する場合、AWS はこれらのウィンドウに合わせてドレイニングスケジュールを調整します。これにより、指定されたイベントウィンドウを優先するために、デフォルトの 14 日よりも早いドレイニングが発生する可能性があります。イベントウィンドウの変更は、新しく起動されたマネージドコンテナインスタンスにのみ影響するため、予知保全のスケジューリングが確保されます。

スケジュールされたドレイニング時間まで、Amazon ECS は顧客の設定に従ってマネージドコンテナインスタンス上で通常のタスク配置動作を続行します。

メンテナンスシーケンス

指定されたメンテナンス時に、Amazon ECS は `UpdateContainerInstancesState` API を呼び出してメンテナンスシーケンスを開始し、段階的なワークロードのドレイニングを起動します。段階的な停止期間中、Amazon ECS はドレイニング対象として印付けされたインスタンスのワークロードを停止しようとします。

Amazon ECS は、サービスのデプロイ設定 (`maximumPercent` および `minimumHealthyPercent`) に従ってタスクを停止し、代替タスクを起動します。デフォルトの `maximumPercent` 値が 200% の場合、Amazon ECS はまず新しいタスクをスケジュールし、それらが実行されるのを待ってから既存のタスクを停止し、サービスの中断を最小限に抑えます。`maximumPercent` が 100% に設定されている場合、Amazon ECS はまずタスクを停止し、その後置き換えます。このプロセス全体を通して、Amazon ECS サービスは、インスタンスの起動から 21 日目までドレイニング試行を継続しながら、すべてのサービスデプロイ設定を優先します。

ドレイニングが 21 日目までに完了していない場合、Amazon ECS は `DeregisterContainerInstance` API を実行してマネージドコンテナインスタンスとその残りのワークロードを停止することにより、コンプライアンスを維持し、マネージドインスタンスに最新のソフトウェアをパッチ適用します。

Amazon ECS マネージドインスタンスのセキュリティ上の考慮事項

Amazon ECS マネージドインスタンスは、完全マネージド型のコンテナコンピューティング体験を提供します。これにより、セキュリティ上の責任を AWS で負荷軽減しながら、特定の Amazon EC2 インスタンスタイプでワークロードを実行できます。このトピックでは、Amazon ECS マネージドインスタンスを使用する際のセキュリティモデル、機能、考慮事項について説明します。

Amazon ECS マネージドインスタンスのセキュリティにおける AWS とお客様の責任については、「Amazon ECS マネージドインスタンスの責任共有モデル」を参照してください。

セキュリティモデル

Amazon ECS マネージドインスタンスは、柔軟性と保護のバランスをとる包括的なセキュリティモデルを実装しています。

- AWS マネージドインフラストラクチャ – AWS はマネージドインスタンスのライフサイクルを制御し、セキュリティパッチ適用を処理することで、人為的ミスや改ざんの可能性を排除します。
- 管理アクセスの禁止 – セキュリティモデルは隔離されて、マネージドインスタンスへの管理アクセスが禁止されます。
- マルチタスク配置 – デフォルトでは、Amazon ECS マネージドインスタンスはコストと使用率を最適化するために 1 つのインスタンスに複数のタスクを配置するため、Fargate と比較してワークロード分離の制約が緩和されます。
- データ分離 – AWS はインスタンスのライフサイクルとタスクの配置を制御しますが、AWS はマネージドインスタンスにログインしたり、顧客データにアクセスしたりすることはできません。

マネージドインスタンスの概要

Amazon ECS マネージドインスタンスは、アカウントに EC2 マネージドインスタンスをプロビジョニングします。Amazon ECS は指定されたオペレーターとして、ユーザーに代わって、プロビジョニング、スケーリング、パッチ適用、終了など、インスタンスのライフサイクル全体を管理します。お客様には、これらのインスタンスを直接終了する権限も、インスタンス設定を変更する権限もありません。詳細については、「Amazon EC2 マネージドインスタンス」を参照してください。

マネージドインスタンスの特定

アカウント内の Amazon ECS マネージドインスタンスは、次のインジケータを使用して識別できます。

- Amazon EC2 `DescribeInstances` レスポンスの `Operator` フィールド。値は `ecs.amazonaws.com`。
- インスタンスの `aws:ec2:managed-launch` タグ。値は `ecs-managed-instances`。

マネージドリソースの可視性

2026 年 4 月 22 日以降、Amazon EC2 はデフォルトで Amazon EC2 コンソールビューと API リストオペレーションの新しいマネージドインスタンスを非表示にします。可視性設定は請求やリソースオペレーションには影響せず、マネージドインスタンスは可視性設定に関係なく完全に動作し、請求可能です。(この動作はいつでも調整できます)。詳細については、「マネージドリソースの可視性設定」を参照してください。

セキュリティ機能

Amazon ECS マネージドインスタンスには、ワークロードを保護し、強力なセキュリティ体制を維持するように設計されたいくつかの組み込みセキュリティ機能が含まれています。これらの機能は、自動セキュリティパッチ適用から、必要に応じた特権 Linux 機能への対応まで多岐にわたります。

セキュリティのベストプラクティス

マネージドインスタンスは、AWS セキュリティのベストプラクティスに従って設定され、それには次の項目を含みます。

- SSH アクセス禁止 – 不正アクセスを防ぐために、リモートシェルアクセスは無効化されています。
- 変更不可能なルートファイルシステム – ルートファイルシステムは変更できず、システムの整合性が確保されます。
- カーネルレベルの必須アクセスコントロール – SELinux は、カーネルレベルで追加のセキュリティ施行を提供します。

自動セキュリティパッチ適用

Amazon ECS マネージドインスタンスは、自動パッチ適用を通じてワークロードのセキュリティ体制の改善を支援します。

- 定期的なセキュリティ更新 – インスタンスは、AWS によって、設定したメンテナンスウィンドウに従い、最新のセキュリティパッチを用いて定期的に更新されます。
- インスタンスの有効期間の制限 – 実行中のインスタンスの最大存続期間は 14 日間に制限され、最新のセキュリティパッチで適切に設定されたインスタンスでアプリケーションが実行されることを確実にします。
- メンテナンスウィンドウの制御 – Amazon EC2 イベントウィンドウ機能を使用して、Amazon ECS がインスタンスをパッチ適用済みのインスタンスにいつ置き換えるかを指定できます。

特権 Linux 機能

Amazon ECS マネージドインスタンスは、高いレベルの Linux 権限を必要とするソフトウェアに対応することで、高度なモニタリングとセキュリティソリューションを可能にします。

- サポート対象の機能 – `CAP_NET_ADMIN`、`CAP_SYS_ADMIN`、`CAP_BPF`、`CAP_PERFMON` など、すべての特権 Linux 機能に対してオプトインできます。
- 一般的に普及しているソリューション – これにより、Wireshark や Datadog などの一般的に普及しているネットワークモニタリングおよびオブザーバビリティソリューションを実行できます。
- 明示的な設定が必要 – アプリケーションに追加のセキュリティリスクが生じる可能性があるため、特権 Linux 機能を有効にするように Amazon ECS マネージドインスタンスのキャパシティプロバイダを明示的に設定する必要があります。

> **Important**
>
> 特権 Linux 機能を有効にすると、タスクが追加のセキュリティリスクにさらされる可能性があります。これらの機能は、アプリケーションが必要とする場合にのみ有効にし、セキュリティへの影響に対する理解を確実にしてください。

コンプライアンスと規制への対応

Amazon ECS マネージドインスタンスは、Amazon ECS と同じコンプライアンス体制を維持します。

- コンプライアンスプログラム – Amazon ECS マネージドインスタンスは、PCI-DSS、HIPAA、FedRAMP など、Amazon ECS と同じ AWS 保証プログラムの対象です。
- FIPS エンドポイント - Amazon ECS マネージドインスタンスは、キャパシティプロバイダーレベルで FIPS エンドポイント設定をサポートします。アカウントレベルの設定を使用する Fargate とは異なり、FIPS はインスタンスごとの設定であるため、Amazon ECS マネージドインスタンスはキャパシティプロバイダーごとの設定を使用します。キャパシティプロバイダーを作成または更新するときに FIPS を設定します。
- カスタマーマネージドキー – 暗号化用のカスタマーマネージドキーなど、コンプライアンスの達成に必要なセキュリティ機能に対応します。

Amazon ECS マネージドインスタンスにおける FIPS-140 関連の考慮事項

Amazon ECS マネージドインスタンスで FIPS-140 コンプライアンスを使用する場合は、次の点を考慮してください。

- FIPS-140 準拠のマネージドインスタンス AMI を利用できるのは AWS GovCloud (US) リージョンのみです。
- Amazon ECS マネージドインスタンスは FIPS-140-3 をサポートします。
- AWS GovCloud (US) リージョンでは、FIPS-140 コンプライアンスがデフォルトで有効になっています。FIPS コンプライアンスなしでワークロードを実行する必要がある場合は、マネージドインスタンスキャパシティプロバイダーの設定で FIPS コンプライアンスを無効にします。
- FIPS-140 コンプライアンスのため、タスクの `cpuArchitecture` は `X86_64` にする必要があります。

Amazon ECS マネージドインスタンスで FIPS を無効にする

デフォルトで、AWS GovCloud (US) リージョン内の Amazon ECS マネージドインスタンスキャパシティプロバイダーは FIPS 準拠の AMI を起動します。FIPS-140 コンプライアンスの無効化は、新しい Amazon ECS マネージドインスタンスキャパシティプロバイダーを作成するときに選択します。FIPS コンプライアンスなしで新しいキャパシティプロバイダーを作成するには、次の手順を実行します。

1. キャパシティプロバイダーの FIPS-140 コンプライアンスを無効にします。

```
aws ecs create-capacity-provider \
    --cluster cluster-name \
```

```
    --name capacity-provider-name \
    --managed-instances-provider '{
        "infrastructureRoleArn": "infrastructure-role-arn",
        "instanceLaunchTemplate": {
            "ec2InstanceProfileArn": "instance-profile-arn",
            "fipsEnabled": false,
            "networkConfiguration": {
                "subnets": ["subnet-id"],
                "securityGroups": ["security-group-id"]
            }
        }
    }'
```

2. オプションで、ECS Exec を使用して次のコマンドを実行し、キャパシティプロバイダーの FIPS-140 コンプライアンスステータスを検証できます。

   cluster-name をクラスターの名前に、*task-id* をタスクの ID または ARN に、*container-name* をコマンドを実行するタスク内のコンテナ名に置き換えます。

   戻り値「1」は、FIPS が使用されていることを示します。

```
aws ecs execute-command \
    --cluster cluster-name \
    --task task-id \
    --container container-name \
    --interactive \
    --command "cat /proc/sys/crypto/fips_enabled"
```

セキュリティに関する考慮事項

Amazon ECS マネージドインスタンスを使用する場合は、いくつかの重要なセキュリティ上の考慮事項を理解して計画する必要があります。これらの考慮事項は、ワークロードアーキテクチャとセキュリティ要件について情報に基づいた意思決定を行うのに役立ちます。

マルチタスクセキュリティモデル

Amazon ECS マネージドインスタンスでのデフォルトのマルチタスク配置モデルは、Fargate の単一タスク分離とは異なります。

- 共有インスタンスリソース – 複数のタスクが同じインスタンス上で実行され、同じインスタンスまたは同じ ECS クラスターで実行されている他のタスクの脆弱性にタスクが晒される可能性があります。
- シングルタスクオプション – VM レベルのセキュリティ分離境界を持つデフォルトの Fargate セキュリティモデルを必要とする顧客のために、シングルタスクモードを使用するように Amazon ECS マネージドインスタンスを設定することができます。
- コストとセキュリティのトレードオフ – マルチタスクモードはコストの最適化とタスクスタートアップ時間の短縮化を実現し、一方で、シングルタスクモードはより強力な分離を提供します。

インスタンスの中断処理

Amazon ECS マネージドインスタンスを使用する際は、中断を許容するようにアプリケーションを設計することが重要です。

- 中断耐性 – 基盤となるサービスまたはタスクの中断を許容するアプリケーションで Amazon ECS マネージドインスタンスを使用します。
- サービスベースのワークロード – タスクの自動置き換えに Amazon ECS サービスを使用するか、独立型タスクで 14 日以内の制御および制限された期間でワークロードを実行します。
- 段階的なシャットダウン – タスクシャットダウンの猶予期間を設定して、中断の影響を制御します。

データアクセスとプライバシー

Amazon ECS マネージドインスタンスは、厳格なデータアクセスコントロールを維持します。

- 顧客データアクセス禁止 – AWS はマネージドインスタンスのライフサイクルとインスタンス上のタスクの配置を制御しますが、AWS はマネージドインスタンスにログインしたり、顧客データにアクセスしたりすることはできません。
- メトリックとログに限定 – AWS は、Amazon ECS マネージドインスタンス機能の提供に必要なメトリックと関連ログのみをキャプチャします。
- 隔離されたセキュリティモデル – セキュリティモデルでは管理アクセスを禁止することで、人為的ミスや改ざんの可能性を排除します。

セキュリティのベストプラクティス

Amazon ECS マネージドインスタンスを使用する場合は、次に挙げるベストプラクティスに従います。

- セキュリティモデルを評価する – セキュリティ要件、特にマルチタスク配置モデルに基づいて、Amazon ECS マネージドインスタンスの採用を意識的に決定します。
- 必要に応じてシングルタスクモードを使用する – ワークロードでより強力な分離が必要な場合は、Amazon ECS マネージドインスタンスがシングルタスクモードを使用するように設定します。
- 特権機能を最小限に抑える – 絶対に必要な場合にのみ特権的 Linux 機能を有効にし、関連するセキュリティリスクを理解します。
- 中断を計画する – 特に 14 日間のインスタンス最大存続期間を考慮して、インスタンスの置き換えを段階的に処理するようにアプリケーションを設計します。
- メンテナンスウィンドウを設定する – EC2 イベントウィンドウを使用して、インスタンスの置き換えが発生するタイミングを制御することで、ワークロードへの影響を最小限に抑えます。
- モニタリングと監査 – Amazon ECS マネージドインスタンスの設定を定期的に確認し、セキュリティ関連のイベントや変更がないかモニタリングします。

Amazon ECS マネージドインスタンスの VPC 暗号化コントロールの有効化

Amazon ECS マネージドインスタンスは、VPC 暗号化コントロールをサポートしています。これは、セキュリティとコンプライアンスの機能であり、リージョン内にある VPC 内 および VPC 間における転送時の暗号化を監視して適用するための一元化されたコントロールを提供します。サブネットで VPC 暗号化コントロールが有効化されていると、Amazon ECS マネージドインスタンスのカスタムキャパシティプロバイダーで転送時の暗号化をサポートするインスタンスタイプを指定できるため、Amazon ECS マネージドインスタンスのワークロードが転送時の暗号化を使用して実行されることが保証されます。

前提条件

開始するには、以下が必要です。

- サブネットで転送時の暗号化が有効化されている VPC。詳細については、VPC 暗号化コントロールに関するドキュメントを参照してください。

- Amazon ECS マネージドインスタンスのカスタムキャパシティプロバイダー。詳細については、「Amazon ECS マネージドインスタンスの設計」を参照してください。

互換性のあるインスタンスタイプを特定する

Amazon EC2 インスタンスタイプは、次の 2 つの要件を満たす必要があります。

1. 転送時の VPC 暗号化をサポートする - 次の AWS CLI コマンドを使用して、転送時の暗号化をサポートする Amazon EC2 インスタンスタイプを一覧表示します。

```
aws ec2 describe-instance-types \
    --filters Name=network-info.encryption-in-transit-supported,Values=true \
    --query "InstanceTypes[*].[InstanceType]" \
    --output text | sort
```

2. Amazon ECS マネージドインスタンスでサポートされる - Amazon ECS マネージドインスタンスでサポートされるすべての Amazon EC2 インスタンスタイプは、「Amazon ECS マネージドインスタンスのインスタンスタイプ」に記載されています。

追加の要件 (特定の CPU、メモリ、アーキテクチャ上のニーズなど) がある場合は、互換性のあるインスタンスタイプをワークロードの要件に基づいてさらにフィルタリングします。

VPC 暗号化をサポートするクラスターを作成する

転送時の VPC 暗号化向けに Amazon ECS マネージドインスタンスを設定する:

1. 新しいクラスターを作成して、インフラストラクチャに [Fargate インスタンスとマネージドインスタンス] を選択します。
2. [カスタムを使用 – アドバンスト] を選択し、追加の設定パラメータにアクセスします。
3. [許可されるインスタンスタイプ] には、転送時の VPC 暗号化をサポートする特定のインスタンスタイプのみを追加します。

Amazon ECS マネージドインスタンスをこのように設定すると、転送時の VPC 暗号化をサポートする Amazon EC2 インスタンスタイプのみを起動するようになります。

考慮事項

- バーストパフォーマンスインスタンス - T3、T3a、および T4g インスタンスタイプは、転送時の VPC 暗号化をサポートしておらず、強制モードで暗号化制御が有効化されているサブネットで使用することはできません。
- モードの移行 - 実行中のすべてのインスタンスが転送時の VPC 暗号化をサポートしている場合に限り、VPC サブネットをモニタリングモードから強制モードに移行できます。
- タスクの起動失敗 - 強制モードでは、転送時の暗号化をサポートしないインスタンスタイプを指定した場合にタスクの起動が失敗します。

トラブルシューティング

強制モードでタスクの起動が失敗する

タスクの起動が失敗する場合は、上述の AWS CLI コマンドを使用して、指定されたすべてのインスタンスタイプが転送時の VPC 暗号化をサポートしていることを確認します。

強制モードに移行できない

コンソールまたは `GetVpcResourcesBlockingEncryptionEnforcement` コマンドを使用して、転送時の暗号化を強制していないリソースを特定します。

VPC 暗号化コントロールの詳細については、VPC 暗号化コントロールに関するドキュメントを参照してください。

Amazon ECS マネージドインスタンスのインフラストラクチャの最適化

Amazon ECS マネージドインスタンスは、キャパシティプロバイダの設定と現在のワークロード需要に基づいて、適切なサイズの EC2 インスタンスを自動的に割り当てることで、コンテナ化されたアプリケーションが展開された瞬間から適切なコンピューティングリソースを確保します。アプリケーションのトラフィックパターンが進化し、ワークロード要件が時間の経過とともに変化するにつれて、Amazon ECS マネージドインスタンスは、現在のニーズに合わせてインスタンスサイズを自律的に調整し、最適な設定から逸脱したインスタンスを積極的に置き換え、コスト効率、アプリケーションパフォーマンス、システムの信頼性を動的に調整することで、インフラストラクチャを継続的にモニタリングおよび最適化します。このリソース管理システムは手動の介入なしで動作するため、アプリケーションの高可用性を維持しながらインフラストラクチャコストを削減できます。

インフラストラクチャの最適化には次の利点があります。

- コスト最適化 – リソース使用率を最大化し、余剰能力を排除することで、インフラストラクチャコストを削減します
- パフォーマンスの改善 – リソース要件とパフォーマンス特性に基づいてワークロード配置を最適化します
- 運用の簡素化 – 手動の介入を必要とせずに、複雑なリソース管理の決定を自動化します
- 信頼性の向上 – 自律的なワークロード分散と動作状態のモニタリングを通じて高可用性を維持します

Amazon ECS マネージドインスタンスは、効率を最大化しコストを削減するために、2 種類のインフラストラクチャ最適化を実行します。

非稼働インスタンスの検出

実行中のタスクがない EC2 インスタンスを特定して削除することで、未使用の能力から不要なインフラストラクチャコストを排除します。非稼働状態のインスタンスが検出されると、最適化プロセスはコンテナインスタンスを DEREGISTERING として印を付け、基盤となる EC2 インスタンスを安全に終了するクリーンアップシーケンスを開始します。

使用率の低いインスタンス検出

インスタンス間にわたるタスク分散を分析し、リソース割り当ての改善機会を特定します。タスクが複数のインスタンス間で最適に実行されていない場合、Amazon ECS マネージドインスタンスはワークロードをより少なく、より効率的に使用されているインスタンスに統合し、パフォーマンスを維持しながら全体的なコストを削減します。最適化プロセスでは、使用率の低いコンテナインスタンスを DRAINING として印を付けます。これにより、タスクの置き換えが開始され、ワークロードが既存または新しい、より効率的なインスタンスに移行されます。すべてのタスクが安全に移行されると、インスタンスは DEREGISTERING 状態に移行し、クリーンアップされます。この最適化は、サービスタスクを実行しているインスタンスに適用され、サービスの最小タスク制限と最大タスク制限を尊重し、スタートビフォーストップのデプロイ動作を優先し、ドレイニングプロセス全体を通してタスク保護設定を維持することで、安全な統合を実現します。ECS マネージドインスタンスは独立型タスクを置き換えないため、独立型タスクを実行しているインスタンスは最適化対象とはみなされません。

> **Important**
>
> 現在、使用率の低いインスタンス検出はスポットインスタンスではサポートされていません。Amazon ECS マネージドインスタンスでスポットインスタンスを使用する場合、インフラストラクチャ最適化機能はアイドル状態のインスタンス (実行中のタスクがないインスタンス) を検出して削除します。ただし、使用率の低いスポットインスタンスからワークロードを統合しません。

これらの最適化は連携して、アプリケーションの可用性に影響を与えることなく、自動的に無駄を排除し、リソース使用率を改善することで、インフラストラクチャが実際のワークロードの需要に継続的に適応することを確実にします。どちらのメカニズムも、タスクとインスタンスのライフサイクルイベントに応答するイベント駆動型モニタリングを使用して、最適化の機会をリアルタイムで特定します。Amazon ECS マネージドインスタンスは、コンテナインスタンス上で最後のタスクが停止したときに検出し、コスト最適化のため、非稼働状態になる見込みを提示します。使用率の低いインスタンスの場合、タスクの停止または新しいインスタンスの起動によって分析が開始され、ワークロード統合とリソース効率改善の機会を特定します。

ScaleInAfter

どちらのインフラストラクチャの最適化も、実行中のインスタンスを終了して使用率を向上し、コストを削減する機会を探します。Amazon ECS マネージドインスタンスのキャパシティプロバイダ設定の ScaleInAfter 設定を使用して、これらのアクションのタイミングを制御できます。これは、非稼働状態のインスタンスと使用率の低いインスタンスの両方に適用されます。ScaleInAfter では、インスタンスが非稼働状態または使用率の低い状態になるときから、Amazon ECS マネージドインスタンスがインフラストラクチャの最適化を開始するときまでの遅延を秒単位で指定できます。遅延は 0 ~ 3600 秒の間で設定できます。-1 を指定してインフラストラクチャの最適化を無効にすることもできます。

非稼働状態のインスタンス

- ECS は、最後のタスクが停止した後、インスタンスの登録を解除する前に、指定された期間待機します。
- 待機期間中に新しいタスクが開始された場合、インスタンスは非稼働状態とみなされなくなり、停止はキャンセルされます。

使用率の低いインスタンス

- ECS は、インスタンスが十分に活用されない結果となるタスク停止イベントの後、インスタンスをドレイニングする前に、指定された期間待機します。

- 待機期間中に新しいタスクが起動、または既存のタスクが特定のインスタンスで停止された場合、タイマーは最新のタスク停止時間または新しいタスク作成時間からリセットされ、Amazon ECS マネージドインスタンスは非効率について再評価し、新しい待機期間が終了した後に必要に応じてアクションを実行します。

この設定はオプションです。指定しない場合、ECS マネージドインスタンスは ECS マネージドインスタンスのデフォルト設定に基づいて最適なタイミングを自動的に決定します。

Amazon ECS マネージドインスタンスの GPU 自動修復

Amazon ECS は、GPU ハードウェアを使用する Amazon ECS マネージドインスタンスの NVIDIA GPU ヘルスをモニタリングします。Amazon ECS は GPU ハードウェアの障害を検出すると、障害が発生しているインスタンスを自動的に置き換えることができます。GPU 自動修復は、Amazon ECS マネージドインスタンスではデフォルトで有効になっています。

仕組み

Amazon ECS は NVIDIA Data Center GPU Manager (DCGM) を使用して、GPU ハードウェアがあるマネージドインスタンスの NVIDIA GPU ヘルスをモニタリングします。DCGM が重大な GPU 障害を報告すると、Amazon ECS はそのインスタンスを異常としてマークします。

GPU 自動修復を有効にすると、Amazon ECS は異常インスタンスをスタートビフォーストップワークフローを使用して置き換えます。

1. Amazon ECS は異常インスタンスを DRAINING に設定します。新しいタスクはそのインスタンスには配置されません。
2. Amazon ECS は代替インスタンスをプロビジョニングします。
3. Amazon ECS は既存のタスクを正常に停止させることができます。Amazon ECS はそのインスタンス上のタスクのタスク停止タイムアウトを優先します。
4. ドレイン期間が終了すると、Amazon ECS は異常インスタンスを終了します。

Amazon ECS レートで修復アクションを制限することで、置き換えのカスケードを防ぎます。キャパシティープロバイダーに属するインスタンスのうち、一度に使用できる量は 20% までです。キャパシティープロバイダーに 9 つ未満のインスタンスがある場合、一度に最大 1 つのインスタンスのみがドレインされます。

GPU ヘルスのモニタリング

`DescribeContainerInstances` API を使用して GPU ヘルスを確認できます。詳細については、「Amazon ECS コンテナインスタンスの正常性をモニタリングする」を参照してください。Amazon ECS コンテナインスタンスのヘルス変更イベント を通じて GPU ヘルスの変化をモニタリングすることもできます。

モニタリングされた XID エラーコード

Amazon ECS は、次の NVIDIA Xid エラーコードをモニタリングします。Amazon ECS は、これらのエラーのいずれかを検出すると、そのインスタンスを異常としてマークし、インスタンスを置き換えます。

Xid	説明
46	GPU が処理を停止した
48	ダブルビット ECC エラー
54	補助電源コネクタが接続されていない
62	内部マイクロコントローラーの停止
64	GPU メモリの再マッピングの失敗
74	NVLink エラー
79	GPU がバスから落下した
95	メモリ隔離エラー
109	コンテキストスイッチのタイムアウト
110	GPU がバスから消えた
136	GPU メモリページの廃止制限を超えた
140	ECC 復旧不能エラー
142	修正不可能なエラーにより GPU メモリページが廃止された

Xid	説明
143	修正可能なエラーしきい値により GPU メモリページが廃止された
151	GPU から CPU への相互接続エラー
155	GPU NVLink Flit CRC エラー
156	GPU NVLink レーンエラー
158	GPU InfoROM が破損した

XID エラーの詳細については、「NVIDIA GPU のデプロイおよび管理ドキュメント」の「Xid エラー」を参照してください。個々の XID メッセージの詳細については、「NVIDIA GPU Deployment and Management Documentation」の「Understanding Xid Messages」を参照してください。

自動修復の無効化

GPU 自動修復は、Amazon ECS マネージドインスタンスではデフォルトで有効になっています。GPU 自動修復を無効にするには、キャパシティープロバイダーを作成または更新するときに、`autoRepairConfiguration` で `actionsStatus` を `DISABLED` に設定します。キャパシティープロバイダーを作成または更新するときに、Amazon ECS コンソールで GPU 自動修復を無効にすることもできます。

GPU 自動修復が無効になっている場合、Amazon ECS は GPU ヘルスを引き続きモニタリングしますが、異常インスタンスの自動置換は行いません。

> **Note**
>
> GPU 自動修復を無効にすると、Amazon ECS マネージドデーモン自動修復も無効になります。詳細については、「Amazon ECS マネージドデーモン自動修復」を参照してください。

GPU 自動修復を無効にする方法

```
aws ecs update-capacity-provider \
    --name my-gpu-capacity-provider \
    --managed-instances-provider '{
```

```
        "infrastructureRoleArn": "arn:aws:iam::111122223333:role/
ecsInfrastructureRole",
        "instanceLaunchTemplate": {
            "ec2InstanceProfileArn": "arn:aws:iam::111122223333:instance-profile/
ecsInstanceRole",
            "networkConfiguration": {
                "subnets": ["subnet-0123456789abcdef0"],
                "securityGroups": ["sg-0123456789abcdef0"]
            }
        },
        "autoRepairConfiguration": {
            "actionsStatus": "DISABLED"
        }
    }'
```

GPU 自動修復を有効にする方法

```
aws ecs update-capacity-provider \
    --name my-gpu-capacity-provider \
    --managed-instances-provider '{
        "infrastructureRoleArn": "arn:aws:iam::111122223333:role/
ecsInfrastructureRole",
        "instanceLaunchTemplate": {
            "ec2InstanceProfileArn": "arn:aws:iam::111122223333:instance-profile/
ecsInstanceRole",
            "networkConfiguration": {
                "subnets": ["subnet-0123456789abcdef0"],
                "securityGroups": ["sg-0123456789abcdef0"]
            }
        },
        "autoRepairConfiguration": {
            "actionsStatus": "ENABLED"
        }
    }'
```

設定を確認する方法

```
aws ecs describe-capacity-providers \
    --capacity-providers my-gpu-capacity-provider
```

AWS Fargate から Amazon ECS マネージドインスタンスへの移行

既存のワークロードを Fargate から Amazon ECS マネージドインスタンスに移行できます。この移行により、AWS マネージドインフラストラクチャを維持しながら、Amazon EC2 インスタンスタイプ、キャパシティ予約、および高度な機能のすべてにアクセスできます。

移行に関する考慮事項

Fargate から Amazon ECS マネージドインスタンスに移行するときは、次の考慮事項に注意してください。

タスク互換性

Fargate 用に設定された既存のタスク定義は、ほとんどが Amazon ECS マネージドインスタンスと互換性があります。タスク定義の違いについては、「Amazon ECS マネージドインスタンスの Amazon ECS タスク定義の違い」を参照してください。

セキュリティモデルの変更

Amazon ECS マネージドインスタンスでは、デフォルトでインスタンスごとに複数のタスクが許可されます。ワークロードで、より強力な分離が必要な場合は、シングルタスクモードを有効にすることを検討してください。

インスタンスのライフサイクル

Amazon ECS マネージドインスタンスの最大存続期間は 14 日間です。タスクの置き換えを計画し、Amazon ECS サービスを使用して自動タスク管理を行います。

料金の変更

Amazon ECS マネージドインスタンスでは、Fargate のようにタスクごとのリソースではなく、インスタンス全体の料金および管理費が発生します。

メンテナンスウィンドウ

Amazon EC2 イベントウィンドウを使用してメンテナンスウィンドウを設定し、Amazon ECS マネージドインスタンスをパッチ適用のために置き換えるタイミングを制御します。

前提条件

Amazon ECS マネージドインスタンスに移行する前に、次の条件を確認してください。

- プラットフォームバージョン 1.4.0 以降を使用した既存の Fargate タスク
- Amazon ECS マネージドインスタンスに必要な IAM ロールがあること。これには、以下が含まれます。
  - インフラストラクチャロール – Amazon ECS がユーザーに代わって AWS サービスを呼び出し、Amazon ECS マネージドインスタンスインフラストラクチャを管理できるようにします。

    詳細については、「Amazon ECS インフラストラクチャ IAM ロール」を参照してください。
  - インスタンスプロファイル – マネージドインスタンスで実行されている Amazon ECS コンテナエージェントと Docker デーモンへのアクセス許可を提供します。

    詳細については、「Amazon ECS マネージドインスタンスのインスタンスプロファイル」を参照してください。
- Fargate と Amazon ECS マネージドインスタンスのセキュリティモデルの違いを理解していること

> **Important**
>
> Amazon ECS マネージドインスタンスは、Fargate とは異なるセキュリティモデルを使用しています。デフォルトでは、複数のタスクを同じインスタンスで実行できます。これにより、タスクが他のタスクの脆弱性にさらされる可能性があります。移行前にセキュリティ要件を確認してください。

ステップ 1: Amazon ECS マネージドインスタンスを使用するようにクラスターを更新する

キャパシティプロバイダを作成します。Amazon ECS マネージドインスタンスでキャパシティプロバイダを作成すると、指定されたクラスター内だけで使用可能になります。

詳細については、「Amazon ECS マネージドインスタンスのキャパシティプロバイダーを作成する」を参照してください。

ステップ 2: Amazon ECS マネージドインスタンス機能に合わせてタスク定義を更新する

Amazon ECS マネージドインスタンス用に requiresCapabilities を追加してタスク定義を更新します。

詳細については、「コンソールを使用した Amazon ECS タスク定義の更新」を参照してください。

ステップ 3: Amazon ECS マネージドインスタンスのキャパシティプロバイダを使用するようにサービスを更新する

Amazon ECS マネージドインスタンスのキャパシティプロバイダを使用するように既存の Amazon ECS サービスを更新します。

詳細については、「キャパシティプロバイダーを使用するように Amazon ECS サービスを更新する」を参照してください。

ステップ 4: 独立型タスクを移行する

独立型タスクの場合は、タスク実行時に Amazon ECS マネージドインスタンスのキャパシティプロバイダを指定します。

詳細については、「Amazon ECS タスクとしてのアプリケーションの実行」を参照してください。

Amazon EC2 から Amazon ECS マネージドインスタンスへの移行

既存のワークロードを Amazon EC2 から Amazon ECS マネージドインスタンスに移行します。この移行により、AWS マネージドインフラストラクチャを維持しながら、Amazon EC2 インスタンスタイプ、キャパシティ予約、および高度な機能のすべてにアクセスできます。

移行に関する考慮事項

Amazon EC2 から Amazon ECS マネージドインスタンスに移行するときは、次の考慮事項に注意してください。

タスク互換性

Amazon EC2 用に設定された既存のタスク定義は、ほとんどが Amazon ECS マネージドインスタンスと互換性があります。タスク定義の違いのリストについては、「Amazon ECS マネージドインスタンスの Amazon ECS タスク定義の違い」を参照してください。

セキュリティモデルの変更

Amazon ECS マネージドインスタンスでは、デフォルトでインスタンスごとに複数のタスクが許可されます。ワークロードで、より強力な分離が必要な場合は、シングルタスクモードを有効にすることを検討してください。

インスタンスのライフサイクル

Amazon ECS マネージドインスタンスの最大存続期間は 14 日間です。タスクの置き換えを計画し、Amazon ECS サービスを使用して自動タスク管理を行います。

料金の変更

Amazon ECS マネージドインスタンスでは、インスタンス全体の料金と、AWS がインフラストラクチャ管理のオーバーヘッドを処理するための管理費が発生します。

メンテナンスウィンドウ

Amazon EC2 イベントウィンドウを使用してメンテナンスウィンドウを設定し、Amazon ECS マネージドインスタンスをパッチ適用のために置き換えるタイミングを制御します。

前提条件

Amazon ECS マネージドインスタンスに移行する前に、次の条件を確認してください。

- Amazon ECS マネージドインスタンスに必要な IAM ロールがあること。これには、以下が含まれます。
  - インフラストラクチャロール – Amazon ECS がユーザーに代わって AWS サービスを呼び出し、Amazon ECS マネージドインスタンスインフラストラクチャを管理できるようにします。

    詳細については、「Amazon ECS インフラストラクチャ IAM ロール」を参照してください。
  - インスタンスプロファイル – マネージドインスタンスで実行されている Amazon ECS コンテナエージェントと Docker デーモンへのアクセス許可を提供します。

    詳細については、「Amazon ECS マネージドインスタンスのインスタンスプロファイル」を参照してください。
- Amazon EC2 と Amazon ECS マネージドインスタンスのセキュリティモデルの違いを理解していること

ステップ 1: Amazon ECS マネージドインスタンスを使用するようにクラスターを更新する

キャパシティプロバイダを作成します。Amazon ECS マネージドインスタンスでキャパシティプロバイダを作成すると、指定されたクラスター内だけで使用可能になります。

詳細については、「Amazon ECS マネージドインスタンスのキャパシティプロバイダーを作成する」を参照してください。

ステップ 2: Amazon ECS マネージドインスタンス機能に合わせてタスク定義を更新する

Amazon ECS マネージドインスタンス用に requiresCapabilities を追加してタスク定義を更新します。

詳細については、「コンソールを使用した Amazon ECS タスク定義の更新」を参照してください。

ステップ 3: Amazon ECS マネージドインスタンスのキャパシティプロバイダを使用するようにサービスを更新する

Amazon ECS マネージドインスタンスのキャパシティプロバイダを使用するように既存の Amazon ECS サービスを更新します。

詳細については、「キャパシティプロバイダーを使用するように Amazon ECS サービスを更新する」を参照してください。

ステップ 4: 独立型タスクを移行する

独立型タスクの場合は、タスク実行時に Amazon ECS マネージドインスタンスのキャパシティプロバイダを指定します。

詳細については、「Amazon ECS タスクとしてのアプリケーションの実行」を参照してください。

Amazon ECS マネージドデーモン

Amazon ECS マネージドデーモンを使用すると、セキュリティ、オブザーバビリティ、ネットワークエージェントなどのソフトウェアエージェントを Amazon ECS マネージドインスタンスのコンテナインフラストラクチャ全体にデプロイおよび管理できます。マネージドデーモンは、デーモンのライフサイクル管理をアプリケーションオペレーションから切り離します。ワークロードを再デプロイしたり、サービス全体で変更を調整したりすることなく、エージェントを個別にデプロイ、更新、モニタリングできます。

デーモンの重要度

デーモンの障害によって Amazon ECS がコンテナインスタンスをドレインするかどうかに基づいて、各マネージドデーモンをクリティカルまたは非クリティカルとして設定できます。

- クリティカル - デフォルト。Amazon ECS は、アプリケーションタスクをインスタンスに配置する前にデーモンタスクを開始し、デーモンタスクが停止または異常になると、Amazon ECS はコンテナインスタンスをドレインして置き換えます。
- 非クリティカル - `critical` パラメータを `false` に設定します。デーモンタスクは、コンテナインスタンスの状態とは無関係に動作します。非クリティカルなデーモンタスクが失敗、停止、または異常になった場合、Amazon ECS はコンテナインスタンスをアクティブに保ちます。既存のアプリケーションタスクは実行され続け、Amazon ECS はそのインスタンスに新しいアプリケーションタスクを配置し続けます。非クリティカルなデーモンはインスタンス登録をブロックしないため、デーモンの起動に失敗してもアプリケーションタスクをすぐに配置できます。

マネージドデーモンの仕組み

マネージドデーモンを使用するには、まずデーモンタスク定義を登録します。デーモンタスク定義は、デーモンを形成するコンテナを記述するテンプレートです。デーモンタスク定義を登録したら、デーモンを作成し、クラスターと 1 つまたは複数の Amazon ECS マネージドインスタンスキャパシティプロバイダーに関連付けます。その後、Amazon ECS は、それらのキャパシティプロバイダーを介してプロビジョニングされたすべての Amazon EC2 インスタンスでちょうど 1 つのデーモンタスクが実行されるようにします。

デーモンはインスタンスを個別に起動しません。Amazon ECS マネージドインスタンスキャパシティプロバイダーでアプリケーションタスクを実行すると、Amazon ECS は Amazon EC2 インスタンスをプロビジョニングし、デーモンタスクを起動してから、アプリケーションタスクを `RUNNING`

に移行します。Amazon ECS は、重要度に関係なくすべてのデーモンにこの順序を使用するため、アプリケーションがリクエストの処理を開始する前に、ログ記録、トレース、メトリクス収集などのクロスカット機能が動作します。

重要度は、デーモンタスクが開始しない場合、または後で停止した場合に Amazon ECS が行う処理を制御します。クリティカルなデーモンの場合、開始に失敗したタスクにより、インスタンスがアクティブにならないようにします。インスタンスがアクティブになった後にタスクが停止または異常になると、Amazon ECS はインスタンスをドレインして置き換えます。この自動修復動作により、手動による介入なしに信頼性の高いデーモンカバレッジが維持されます。

非クリティカルなデーモンは、インスタンスをブロックしたり、ドレインをトリガーしたりすることはありません。デーモンタスクの開始に失敗してもインスタンスはアクティブになり、デーモンタスクが停止したり後で異常になったりしてもアクティブのままになります。アプリケーションタスクは実行され続け、Amazon ECS はインスタンスへの新しいアプリケーションタスクの配置を続けます。

デーモンを新しいタスク定義リビジョンに更新すると、Amazon ECS は関連付けられたキャパシティプロバイダー内のすべてのインスタンス全体でローリングデプロイを実行します。デプロイ中、Amazon ECS は設定可能な割合のインスタンスを同時にドレインし、更新されたデーモンで置換インスタンスをプロビジョニングし、Amazon ECS サービスタスクを自動的に置き換えます。Amazon ECS は組み込みのサーキットブレーカー保護を提供します。ベイク時間と CloudWatch アラームを設定して、Amazon ECS がすべてのインスタンスを更新した後にデプロイをモニタリングし、問題が発生した場合に自動的にロールバックできるようにします。Amazon ECS は、クリティカルなデーモンと非クリティカルなデーモンの両方について、デーモンタスクの開始に失敗したときに EventBridge イベントを発行し、サービスアクションログを記録します。

主な利点

- 分離されたライフサイクル管理 - アプリケーションのデプロイとは別にデーモンを更新します。
- カバレッジの保証 - Amazon ECS はすべてのインスタンスでアプリケーションタスクの前にデーモンタスクを開始するため、アプリケーションがリクエストの処理を開始する前にクロスカット機能が動作します。クリティカルなデーモンの場合、起動に失敗したデーモンタスクによってインスタンスの登録も行われなくなるため、インスタンスがデーモンなしでアプリケーションタスクを処理することはありません。
- 信頼性の高いバージョン更新 - デーモンバージョンを更新すると、Amazon ECS は、すべてのインスタンスがターゲットリビジョンを実行するように、組み込みのサーキットブレーカー保護と自動ロールバックを使用して、関連付けられたキャパシティプロバイダー内のすべてのインスタンス全体でそのバージョンをロールアウトします。

- リソース使用率の向上 - インスタンスごとに 1 つのデーモンタスクを実行すると、タスクごとのサイドカーモデルがなくなり、クラスター全体のリソースオーバーヘッドが削減されます。
- インスタンスの自動修復 – クリティカルなデーモンでは、デーモンタスクが停止または異常になった場合、Amazon ECS は自動的にそのコンテナインスタンスをドレインして置き換えます。これにより、手動による介入なしで信頼性の高いデーモンカバレッジが維持されます。
- ワークロードの継続性 - 非クリティカルなデーモンでは、デーモンタスクが失敗、停止、または異常になった場合でも、アプリケーションワークロードを中断することなく実行し続けます。

> **Note**
>
> Amazon ECS は、Amazon EC2 起動タイプの Amazon ECS サービスの DAEMON スケジューリング戦略を提供します。マネージドデーモンは、Amazon ECS マネージドインスタンス用に構築された新機能で、デーモンのデプロイを簡素化し、強化されたデーモンカバレッジの保証を提供します。

開始方法

使用を開始するには、コンテナイメージを指定するデーモンタスク定義を登録し、クラスター内の関連付けられたキャパシティプロバイダーでデーモンを作成します。AWS マネジメントコンソール、AWS CLI、CloudFormation、または AWS SDK を使用できます。手順については、「デーモンの作成と管理」を参照してください。

デーモンタスク定義

デーモンタスク定義は、デーモンのブループリントです。Amazon ECS マネージドデーモンは、標準の Amazon ECS タスク定義とは異なる専用のデーモンタスク定義リソースを使用します。デーモンタスク定義を登録して、デーモンのコンテナイメージ、リソース要件、および設定を指定します。

デーモンタスク定義の作成

AWS マネジメントコンソール または AWS CLI を使用してデーモンタスク定義を作成できます。

の使用AWS マネジメントコンソール

1. Amazon ECS コンソールを開きます。ナビゲーションペインで、[デーモンタスク定義] を選択します。

2. [新しいデーモンタスク定義の作成] を選択します。
3. [デーモンタスク定義ファミリー] では、一意の名前を入力します。名前には、最大 255 文字の英数字、ハイフン、およびアンダースコアを使用できます。
4. (オプション) [タスクロール] では、コンテナで実行されているアプリケーションにアクセス許可を付与する IAM ロールを選択します。コンテナが AWS API を呼び出さない場合は、これを空白のままにします。
5. (オプション) [タスク実行ロール] では、`ecsTaskExecutionRole` を選択します。Amazon ECS には、コンテナイメージをプルしてログを発行するためにこのロールが必要です。
6. (オプション) [タスクサイズ] では、デーモン用に予約する CPU とメモリを指定します。例えば、`0.25 vCPU` および `512 MB` メモリです。
7. コンテナの詳細を設定します:
   - [コンテナ名] - コンテナの名前を入力します。名前には、最大 255 文字の英数字、ハイフン、およびアンダースコアを使用できます。
   - [必須コンテナ] - [はい] を選択します。各デーモンタスク定義には、少なくとも 1 つの必須コンテナが必要です。必須コンテナに障害が発生した場合、Amazon ECS はタスク全体を停止します。
   - [イメージ URI] - Docker イメージ URI を入力します。Amazon ECR イメージを参照することも、Docker Hub または他のレジストリからパブリックイメージを使用することもできます。
8. (オプション) 必要に応じて、リソース割り当て、ヘルスチェック、環境変数、ログ収集、およびタグを設定します。
9. 設定を確認し、[作成] を選択します。

の使用AWS CLI

JSON ファイルを作成し、`register-daemon-task-definition` コマンドを使用して、デーモンタスク定義を登録します。

JSON ファイルの例を次に示します:

```
{
    "family": "my-daemon-task",
    "containerDefinitions": [
        {
            "name": "daemon-container",
            "image": "public.ecr.aws/docker/library/busybox:latest",
            "essential": true,
```

```
            "command": ["sh", "-c", "while true; do echo 'Daemon running'; sleep 30;
 done"],
            "memoryReservation": 512
        }
    ]
}
```

次のコマンドを実行して、デーモンタスク定義を登録します:

```
aws ecs register-daemon-task-definition --cli-input-json file://daemon-taskdef.json
```

デーモンタスク定義パラメータ

デーモンタスク定義を登録するときは、次のパラメータを使用できます。

必須パラメータ

- family - タスク定義のファミリー名。これにより、同じデーモンタスク定義の複数のリビジョンがグループ化されます。
- containerDefinitions - デーモンタスクのコンテナを記述する DaemonContainerDefinition オブジェクトの配列。

任意指定のパラメータ: 、、。

- cpu - 文字列としてのタスクレベルの CPU ユニット (例: "256")。
- memory - 文字列としてのタスクレベルのメモリ (MiB 単位) (例: "512")。
- taskRoleArn - デーモンタスクのコンテナにアクセス許可を付与する IAM ロールの ARN。
- executionRoleArn - Amazon ECS コンテナエージェントがユーザーに代わって AWS API コールを行うために使用する IAM ロールの ARN (例: Amazon ECR からイメージをプルする)。
- volumes - DaemonVolume オブジェクトの配列。デーモンボリュームは、host および sourcePath を使用したバインドマウントのみをサポートします。デーモンタスク定義は、Amazon EBS、Amazon EFS、FSx for Windows File Server、Docker ボリューム、または エフェメラルストレージをサポートしていません。
- tags - デーモンタスク定義にタグを付けるためのキーと値のペアの配列。
- pidMode - デーモンの PID 名前空間モードを制御します。
- ipcMode - デーモンの IPC 名前空間モードを制御します。

ネットワークモード

デーモンは、Amazon ECS が自動的に設定する特別な daemon_bridge ネットワークモードを使用します。デーモンタスク定義でネットワークモードを指定することはできません。インスタンス上のすべてのデーモンは、単一のネットワーク名前空間を共有し、静的デーモンブリッジ IP アドレス (IPv4 の場合は 169.254.172.2、IPv6 の場合は fd00:ec2::172:2) を介してローカルにアクセスできます。アプリケーションタスク (デーモン以外) は、追加のネットワーク設定なしで、このアドレスでデーモンと通信できます。

デーモンはアプリケーションタスクとは別のネットワーク名前空間で実行されるため、デーモン以外のタスクはデーモンとのポート競合について心配する必要はありません。ただし、すべてのデーモンタスクが同じ名前空間を共有するため、同じインスタンス上のデーモンを同じポートにバインドすることはできません。複数のデーモンをデプロイする場合は、各デーモンが一意のポートを使用していることを確認します。

特権機能

Amazon ECS マネージドインスタンスでは、デーモンはシステムレベルのオペレーションのための特権 Linux 機能をサポートしています。セキュリティエージェント、ネットワークモニタリングツール、オブザーバビリティエージェントは、多くの場合、正しく機能するためにカーネルレベルのアクセスを必要とします。

機能を付与するには、次の 2 つの方法があります:

フル特権モードでは、すべての Linux 機能がコンテナに付与されます。エージェントが広範なシステムアクセスを必要とする場合に使用します:

```
{
    "containerDefinitions": [{
        "name": "security-daemon",
        "image": "my-security-agent:latest",
        "privileged": true
    }]
}
```

個々の機能は、最小特権の原則に従って、デーモンに必要なアクセス許可のみを付与します。linuxParameters.capabilities フィールドを使用して、SYS_ADMIN、NET_ADMIN、SYS_PTRACE、BPF、PERFMON などの個々の機能を追加します。

```
{
```

```
    "containerDefinitions": [{
        "name": "monitoring-daemon",
        "image": "my-monitoring-agent:latest",
        "linuxParameters": {
            "capabilities": {
                "add": ["NET_ADMIN", "SYS_PTRACE"]
            }
        }
    }]
}
```

PID と IPC 名前空間の共有

Amazon ECS マネージドインスタンスでは、デーモンは PID および IPC 名前空間の共有に対応し、アプリケーションタスクプロセスまたは IPC リソースの可視性を必要とするセキュリティエージェント、モニタリングツール、オブザーバビリティデーモンなどのユースケースを可能にします。

`pidMode`

タイプ: 文字列

必須: いいえ

有効な値: none | shared

デフォルト: none

デーモンの PID 名前空間モード。none が指定されている場合、または値が提供されない場合、デーモンは他のタスクから分離された独自の PID 名前空間を使用して実行されます。shared が指定されている場合、デーモンはホスト PID 名前空間を結合し、pidMode: "host" を使用するデーモン以外のタスク、または pidMode: "shared" を使用する他のデーモンにアクセスできるようにします。

`ipcMode`

タイプ: 文字列

必須: いいえ

有効な値: none | shared

デフォルト: none

デーモンの IPC 名前空間モード。none が指定されている場合、または値が提供されない場合、デーモンは他のタスクから分離された独自の IPC 名前空間を使用して実行されます。shared が指定されている場合、デーモンはホスト IPC 名前空間を結合し、ipcMode: "host" を使用するデーモン以外のタスク、または ipcMode: "shared" を使用する他のデーモンにアクセスできるようにします。

次の例は、PID と IPC の両方の名前空間の共有が有効になっているデーモンタスク定義を示しています。

```
{
    "family": "my-monitoring-daemon",
    "pidMode": "shared",
    "ipcMode": "shared",
    "containerDefinitions": [
        {
            "name": "monitoring-agent",
            "image": "public.ecr.aws/my-company/monitoring-agent:latest",
            "essential": true
        }
    ]
}
```

PID 名前空間を、pidMode が shared に設定されているデーモンと共有するには、デーモン以外のタスクがタスク定義で pidMode: "host" を指定する必要があります。これにより、タスクのコンテナはデーモンのプロセスを表示することができ、プロセスのモニタリングやシグナル配信などのユースケースが可能になります。

IPC 名前空間を、ipcMode が shared に設定されているデーモンと共有するには、デーモン以外のタスクがタスク定義で ipcMode: "host" を指定する必要があります。これにより、タスクのコンテナは、共有メモリセグメントやメッセージキューなどのデーモンの IPC リソースにアクセスできるようになります。

ボリューム

マネージドデーモンは、sourcePath 仕様を持つホストボリュームを使用したバインドマウントをサポートします。これにより、デーモンコンテナはホストディレクトリをマウントして、基盤となる Amazon EC2 インスタンスのログ、メトリクス、およびシステム情報にアクセスできます。ログ収集エージェント、メトリクスエクスポーター、およびセキュリティスキャナーは、一般的にこれを使用してホストレベルのデータを可視化します。

デーモンタスク定義は、sourcePath でホストボリュームの仕様をサポートしています。デーモンボリュームは、基盤となるコンテナインスタンスのライフサイクルの間、保持されます。

> **Note**
>
> Amazon EBS ボリューム、Amazon EFS ボリューム、FSx for Windows File Server ボリューム、Docker ボリューム、およびエフェメラルストレージは、デーモンタスク定義ではサポートされていません。

> **Note**
>
> デーモンは、お互いの間のボリューム共有をサポートしていません。各デーモンは、独自の独立したボリューム設定で動作します。

次の例は、バインドマウントを持つログコレクターデーモンを示しています。

```
{
    "containerDefinitions": [{
        "name": "log-collector",
        "image": "fluent/fluentd:latest",
        "mountPoints": [{
            "sourceVolume": "var-log",
            "containerPath": "/var/log"
        }]
    }],
    "volumes": [{
        "name": "var-log",
        "host": {
            "sourcePath": "/var/log"
        }
    }]
}
```

サポートされているコンテナパラメータ

デーモンコンテナ定義は、次のパラメータをサポートしています:

- image (必須) - コンテナイメージ URI

- name (必須) - コンテナ名
- cpu - コンテナ用に予約されている CPU ユニット
- memory/memoryReservation - ハードメモリとソフトメモリの制限
- essential - コンテナが必須かどうか (デフォルト: true)
- command/entryPoint - コンテナコマンドまたはエントリポイントを上書きする
- environment/environmentFiles/secrets - 環境設定
- privileged - 特権モードでコンテナを実行する
- user - コンテナの実行ユーザー
- workingDirectory - コンテナ内の作業ディレクトリ
- readonlyRootFilesystem - ルートファイルシステムを読み取り専用としてマウントする
- mountPoints - ボリュームマウントポイント (ホストと sourcePath のみを使用するバインドマウント)
- logConfiguration - ログ記録設定 (awslogs、splunk、awsfirelens をサポート)
- healthCheck - コンテナのヘルスチェック設定
- dependsOn - コンテナスタートアップの依存関係
- ulimits - リソースの制限
- systemControls - カーネルパラメータ
- linuxParameters.capabilities - 追加またはドロップする Linux 機能
- linuxParameters.initProcessEnabled - コンテナで init プロセスを実行する
- repositoryCredentials - プライベートレジストリの認証情報
- restartPolicy - ECS コンテナ再起動ポリシー

デーモンの作成と管理

デーモンタスク定義を登録したら、デーモンを作成して Amazon ECS マネージドインスタンスキャパシティプロバイダーにデプロイします。Amazon ECS は、関連付けられたキャパシティプロバイダー内のすべての Amazon EC2 インスタンスに 1 つのデーモンタスクを自動的に配置し、デーモンのライフサイクルを管理します。

前提条件

デーモンを作成する前に、以下があることを確認します:

- Amazon ECS クラスター。
- クラスターに関連付けられた 1 つまたは複数の Amazon ECS マネージドインスタンスキャパシティプロバイダー。
- 登録されたデーモンタスク定義。

デーモンの作成

AWS マネジメントコンソール

1. Amazon ECS コンソールを開きます。ナビゲーションで [クラスター] を選択し、クラスターを選択します。
2. [デーモン] タブを選択し、[作成] を選択します。
3. [デーモンタスク定義ファミリー] では、ドロップダウンからデーモンタスク定義を選択します。
4. [デーモンタスク定義リビジョン] では、使用するリビジョンを選択します。最新のリビジョンを使用するには、空白のままにします。
5. [デーモン名] では、一意の名前を入力します。名前には、最大 255 文字の英数字、ハイフン、およびアンダースコアを使用できます。
6. [キャパシティプロバイダー] では、1 つまたは複数の Amazon ECS マネージドインスタンスキャパシティプロバイダーを選択します。これらは、デーモンタスクを実行するインスタンスを決定します。
7. (オプション) デプロイ設定を指定する:
   - [ドレイン率] - 更新中に同時にドレインするインスタンスの割合。デフォルト: 25。
   - [CloudWatch アラームを使用する] - オンにしてデプロイヘルスをモニタリングし、アラームがトリガーされた場合に自動的にロールバックします。
   - [ベイク時間] - Amazon ECS がすべてのインスタンスを新しいデーモンリビジョンに更新してからデプロイを完了するまで待機する分数。この期間中、Amazon ECS は CloudWatch アラームをモニタリングし、アラームがトリガーされるとデプロイを自動的にロールバックします。デフォルト: 0。
8. (オプション) [クリティカル] では、デーモンがインスタンスのヘルスにとって重要かどうかを選択します。
9. (オプション) タグを追加します。
10. (オプション) [ECS Exec を有効する] をオンにして、トラブルシューティングのためにデーモンコンテナでインタラクティブコマンドを実行します。

11設定を確認し、[作成] を選択します。

AWS CLI

デーモン設定を含む JSON ファイルを作成し、create-daemon コマンドを実行します。

JSON ファイルの例を次に示します:

```
{
    "clusterArn": "arn:aws:ecs:us-east-1:123456789012:cluster/my-daemon-cluster",
    "daemonName": "my-monitoring-daemon",
    "daemonTaskDefinitionArn": "arn:aws:ecs:us-east-1:123456789012:daemon-task-
definition/my-daemon-task:1",
    "capacityProviderArns": [
        "arn:aws:ecs:us-east-1:123456789012:capacity-provider/my-daemon-capacity-
provider"
    ],
    "critical": true
}
```

次のコマンドを実行してデーモンを作成します:

```
aws ecs create-daemon --cli-input-json file://create-daemon.json
```

必須フィールド:

- daemonName - デーモンの一意の名前。
- clusterArn - クラスターの ARN。
- daemonTaskDefinitionArn - デーモンタスク定義の ARN。
- capacityProviderArns - Amazon ECS マネージドインスタンスのキャパシティプロバイダー ARN の配列。

省略しても構わないフィールドです。

- deploymentConfiguration - デプロイ動作をカスタマイズする DaemonDeploymentConfiguration オブジェクト。
- critical - デーモンがインスタンスのヘルスにとって重要かどうか。
- tags - タグ付け用のキーと値のペア。

- propagateTags - タグ伝達設定。
- clientToken - べき等性トークン。

デーモンのデプロイの検証

デーモンを作成したら、AWS マネジメントコンソール または AWS CLI を使用してそのステータスを検証します。

AWS マネジメントコンソール

1. Amazon ECS コンソールを開きます。ナビゲーションで [クラスター] を選択し、クラスターを選択します。
2. [デーモン] タブを選択します。
3. デーモンに [アクティブ] ステータスが表示されていることを確認します。
4. [タスク] タブを選択して、各コンテナインスタンスで 1 つのデーモンタスクが実行されていることを確認します。

AWS CLI

デーモンステータスを検証するために、次のコマンドを実行します:

```
aws ecs list-daemons \
    --cluster-arn arn:aws:ecs:us-east-1:123456789012:cluster/my-daemon-cluster
```

```
aws ecs describe-daemons \
    --daemon-arn arn:aws:ecs:us-east-1:123456789012:daemon/my-daemon-cluster/my-monitoring-daemon
```

デーモンの更新

新しいタスク定義リビジョンでデーモンを更新するか、キャパシティプロバイダーを追加するには、AWS マネジメントコンソール または AWS CLI を使用します。これにより、すべてのインスタンス全体でローリングデプロイがトリガーされます。

AWS マネジメントコンソール

1. Amazon ECS コンソールを開きます。ナビゲーションで [クラスター] を選択し、クラスターを選択します。
2. [デーモン] タブを選択し、更新するデーモンを選択します。
3. [更新] を選択します。
4. [デーモンタスク定義リビジョン] では、新しいリビジョンを選択します。
5. (オプション) キャパシティプロバイダーまたはデプロイ設定を更新します。
6. [更新] を選択して、ローリングデプロイを開始します。

AWS CLI

`update-daemon` コマンドを実行します。

```
aws ecs update-daemon \
    --daemon-arn arn:aws:ecs:us-east-1:123456789012:daemon/my-daemon-cluster/my-monitoring-daemon \
    --daemon-task-definition-arn arn:aws:ecs:us-east-1:123456789012:daemon-task-definition/my-daemon-task:2 \
    --capacity-provider-arns arn:aws:ecs:us-east-1:123456789012:capacity-provider/my-daemon-capacity-provider \
    --no-critical
```

> **Important**
>
> UpdateDaemon リクエストでデーモン設定を指定すると、Amazon ECS はデフォルトの代わりに指定された設定を使用します。タグや、enable execute コマンドフラグなどのデーモン設定は、更新間で保持されません。UpdateDaemon への各呼び出しには、適用する構成設定の完全なセットを含める必要があります。リクエストから省略された設定は、デフォルト値に戻ります。`critical` パラメータのデフォルトは `true` であるため、UpdateDaemon リクエストから省略すると、デーモンは重要になります。

デーモンの削除

デーモンを削除するには、AWS マネジメントコンソール または AWS CLI を使用します。キャパシティプロバイダーまたはクラスターを削除する前に、すべてのデーモンタスクが停止するのを待機します。

AWS マネジメントコンソール

1. Amazon ECS コンソールを開きます。ナビゲーションで [クラスター] を選択し、クラスターを選択します。
2. [デーモン] タブを選択し、削除するデーモンを選択します。
3. [削除] を選択します。
4. 確認ダイアログで、[削除] を選択して確認します。

AWS CLI

`delete-daemon` コマンドを実行します。

```
aws ecs delete-daemon \
    --daemon-arn arn:aws:ecs:us-east-1:123456789012:daemon/my-daemon-cluster/my-monitoring-daemon
```

デーモンデプロイ

デーモンを作成または更新すると、Amazon ECS はローリングデプロイを実行して、関連付けられたキャパシティプロバイダー内のすべてのインスタンスでデーモンタスクを実行します。このセクションでは、デーモンデプロイの仕組みとデプロイ動作の設定方法について説明します。

デーモンデプロイの仕組み

デーモンデプロイは、drain-provision-replace パターンに従います。既存のインスタンスがあるキャパシティプロバイダーでデーモンを作成するか、デーモンを新しいタスク定義リビジョンに更新すると、Amazon ECS はインスタンスのバッチをドレインし、更新されたデーモンで置換インスタンスをプロビジョニングし、アプリケーションタスクを自動的に置き換えます。このプロセスは、すべてのインスタンスが新しいデーモンリビジョンを実行するまで繰り返されます。

デプロイライフサイクルは、次の状態を介して移行します:

1. PENDING - Amazon ECS はデプロイを作成し、開始する準備をしています。
2. IN_PROGRESS - Amazon ECS はインスタンスをアクティブにドレインし、置き換えをプロビジョニングしています。
3. SUCCESSFUL - すべてのインスタンスがターゲットデーモンリビジョンを実行しています。
4. STOPPED - Amazon ECS は、デプロイが失敗したか、新しいデプロイに置き換えられたため、デプロイを停止しました。デーモンは、古いリビジョンを実行するいくつかのインスタンスと新しいリビジョンを実行する他のインスタンスが混在している場合があります。

デプロイサーキットブレーカーが障害を検出した場合、またはデプロイ中に CloudWatch アラームがトリガーされた場合、Amazon ECS は自動的に以前のデーモンリビジョンにロールバックされます。

デプロイ中、非クリティカルなデーモンタスクが失敗しても、インスタンスはドレインまたは置き換えられないため、インスタンスはアクティブのままになり、アプリケーションタスクの実行を継続します。デーモンタスクの起動失敗は、引き続きデプロイサーキットブレーカーにカウントされ、不安定なターゲットリビジョンをロールバックする可能性があります。

デプロイ設定パラメータ

デーモンを作成または更新するときに、deploymentConfiguration パラメータを使用してデプロイ動作をカスタマイズできます。

- drainPercent (1.0 ~ 100.0) - デプロイ中に同時にドレインするインスタンスの割合。値を大きくするとデプロイが高速化されますが、一時的に使用可能な容量が減る可能性があります。例えば、20.0 の値は、一度に 20% のインスタンスをドレインします。指定されなかった場合、デフォルト値は 25.0 です。
- alarms (DaemonAlarmConfiguration) - デプロイプロセス中にモニタリングする CloudWatch のアラーム。Amazon ECS は、デプロイ中に指定されたアラームを評価し、アラームが ALARM 状態になると自動的にロールバックします。Amazon ECS は、デプロイの開始時に既に ALARM 状態にあるアラームを無視します。
- bakeTimeInMinutes (0 ~ 1440) - Amazon ECS がすべてのインスタンスを新しいデーモンリビジョンに更新してからデプロイを完了するまで待機する分数。この期間中、Amazon ECS は CloudWatch アラームをモニタリングし、アラームがトリガーされるとデプロイを自動的にロールバックします。指定されなかった場合、デフォルト値は 0 です。

デプロイ設定の例:

```
aws ecs create-daemon \
    --cluster-arn arn:aws:ecs:us-east-1:123456789012:cluster/my-daemon-cluster \
    --daemon-name my-monitoring-daemon \
    --daemon-task-definition-arn arn:aws:ecs:us-east-1:123456789012:daemon-task-
definition/my-daemon-task:1 \
    --capacity-provider-arns arn:aws:ecs:us-east-1:123456789012:capacity-provider/my-
daemon-capacity-provider \
    --deployment-configuration '{"drainPercent":20.0,"bakeTimeInMinutes":5}'
```

デプロイのモニタリング

次のコマンドを使用してデプロイの進捗状況を追跡します:

```
aws ecs list-daemon-deployments \
    --daemon-arn arn:aws:ecs:us-east-1:123456789012:daemon/my-daemon-cluster/my-
monitoring-daemon
```

```
aws ecs describe-daemon-deployments \
    --daemon-deployment-arns arn:aws:ecs:us-east-1:123456789012:daemon-deployment/my-
daemon-cluster/abc123
```

describe-daemon-deployments レスポンスには、デプロイステータス、ターゲットデーモンリビジョン、およびターゲットデーモンを実行するインスタンスの数を示す totalRunningInstanceCount フィールドが含まれます。

デプロイ例のシナリオ

このセクションでは、マネージドデーモンの一般的なデプロイシナリオについて説明します。

- 空のキャパシティプロバイダーでデーモンをデプロイする
- 既存のインスタンスがあるキャパシティプロバイダーにデーモンをデプロイする
- キャパシティプロバイダーを既存のデーモンに追加する
- デーモンを新しいリビジョンに更新する

空のキャパシティプロバイダーでデーモンをデプロイする

このシナリオでは、既存のインスタンスがないキャパシティプロバイダーでデーモンをデプロイします。インスタンスは、アプリケーションタスクをスケジュールするときに起動します。

前提条件: クラスターおよび実行中のインスタンスがない Amazon ECS マネージドインスタンスキャパシティプロバイダー。

1. デーモンタスク定義を登録します。
2. デーモンを作成します。Amazon ECS は、インスタンスがまだ存在しない場合でも、すぐに作成を完了します。

```
aws ecs create-daemon \
    --cluster-arn arn:aws:ecs:us-east-1:123456789012:cluster/my-daemon-cluster \
    --daemon-name my-monitoring-daemon \
    --daemon-task-definition-arn arn:aws:ecs:us-east-1:123456789012:daemon-task-definition/my-daemon-task:1 \
    --capacity-provider-arns arn:aws:ecs:us-east-1:123456789012:capacity-provider/my-daemon-capacity-provider
```

3. 同じキャパシティプロバイダーを使用するサービスを作成します。Amazon ECS がサービスのインスタンスをプロビジョニングすると、最初にデーモンタスクを開始し、アプリケーションタスクを RUNNING に移行します。

```
aws ecs create-service \
    --cluster my-daemon-cluster \
    --service-name my-app-service \
    --task-definition my-app-task \
    --desired-count 2 \
    --capacity-provider-strategy capacityProvider=my-daemon-capacity-provider,weight=1
```

4. デーモンタスクが各インスタンスで実行されていることを確認するには、describe-daemons コマンドを使用するか、コンソールの [デーモン] タブを確認します。

> **Note**
>
> デーモンデプロイは、Amazon ECS がインスタンスをプロビジョニングし、デーモンタスクを開始する間、一時的に PENDING 状態のままになることがあります。デプロイが SUCCESSFUL に到達するのを待ってから、デーモンタスクの配置を確認します。

既存のインスタンスがあるキャパシティプロバイダーにデーモンをデプロイする

このシナリオでは、実行中のインスタンスとアプリケーションタスクが既にあるキャパシティプロバイダーでデーモンをデプロイします。

前提条件: 実行中のインスタンスを持つ Amazon ECS マネージドインスタンスキャパシティプロバイダーがあるクラスター。

1. デーモンタスク定義を登録します。
2. デーモンを作成します。Amazon ECS はローリングデプロイを開始して、すべての既存のインスタンスにデーモンタスクを配置します。

```
aws ecs create-daemon \
    --cluster-arn arn:aws:ecs:us-east-1:123456789012:cluster/my-daemon-cluster \
    --daemon-name my-monitoring-daemon \
    --daemon-task-definition-arn arn:aws:ecs:us-east-1:123456789012:daemon-task-
definition/my-daemon-task:1 \
    --capacity-provider-arns arn:aws:ecs:us-east-1:123456789012:capacity-provider/my-
daemon-capacity-provider \
    --deployment-configuration '{"drainPercent":25.0,"bakeTimeInMinutes":3}'
```

   Amazon ECS は既存のインスタンスのバッチをドレインし (drainPercent に基づく)、置換インスタンスをデーモンでプロビジョニングして、アプリケーションタスクを置き換えます。このプロセスは、すべてのインスタンスがデーモンを実行するまで繰り返されます。
3. デプロイの進捗状況をモニタリングする

```
aws ecs list-daemon-deployments \
    --daemon-arn arn:aws:ecs:us-east-1:123456789012:daemon/my-daemon-cluster/my-
monitoring-daemon
```

キャパシティプロバイダーを既存のデーモンに追加する

このシナリオでは、既存のデーモンに 2 番目のキャパシティプロバイダーを追加します。デーモンは、新しいキャパシティプロバイダーのインスタンスに自動的にデプロイされます。

前提条件: 最初のキャパシティプロバイダーで実行されているデーモンと、作成され、クラスターと関連付けられた 2 番目のキャパシティプロバイダー。

1. デーモンを更新して、両方のキャパシティプロバイダーを含めます。

```
aws ecs update-daemon \
    --daemon-arn arn:aws:ecs:us-east-1:123456789012:daemon/my-daemon-cluster/my-monitoring-daemon \
    --daemon-task-definition-arn arn:aws:ecs:us-east-1:123456789012:daemon-task-definition/my-daemon-task:1 \
    --capacity-provider-arns \
        arn:aws:ecs:us-east-1:123456789012:capacity-provider/my-daemon-capacity-provider \
        arn:aws:ecs:us-east-1:123456789012:capacity-provider/my-daemon-capacity-provider-2
```

2. デプロイをモニタリングし、デーモンタスクが両方のキャパシティプロバイダーのインスタンスで実行されていることを確認します。

デーモンを新しいリビジョンに更新する

このシナリオでは、新しいタスク定義リビジョンを使用するように既存のデーモンを更新します。

1. 更新されたコンテナイメージまたは設定を使用して、デーモンタスク定義の新しいリビジョンを登録します。
2. 新しいリビジョンを使用するようにデーモンを更新します。

```
aws ecs update-daemon \
    --daemon-arn arn:aws:ecs:us-east-1:123456789012:daemon/my-daemon-cluster/my-monitoring-daemon \
    --daemon-task-definition-arn arn:aws:ecs:us-east-1:123456789012:daemon-task-definition/my-daemon-task:2
```

   Amazon ECS はローリングデプロイを実行します。古いリビジョンを実行しているインスタンスをドレインし、新しいリビジョンで置換インスタンスをプロビジョニングし、アプリケーションタスクを自動的に置き換えます。サーキットブレーカーが障害を検出した場合、Amazon ECS は以前のリビジョンにロールバックします。

3. デプロイをモニタリングし、すべてのインスタンスで新しいリビジョンが実行されていることを確認します。

```
aws ecs describe-daemon-deployments \
    --daemon-deployment-arns arn:aws:ecs:us-east-1:123456789012:daemon-deployment/my-daemon-cluster/deployment-id
```

デーモンイベント

Amazon ECS はデーモンイベントを送信して、デーモンタスク管理とデプロイライフサイクルの変更を可視化します。これらのイベントを使用して、デーモンヘルスのモニタリング、デプロイの進捗状況の追跡、および自動ワークフローのトリガーを行うことができます。

Amazon ECS は、次のデーモンイベントタイプをサポートします。

- デーモンサービスアクションイベント - デーモンタスクの配置とスタートアップの問題に関連するイベント。
- デーモンデプロイ状態変更イベント - デーモンデプロイライフサイクルの移行に関連するイベント。

これらのイベントのルールを作成する方法については、「デーモンイベントのルールの作成」を参照してください。

デーモンサービスアクションイベント

Amazon ECS は、詳細タイプ `ECS Daemon Service Action` のデーモンサービスアクションイベントを送信します。これらのイベントは、Amazon ECS がコンテナインスタンスでデーモンタスクを開始するときに問題が発生した場合に通知します。Amazon ECS は、クリティカルなデーモンと非クリティカルなデーモンの両方について、デーモンタスクの起動に失敗したときに EventBridge イベントを出力します。

DAEMON_TASK_START_IMPAIRED

Amazon ECS は、コンテナインスタンスでデーモンタスクを正常に開始できない場合に `DAEMON_TASK_START_IMPAIRED` イベントを送信します。イベントには、障害の原因を示す `failureType`フィールドが含まれます。

- `TASK_FAILED_TO_RUN` - デーモンタスクが作成されましたが、`RUNNING` ステータスに到達できませんでした。一般的な原因には、コンテナイメージのプルの失敗、コンテナヘルスチェックの失敗、必須コンテナの終了などがあります。`taskArn` フィールドはイベントに存在します。
- `TASK_CREATION_FAILED` - デーモンタスクをコンテナインスタンスに作成できませんでした。一般的な原因には、インスタンスの CPU、メモリ、またはその他のリソースの不足などがあります。タスクが作成されていないため、`taskArn` フィールドはイベントに存在しません。

例: TASK_FAILED_TO_RUN イベント

次のイベントは、作成されたけれども、コンテナイメージのプルに失敗したために RUNNING ステータスに到達できなかったデーモンタスクを示しています。

```
{
    "version": "0",
    "id": "12345678-1234-1234-1234-123456789012",
    "detail-type": "ECS Daemon Service Action",
    "source": "aws.ecs",
    "account": "123456789012",
    "time": "2026-03-24T12:00:00Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:123456789012:task/my-cluster/a1b2c3d4e5f6"
    ],
    "detail": {
        "eventType": "WARNING",
        "eventName": "DAEMON_TASK_START_IMPAIRED",
        "clusterArn": "arn:aws:ecs:us-west-2:123456789012:cluster/my-cluster",
        "containerInstanceArn": "arn:aws:ecs:us-west-2:123456789012:container-
instance/my-cluster/a1b2c3d4e5f6",
        "taskArn": "arn:aws:ecs:us-west-2:123456789012:task/my-cluster/a1b2c3d4e5f6",
        "daemonRevisionArn": "arn:aws:ecs:us-west-2:123456789012:daemon-revision/my-
cluster/my-daemon/a1b2c3d4-e5f6-7890-abcd-ef1234567890",
        "capacityProviderName": "my-capacity-provider",
        "daemonArn": "arn:aws:ecs:us-west-2:123456789012:daemon/my-cluster/my-daemon",
        "daemonTaskDefinitionArn": "arn:aws:ecs:us-west-2:123456789012:daemon-task-
definition/my-daemon-td:1",
        "failureType": "TASK_FAILED_TO_RUN",
        "createdAt": "2026-03-24T12:00:00.000Z",
        "reason": "Task failed to reach RUNNING status: CannotPullContainerError: pull
 image manifest has been retried 5 time(s)"
    }
}
```

例: TASK_CREATION_FAILED イベント

次のイベントは、CPU リソースが不十分なためにコンテナインスタンスで作成できなかったデーモンタスクを示しています。タスクが作成されていないため、taskArn フィールドは存在しません。

```
{
```

```
    "version": "0",
    "id": "87654321-4321-4321-4321-210987654321",
    "detail-type": "ECS Daemon Service Action",
    "source": "aws.ecs",
    "account": "123456789012",
    "time": "2026-03-24T12:01:00Z",
    "region": "us-west-2",
    "resources": [],
    "detail": {
        "eventType": "WARNING",
        "eventName": "DAEMON_TASK_START_IMPAIRED",
        "clusterArn": "arn:aws:ecs:us-west-2:123456789012:cluster/my-cluster",
        "containerInstanceArn": "arn:aws:ecs:us-west-2:123456789012:container-
instance/my-cluster/b2c3d4e5f6a7",
        "daemonRevisionArn": "arn:aws:ecs:us-west-2:123456789012:daemon-revision/my-
cluster/my-daemon/a1b2c3d4-e5f6-7890-abcd-ef1234567890",
        "capacityProviderName": "my-capacity-provider",
        "daemonArn": "arn:aws:ecs:us-west-2:123456789012:daemon/my-cluster/my-daemon",
        "daemonTaskDefinitionArn": "arn:aws:ecs:us-west-2:123456789012:daemon-task-
definition/my-daemon-td:1",
        "failureType": "TASK_CREATION_FAILED",
        "createdAt": "2026-03-24T12:01:00.000Z",
        "reason": "RESOURCE:CPU - Unable to place daemon task on container instance:
 insufficient CPU"
    }
}
```

デーモンデプロイ状態変更イベント

Amazon ECS は、詳細タイプの ECS Daemon Deployment State Change を使用してデーモンデプロイ状態変更イベントを送信します。Amazon ECS は、デーモンデプロイの状態が変化するたびにこれらのイベントを発行します。

デプロイ状態変更イベントタイプ

Amazon ECS は、デプロイ状態変更イベントを次のイベントタイプで分類します:

INFO イベント

- DAEMON_DEPLOYMENT_PENDING - Amazon ECS はデーモンのデプロイを開始しました。
- DAEMON_DEPLOYMENT_IN_PROGRESS - Amazon ECS はデプロイを開始し、デーモンタスクを積極的に置き換えています。

- DAEMON_DEPLOYMENT_SUCCESSFUL - Amazon ECS は、すべてのデーモンタスクが実行され、正常にデプロイを完了しました。
- DAEMON_DEPLOYMENT_STOPPED - Amazon ECS はデプロイを停止しました。これは、デプロイが失敗した場合、または新しいデプロイに置き換えられた場合に発生します。
- DAEMON_DEPLOYMENT_STOP_REQUESTED - Amazon ECS はデプロイの進行を停止したので、ロールバックを開始します。
- DAEMON_DEPLOYMENT_ROLLBACK_IN_PROGRESS - Amazon ECS は、デプロイの失敗またはサーキットブレーカートリガーによりロールバックを開始しました。
- DAEMON_DEPLOYMENT_ROLLBACK_SUCCESSFUL - Amazon ECS は、デプロイのロールバックを正常に完了しました。

ERROR イベント

- DAEMON_DEPLOYMENT_ROLLBACK_FAILED - Amazon ECS はデプロイのロールバックを完了できませんでした。

例: デプロイ保留中のイベント

次のイベントは、Amazon ECS が開始したデーモンデプロイを示しています。

```
{
    "version": "0",
    "id": "3329f79b-3dca-07f8-b1c2-5fe99f0b5e87",
    "detail-type": "ECS Daemon Deployment State Change",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2026-03-05T15:54:41Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:111122223333:daemon/my-cluster/my-daemon"
    ],
    "detail": {
        "eventType": "INFO",
        "eventName": "DAEMON_DEPLOYMENT_PENDING",
        "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/my-cluster",
        "daemonArn": "arn:aws:ecs:us-west-2:111122223333:daemon/my-cluster/my-daemon",
        "daemonDeploymentArn": "arn:aws:ecs:us-west-2:111122223333:daemon-
deployment/my-cluster/my-daemon/0EYSiB0qap8xf0N76FsbE",
```

```
        "targetDaemonRevisionArn": "arn:aws:ecs:us-west-2:111122223333:daemon-
revision/my-cluster/my-daemon/85707969-3732-4b6a-a37d-5cefddd7d7dd",
        "updatedAt": "2026-03-05T15:54:41.618059641Z"
    }
}
```

デーモンイベントのルールの作成

デーモンイベントが発生したときに通知を受け取るルールを作成できます。ルールの作成の詳細については、「Amazon EventBridge ユーザーガイド」の「ルールの作成」を参照してください。

次のイベントパターンの例は、デーモンイベントをフィルタリングする方法を示しています。

例: すべてのデーモンサービスアクションイベントと一致する

```
{
    "source": ["aws.ecs"],
    "detail-type": ["ECS Daemon Service Action"]
}
```

例: 特定の障害タイプと一致する

```
{
    "source": ["aws.ecs"],
    "detail-type": ["ECS Daemon Service Action"],
    "detail": {
        "eventName": ["DAEMON_TASK_START_IMPAIRED"],
        "failureType": ["TASK_CREATION_FAILED"]
    }
}
```

例: 特定のクラスターのイベントと一致する

```
{
    "source": ["aws.ecs"],
    "detail-type": ["ECS Daemon Service Action"],
    "detail": {
        "eventName": ["DAEMON_TASK_START_IMPAIRED"],
        "clusterArn": ["arn:aws:ecs:us-west-2:123456789012:cluster/my-cluster"]
    }
```

```
}
```

例: すべてのデプロイ状態変更イベントと一致する

```
{
    "source": ["aws.ecs"],
    "detail-type": ["ECS Daemon Deployment State Change"]
}
```

デーモンメトリクス

Amazon ECS はマネージドデーモンの CloudWatch メトリクスを発行します。これらのメトリクスを使用して、クラスター全体でデーモンタスクの CPU とメモリの使用率を追跡できます。

デーモンメトリクスは、サービスメトリクスと同じ `AWS/ECS` 名前空間を使用します。Amazon ECS は、プレフィックス `daemon:` の後にデーモン名 (`daemon:`*`my-daemon`* など) が続く `ServiceName` ディメンション値によって、デーモンメトリクスとサービスメトリクスを区別します。

デーモン使用率メトリクス

Amazon ECS は、デーモンタスク定義で指定されたリソースと比較して、デーモンタスクで使用されるリソースの割合としてデーモンの CPU とメモリ使用率を測定します。

CloudWatch は、デーモン使用率メトリクスの次の統計を提供します。

- 平均 - クラスター内のすべてのデーモンタスクの平均使用率。
- 最小 - リソース使用量が最も少ないデーモンタスクの使用率。
- 最大 - リソース使用量が最も多いデーモンタスクの使用率。

使用率の計算式

Amazon ECS は、次の式を使用して平均統計を計算します:

```
                                    (Total CPU units used by daemon tasks) x 100
Daemon CPU utilization =
 ---------------------------------------------------------------------------
                          (Total CPU units specified in daemon task definition) x
 (Number of daemon tasks)
```

```
                                        (Total MiB of memory used by daemon tasks) x
 100
Daemon memory utilization =
 ------------------------------------------------------------------------------------
                             (Total MiB of memory specified in daemon task definition)
 x (Number of daemon tasks)
```

これらの式は、平均統計にのみ適用されます。最小と最大統計は、それぞれ、リソース使用率が最も低い個々のデーモンタスクと最も高い個々のデーモンタスクを表します。

Amazon ECS コンテナエージェントは、実行中の各デーモンタスクの CPU とメモリの使用率を 20 秒ごとにサンプリングします。1 分ごとに、エージェントは 3 つのサンプルを Amazon ECS に報告します。Amazon ECS は、クラスター内のすべてのデーモンタスク全体のサンプルを集約し、デーモンタスク定義で指定された合計リソースに対する割合として、平均値、最小値、最大値を CloudWatch に発行します。

例: 使用率の計算

デーモンタスク定義は、256 CPU ユニットと 512 MiB のメモリを指定します。デーモンには、2 つのコンテナインスタンス全体で 2 つの実行中のタスクがあります。1 つのデーモンタスクは、200 CPU ユニットと 400 MiB のメモリを使用します。もう 1 つは 100 CPU ユニットと 256 MiB のメモリを使用します。Amazon ECS は、次のメトリクスを報告します:

- 平均 CPU 使用率: ((200 + 100) x 100)/(256 x 2) = 58.6%
- 平均メモリ使用率: ((400 + 256) x 100)/(512 x 2) = 64.1%

デーモンメトリクスの表示

デーモンメトリクスは、AWS マネジメントコンソール または AWS CLI を使用して表示できます。

AWS マネジメントコンソール

1. CloudWatch コンソールを開きます。
2. ナビゲーションペインで、[メトリクス] 、[すべてのメトリクス] の順に選択します。
3. AWS/ECS 名前空間を選択します。
4. ClusterName, ServiceName ディメンションを選択します。
5. `daemon:` で始まる `ServiceName` 値をフィルタリングして、デーモンメトリクスを見つけます。

AWS CLI

get-metric-statistics コマンドを実行して、CloudWatch からデーモンメトリクスを取得します。daemon: プレフィックスで ServiceName ディメンションを指定します。

例: デーモン CPU 使用率の表示

次のコマンドは、5 分間隔で 12 時間にわたってデーモンの平均 CPU 使用率を取得します。

```
aws cloudwatch get-metric-statistics \
    --namespace AWS/ECS \
    --metric-name CPUUtilization \
    --dimensions Name=ClusterName,Value=my-cluster Name=ServiceName,Value=daemon:my-
daemon \
    --start-time 2026-03-24T00:00:00Z \
    --end-time 2026-03-24T12:00:00Z \
    --period 300 \
    --statistics Average
```

例: デーモンメモリ使用率の表示

次のコマンドは、5 分間隔で 12 時間にわたってデーモンの平均メモリ使用率を取得します。

```
aws cloudwatch get-metric-statistics \
    --namespace AWS/ECS \
    --metric-name MemoryUtilization \
    --dimensions Name=ClusterName,Value=my-cluster Name=ServiceName,Value=daemon:my-
daemon \
    --start-time 2026-03-24T00:00:00Z \
    --end-time 2026-03-24T12:00:00Z \
    --period 300 \
    --statistics Average
```

デーモン自動修復

デーモンは、インスタンスのヘルスにとってクリティカルでも非クリティカルであってもよく、デーモンを作成または更新するときに critical パラメータで制御します。critical パラメータのデフォルトは true であるため、critical を false に設定しない限り、デーモンはクリティカルになります。

クリティカルなデーモンのデーモンタスクが停止または異常になった場合、Amazon ECS はインスタンスに障害があると見なし、自動的にドレインして置き換えます。クリティカルなデーモンのデーモン自動修復アクションは次のとおりです:

1. Amazon ECS は、デーモンタスクが停止または異常になったことを検出します。
2. Amazon ECS はインスタンスをドレインとしてマークし、これにより新しいアプリケーションタスクは受け入れられません。
3. Amazon ECS は置換インスタンスをプロビジョニングし、そのインスタンスでデーモンタスクを開始します。
4. デーモンタスクが正常な状態になると、Amazon ECS はドレインインスタンスから置換インスタンスにアプリケーションタスクをスケジュールします。
5. Amazon ECS は、元のインスタンスを終了します。

非クリティカルなデーモンの場合、デーモンタスクはコンテナインスタンスのヘルスとは無関係に動作します。デーモンタスクが失敗、停止、または異常になった場合、Amazon ECS はコンテナインスタンスをアクティブに保ちます。非クリティカルなデーモンはインスタンス登録をブロックしないため、デーモンの起動に失敗してもアプリケーションタスクをすぐに配置できます。

> **Important**
>
> デーモンヘルスチェックはオプションですが、強くお勧めします。ヘルスチェックがないと、Amazon ECS はデーモンタスクが停止したときにのみ障害を検出できます。

`DescribeContainerInstances` API または `DescribeTasks` API を使用してデーモンヘルスをモニタリングできます。

非クリティカルなデーモンの場合、Amazon ECS はデーモンタスクが実行されていないときにインスタンスをアクティブに保ちます。次のフィールドは、その状態にあるインスタンスの数を報告します。

- `DescribeDaemon` は、各キャパシティープロバイダーについて `withoutDaemonCount` を返し、すべてのキャパシティープロバイダーにわたるリビジョンについて `totalWithoutDaemonCount` を返します。
- `DescribeDaemonDeployments` は、各キャパシティープロバイダーについて `withoutDaemonInstanceCount` を返し、すべてのキャパシティープロバイダーにわたるリビジョンについて `totalWithoutDaemonInstanceCount` を返します。

> **Note**
>
> デーモンなしの数は、実行中の数とは別です。DescribeDaemon は runningCount と totalRunningCount を報告します。DescribeDaemonDeployments は runningInstanceCount と totalRunningInstanceCount を報告します。インスタンスはどちらか一方のセットにのみ表示され、両方に表示されることはないため、2 つのセットを合算しないでください。

Amazon ECS の AWS Fargate 用のアーキテクト

AWS Fargate Fargate は Amazon ECS で使用できるテクノロジーであり、サーバーやAmazon EC2 インスタンスの クラスターを管理することなくコンテナを実行できます。AWS Fargate を使用すると、コンテナを実行するために仮想マシンのクラスターをプロビジョニング、設定、スケールする必要がありません。これにより、サーバータイプの選択、クラスターをスケールするタイミングの決定、クラスターのパッキングの最適化を行う必要がなくなります。

Fargate を使用してタスクやサービスを実行する場合、アプリケーションをコンテナにパッケージ化し、CPU とメモリ要件を指定して、ネットワークと IAM ポリシーを定義して、アプリケーションを起動します。各Fargate タスクは、独自の分離境界を持ち、基盤となるカーネル、CPU リソース、メモリリソース、Elastic Network Interface を別のタスクと共有しません。Fargate のタスク定義を構成するには、`requiresCompatibilities` タスク定義パラメータを `FARGATE` に設定します。詳細については、「Capacity」を参照してください。

Fargate は、Amazon Linux 2 (プラットフォームバージョン 1.3.0)、Bottlerocket オペレーティングシステム (プラットフォームバージョン 1.4.0)、および Microsoft Windows 2019 Server Full Edition および Core Edition 用のプラットフォームバージョンを提供しています。特に指定がない限り、この情報はすべての Fargate プラットフォームに適用されます。

Fargate で Linux コンテナをサポートするリージョンの情報については、「the section called “AWS Fargate 上の Linux コンテナ”」を参照してください。

Fargate で Windows コンテナをサポートするリージョンの情報については、「the section called “AWS Fargate 上の Windows コンテナ”」を参照してください。

チュートリアル

コンソールの使用開始方法については、以下を参照してください。

- Fargate 用の Amazon ECS Linux タスクを作成する方法について説明します。
- Fargate 用の Amazon ECS Windows タスクを作成する方法について説明します。

AWS CLI の使用開始方法については、以下を参照してください。

- AWS CLI を使用して、Fargate 用の Amazon ECS Linux タスクを作成する
- AWS CLI を使用して、Fargate 用の Amazon ECS Windows タスクを作成する

キャパシティープロバイダー

以下のキャパシティプロバイダーが利用可能です。

- Fargate
- Fargate Spot - 割り込み許容のある Amazon ECS タスクを、AWS Fargate 料金と比較して割引料金で実行します。Fargate Spot は、予備のコンピュートキャパシティーでタスクを実行します。AWS がキャパシティーを戻す必要がある場合、タスクは中断され、2 分間の警告が表示されます。詳細については、「Fargate 用の Amazon ECS クラスター」を参照してください。

タスク定義

Fargate タスクは、利用可能な Amazon ECS タスク定義パラメータのすべてをサポートするわけではありません。一部のパラメータはサポートされていません。また、Fargate タスクでは動作が異なるパラメータがあります。詳細については、「タスク CPU とメモリ」を参照してください。

プラットフォームのバージョン

AWS Fargate プラットフォームのバージョンを使って、Fargate タスクインフラストラクチャの特定のランタイム環境を参照できます。これは、カーネルとコンテナのランタイムバージョンの組み合わせです。同一のタスクを多数管理するためのタスク実行時、またはそのためのサービス作成時には、プラットフォームバージョンを選択します。

ランタイム環境の進化に合わせて、新しいプラットフォームバージョンがリリースされます。例えば、カーネルやオペレーティングシステムの更新、新機能の追加、バグ修正、セキュリティの更新があった場合が当てはまります。Fargate プラットフォームのバージョンは、新しいプラットフォームバージョンのリビジョンを行うことで更新されます。各タスクは、そのライフサイクルを通じて、単一のプラットフォームバージョンのリビジョンで実行されます。最新のプラットフォームバージョンのリビジョンを使用する場合は、新たにタスクを開始する必要があります。Fargate で実行される新しいタスクは、常にプラットフォームバージョンの最新リビジョンで実行されます。これにより、タスクは必ず、安全でパッチ適用済みのインフラストラクチャで開始されることが保証されます。

既存のプラットフォームのバージョンに影響を与えるセキュリティ上の問題が見つかった場合、AWS は、そのプラットフォームバージョンのパッチ済みリビジョンを新たに作成します。また、脆弱性のあるリビジョンで実行しているタスクは廃止されます。場合によっては、Fargate で使用しているタスクについて、廃止の予定が通知されることがあります。詳細については、「AWS Fargate on Amazon ECS のタスクの廃止とメンテナンス」を参照してください。

詳細については、「Amazon ECS 向け Fargate プラットフォームバージョン」を参照してください。

サービスの負荷分散

AWS Fargate の Amazon ECS サービスは、オプションで Elastic Load Balancing を使用して、サービスのタスク間でトラフィックを均等に分散するように設定できます。

AWS Fargate の Amazon ECS サービスでは、Application Load Balancer、Network Load Balancer、および Gateway Load Balancer のロードバランサータイプがサポートされています。アプリケーションロードバランサーは、HTTP/HTTPS (またはレイヤー 7) トラフィックをルーティングするために使用されます。ネットワークロードバランサーは、TCP または UDP (またはレイヤー 4)トラフィックをルーティングするために使用されます。詳細については、「ロードバランサーを使用して Amazon ECS サービストラフィックを分散する」を参照してください。

また、このようなサービスのターゲットグループを作成する場合は、ターゲットタイプとして `instance` ではなく、`ip` を選択する必要があります。これは、`awsvpc` ネットワークモードを使用するタスクは、Amazon EC2インスタンスではなく、Elastic Network Interface に関連付けられているためです。詳細については、「ロードバランサーを使用して Amazon ECS サービストラフィックを分散する」を参照してください。

Network Load Balancer を使用して AWS Fargate タスクの Amazon ECS に UDP トラフィックをルーティングするには、プラットフォームバージョン 1.4 移行を使用する場合にのみサポートされます。

使用状況メトリクス

CloudWatch 使用状況メトリクスを使用して、アカウントのリソースの使用状況を把握できます。これらのメトリクスを使用して、CloudWatch グラフやダッシュボードで現在のサービスの使用状況を可視化できます。

AWS Fargate 使用状況メトリクスは、AWS のサービスクォータに対応しています。使用量がサービスクォータに近づいたときに警告するアラームを設定することもできます。AWS Fargate のサービスクォータに関する詳細については、「Amazon Web Services 全般のリファレンス」の「Amazon ECS のエンドポイントとクォータ」を参照してください。

AWS Fargate 使用状況メトリクスの詳細については、「AWS Fargate 使用状況メトリクス」を参照してください。

Fargate をどのような場合に使用するかについての Amazon ECS セキュリティ上の考慮事項

タスクの強力な分離を求めているお客様には、Fargate を使用することをお勧めします。Fargate はハードウェア仮想化環境で各タスクを実行します。これにより、これらのコンテナ化されたワークロードがネットワークインターフェイス、Fargate の一時ストレージ、CPU、またはメモリを他のタスクと共有しないことが保証されます。

Amazon ECS の Fargate セキュリティのベストプラクティス

AWS Fargate を使用する際には、次のベストプラクティスを考慮することをお勧めします。

AWS KMS を使用して Fargate のエフェメラルストレージを暗号化する

エフェメラルストレージは、 AWS KMS または独自のカスタマーマネージドキーで暗号化する必要があります。バージョン `1.4.0` 以降のプラットフォームを使用していて、Fargate でホストされているタスクの場合は、最低 20 GiB のエフェメラルストレージを受け取ります。詳細については、「カスタマーマネージドキー (CMK)」を参照してください。エフェメラルストレージの総量は、タスク定義で `ephemeralStorage` パラメータを指定することによって、最大 200 GiB まで増やすことができます。2020 年 5 月 28 日以降に起動されたこのようなタスクでは、エフェメラルストレージは、Fargate によって管理される暗号化キーを使用して、AES-256 暗号化アルゴリズムによって暗号化されます。

詳細については、「Amazon ECS タスクのストレージオプション」を参照してください。

例: Fargate プラットフォーム バージョン 1.4.0 で、エフェメラルストレージの暗号化を使用してタスクを起動する

次のコマンドは、Fargate プラットフォームバージョン 1.4 でタスクを起動します。このタスクは Amazon ECS クラスターの一部として起動されるため、自動的に暗号化された 20 GiB のエフェメラルストレージが使用されます。

```
aws ecs run-task --cluster clustername \
  --task-definition taskdefinition:version \
  --count 1
  --launch-type "FARGATE" \
  --platform-version 1.4.0 \
  --network-configuration
 "awsvpcConfiguration={subnets=[subnetid],securityGroups=[securitygroupid]}" \
```

```
  --region region
```

Fargate を使用したカーネルシステムコールトレーシングの SYS_PTRACE 機能

コンテナに追加または削除される Linux 機能のデフォルト設定は、Docker が行います。

Fargate で起動したタスクでは、SYS_PTRACE カーネル機能の追加のみがサポートされます。

Sysdig Falco プロジェクトでこの機能を使用する方法を示す以下の動画。

#ContainersFromTheCouch - SYS_PTRACE 機能を使用した Fargate タスクのトラブルシューティング

前のビデオで説明したコードは、こちらの GitHub にあります。

Fargate Runtime Monitoring で Amazon GuardDuty を使用する

Amazon GuardDuty は、AWS 環境内のアカウント、コンテナ、ワークロード、データを保護する脅威検知サービスです。GuardDuty は、機械学習（ML）モデル、異常および脅威検出機能を使用して、さまざまなログソースとランタイムアクティビティを継続的に監視し、環境内の潜在的なセキュリティリスクと悪意のあるアクティビティを特定して優先順位を付けます。

GuardDuty のランタイムモニタリングは、AWS ログとネットワークアクティビティを継続的に監視して悪意のある動作や不正な動作を特定することで、Fargate で実行されているワークロードを保護します。ランタイムモニタリングは、軽量でフルマネージド型の GuardDuty セキュリティエージェントを使用して、ファイルアクセス、プロセス実行、ネットワーク接続などのホスト上動作を分析します。これは、Amazon EC2 インスタンスおよびコンテナワークロードでの権限の昇格、流出した認証情報の使用、悪意のある IP アドレスやドメインとの通信、マルウェアの存在などの問題に対応しています。詳細については、「GuardDuty ユーザーガイド」の「GuardDuty ランタイムモニタリング」を参照してください。

Amazon ECS の Fargate のセキュリティに関する考慮事項

Fargate は各ワークロードを独立した仮想環境で実行するため、各タスクには専用のインフラストラクチャ容量があります。Fargate で実行されるワークロードは、ネットワークインターフェイス、エフェメラルストレージ、CPU、またはメモリを他のタスクと共有しません。アプリケーションコンテナやサイドカーコンテナを含む複数のコンテナをタスク内で実行することも、単にサイドカーを

実行することもできます。サイドカーは Amazon ECS タスク内のアプリケーションコンテナと一緒に実行されるコンテナです。アプリケーションコンテナはコアアプリケーションコードを実行しますが、サイドカーで実行されるプロセスはアプリケーションを拡張できます。サイドカーはアプリケーションの機能を専用のコンテナに分離するのに役立ち、アプリケーションの一部を簡単に更新することができます。

同じタスクに属するコンテナは、常に同じホスト上で実行され、コンピュートリソースを共有するため、Fargate 起動タイプのリソースを共有します。これらのコンテナは、Fargate が提供するエフェメラルストレージも共有しています。タスク内の Linux コンテナは、IP アドレスやネットワークポートを含むネットワーク名前空間を共有します。タスク内では、タスクに属するコンテナが localhost を介して相互通信できます。

Fargate のランタイム環境では、EC2 インスタンスでサポートされている特定のコントローラー機能を使用できません。Fargate で実行されるワークロードを設計するときは、次の点を考慮してください。

- 特権コンテナまたはアクセスなし - 特権コンテナやアクセスなどの機能は現在、Fargate では利用できません。これは Docker 内で Docker を実行するなどのユースケースに影響します。
- Linux 機能への制限付きアクセス - Fargate 上でコンテナが動作する環境はロックダウンされています。CAP_SYS_ADMIN や CAP_NET_ADMIN などのそのほかの Linux 機能は、権限昇格を防ぐために制限されています。Fargateは、タスクに CAP_SYS_PTRACE Linux機能を追加することをサポートしています。これにより、タスク内にデプロイされたオブザーバビリティツールとセキュリティツールがコンテナ化されたアプリケーションを監視できるようになります。
- 基盤となるホストにはアクセスできない - お客様も AWS オペレーターも、お客様のワークロードを実行しているホストには接続できません。ECS Exec を使用して、Fargate で実行されているコンテナでコマンドを実行したり、シェルを取得したりできます。ECS Exec を使用すると、デバッグ用の診断情報を収集できます。また、Fargate は、コンテナがファイルシステム、デバイス、ネットワーク、コンテナランタイムなどの基盤となるホストのリソースにアクセスするのを防ぎます。
- ネットワーク - セキュリティグループとネットワーク ACL を使用して、インバウンドトラフィックとアウトバウンドトラフィックをコントロールできます。Fargate のタスクは VPC 内の設定済みサブネットから IP アドレスを受け取ります。

Amazon ECS 向け Fargate プラットフォームバージョン

AWS Fargate プラットフォームのバージョンを使って、Fargate タスクインフラストラクチャの特定のランタイム環境を参照できます。これは、カーネルとコンテナのランタイムバージョンの組み合わ

せです。同一のタスクを多数管理するためのタスク実行時、またはそのためのサービス作成時には、プラットフォームバージョンを選択します。

ランタイム環境の進化に合わせて、新しいプラットフォームバージョンがリリースされます。例えば、カーネルやオペレーティングシステムの更新、新機能の追加、バグ修正、セキュリティの更新があった場合が当てはまります。Fargate プラットフォームのバージョンは、新しいプラットフォームバージョンのリビジョンを行うことで更新されます。各タスクは、そのライフサイクルを通じて、単一のプラットフォームバージョンのリビジョンで実行されます。最新のプラットフォームバージョンのリビジョンを使用する場合は、新たにタスクを開始する必要があります。Fargate で実行される新しいタスクは、常にプラットフォームバージョンの最新リビジョンで実行されます。これにより、タスクは必ず、安全でパッチ適用済みのインフラストラクチャで開始されることが保証されます。

既存のプラットフォームのバージョンに影響を与えるセキュリティ上の問題が見つかった場合、AWS は、そのプラットフォームバージョンのパッチ済みリビジョンを新たに作成します。また、脆弱性のあるリビジョンで実行しているタスクは廃止されます。場合によっては、Fargate で使用しているタスクについて、廃止の予定が通知されることがあります。詳細については、「AWS Fargate on Amazon ECS のタスクの廃止とメンテナンス」を参照してください。

プラットフォームバージョンは、タスクを実行するとき、またはサービスをデプロイするときに指定します。

プラットフォームのバージョンを指定する際には、次の点を検討してください。

- `1.4.0`、LATEST など、特定のバージョン番号を指定できます。

  [LATEST] の Linux プラットフォームバージョンは `1.4.0` です。

  [LATEST] の Windows プラットフォームバージョンは `1.0.0` です。
- サービスのプラットフォームバージョンを更新する場合は、デプロイを作成します。例えば、Linux のプラットフォームバージョン `1.3.0` で、タスクを実行しているサービスがあるとします。Linux プラットフォームバージョン `1.4.0` でタスクを実行するようにサービスを変更するには、サービスを更新し、新しいプラットフォームバージョンを指定します。タスクは、最新のプラットフォームバージョンと最新のプラットフォームバージョンのリビジョンで再デプロイされます。デプロイの詳細については、「Amazon ECS サービス」を参照してください。
- サービスがプラットフォームバージョンを更新しないでスケールアップされた場合、これらのタスクには、サービスの最新のデプロイで指定されたプラットフォームのバージョンが提供されます。例えば、Linux のプラットフォームバージョン `1.3.0` で、タスクを実行しているサービスがあるとします。このサービスについて必要とされる数を増加した場合、サービススケジューラーは、

プラットフォームバージョン 1.3.0 の最新のプラットフォームバージョンのリビジョンを使用して、新しいタスクを開始します。

- 新しいタスクは、常にプラットフォームバージョンの最新リビジョンで実行されます。この結果、タスクは常にセキュリティで保護されパッチが適用されたインフラストラクチャ上になります。
- Fargate の Linux コンテナと Windows コンテナのプラットフォームバージョンの番号は相互に独立しています。例えば、Fargate の Windows コンテナのプラットフォームバージョン 1.0.0 で使用される動作、機能、およびソフトウェアは、Fargate の Linux コンテナのプラットフォームバージョン 1.0.0 の動作、機能、およびソフトウェアとは比較できません。
- Fargate Windows プラットフォームバージョンには、以下が適用されます。

  Microsoft Windows Server のコンテナイメージは、特定のバージョンの Windows Server から作成する必要があります。Windows Server コンテナイメージに適合したタスクの実行やサービスの作成を行う場合は、platformFamily で同じバージョンの Windows Server を選択する必要があります。さらに、タスク定義で一致させる operatingSystemFamily を指定して、間違った Windows バージョンでタスクが実行されないようにできます。詳細については、Microsoft Learn のウェブサイトで「コンテナホストのバージョンとコンテナイメージのバージョンを一致させる」を参照してください。

Amazon ECS 上の Linux プラットフォームバージョン 1.4.0 への移行

FargateタスクのAmazon ECSをプラットフォームバージョン 1.0.0、1.1.0、1.2.0、または 1.3.0 からプラットフォームバージョン 1.4.0 に移行する場合は、次のことを考慮してください。タスクを移行する前に、そのタスクがプラットフォームバージョン 1.4.0 で正しく動作するのを確認することがベストプラクティスです。

- タスクとの間のネットワークトラフィック動作を更新しました。プラットフォームバージョン 1.4.0 以降では、Fargate タスクでのすべての Amazon ECS は単一の Elastic Network Interface (以降タスク ENI)を受け取り、すべてのネットワークトラフィックは VPC 内でこの ENI を通過します。このトラフィックは VPC フローログを通じて表示されます。詳細については、「Fargate における Amazon ECS タスクのネットワークオプション」を参照してください。
- インターフェイス VPC エンドポイントを使用する場合は、次の点を考慮してください。
  - Amazon ECR でホストされているコンテナイメージでは、次のエンドポイントが必要です。詳細については、Amazon Elastic コンテナレジストリ ユーザーガイド の Amazon ECR Interface VPC エンドポイント(AWS PrivateLink) を参照してください。
    - [com.amazonaws.***region***.ecr.dkr] Amazon ECR VPC エンドポイント

  - [com.amazonaws.*region*.ecr.api] Amazon ECR VPC エンドポイント
  - Amazon S3 ゲートウェイエンドポイント
- コンテナの機密データを取得する際にタスク定義が Secrets Manager シークレットを参照する場合は、Secrets Manager のインターフェイス VPC エンドポイントを作成する必要があります。詳細については、AWS Secrets Manager ユーザーガイドの VPC EndpointでSecrets Managerを使用する を参照してください。
- コンテナの機密データを取得する際にタスク定義が Systems Manager Parameter Store パラメータを参照する場合は、SystemsManager のインターフェイス VPC エンドポイントを作成する必要があります。詳細については、「AWS Systems Manager ユーザーガイド」の「Systems Manager のために VPC エンドポイントを使用して EC2 インスタンスのセキュリティを強化する」を参照してください。
- タスクに関連付けられた Elastic Network Interface (ENI) のセキュリティグループには、タスクと VPC エンドポイントとの間のトラフィックを許可するセキュリティグループルールが必要です。

Fargate Linux プラットフォームバージョンの変更ログ

使用できる Linux プラットフォームのバージョンは以下のとおりです。プラットフォームバージョンの廃止については、「AWS Fargate Linux プラットフォームバージョンの廃止」を参照してください。

1.4.0

以下は、プラットフォームバージョン `1.4.0` の変更履歴です。

- 2020 年 11 月 5 日以降、プラットフォームバージョン`1.4.0`を使用して Fargate で起動されたすべての新しい Amazon ECS タスクで次の機能を使用できるようになります:
  - Secrets Managerを使用して機密データを保存する場合、特定の JSON キーまたは特定のシークレットのバージョンを環境変数またはログ設定に挿入できます。詳細については、「Amazon ECS コンテナに機密データを渡す」を参照してください。
  - `environmentFiles` コンテナ定義パラメータを使用して、環境変数を一括で指定します。詳細については、「個々の環境変数を Amazon ECS コンテナに渡す」を参照してください。
  - VPC で実行されるタスクと IPv6 が有効になっているサブネットには、プライベート IPv4 アドレスと IPv6 アドレスの両方が割り当てられます。詳細については、「Fargate 起動タイプの Amazon ECS タスクのネットワークオプション」を参照してください。

  - タスクメタデータエンドポイントバージョン 4 には、タスク起動タイプ、コンテナの Amazon リソースネーム (ARN)、使用されるログドライバーとログドライバーオプションなど、タスクとコンテナに関する追加のメタデータが提供されます。`/stats` エンドポイントに対してクエリを実行すると、コンテナのネットワークレート統計も受け取ります。詳細については、「タスクメタデータエンドポイントバージョン 4」を参照してください。
- 2020 年 7 月 30 日以降、プラットフォームバージョン `1.4.0` を使用して Fargate で起動されたすべての新しい Amazon ECS タスクは、Network Load Balancer を使用して、UDP トラフィックを Fargate タスク上の Amazon ECS にルーティングできるようになります。詳細については、「ロードバランサーを使用して Amazon ECS サービストラフィックを分散する」を参照してください。
- 2020 年 5 月 28 日以降、プラットフォームバージョン`1.4.0`を使用してFargate で起動されたすべての新しい Amazon ECS タスクにはAWS 所有する暗号化キーを使用して AES-256 暗号化アルゴリズムで暗号化されたエフェメラルストレージが搭載されます。詳細については、「Amazon ECS 向けの Fargate タスクエフェメラルストレージ」および「Amazon ECS タスクのストレージオプション」を参照してください。
- 永続的なタスクストレージとして Amazon EFS ファイルシステムボリュームを使用するためにサポートを追加しました。詳細については、「Amazon ECS での Amazon EFS ボリュームの使用」を参照してください。
- エフェメラルタスクストレージは、タスクごとに最低 20 GB に増加しました。詳細については、「Amazon ECS 向けの Fargate タスクエフェメラルストレージ」を参照してください。
- タスクとの間のネットワークトラフィック動作を更新しました。プラットフォームバージョン 1.4.0 以降、すべての Fargate タスクは単一の Elastic Network Interface (タスク ENI と呼ばれる) を受け取り、すべてのネットワークトラフィックは VPC 内でこの ENI を通過し、VPC フローログを通じて表示されます。Amazon EC2 起動タイプのネットワークの詳細については、「EC2 起動タイプの Amazon ECS タスクネットワークオプション」を参照してください。Fargate のネットワーク設定の詳細については、「Fargate における Amazon ECS タスクのネットワークオプション」を参照してください。
- タスク ENI は、ジャンボフレームのサポートを追加しています。ネットワークインターフェイスは、最大転送単位 (MTU) で設定されます。MTU は、1 つのフレームに収まるペイロードの最大サイズです。MTU が大きいほど、1 つのフレーム内に収まるアプリケーションのペイロードが増えるため、フレームあたりのオーバーヘッドが減少し、効率が向上します。ジャンボフレームをサポートすると、オーバーヘッドが減ります。タスクと転送先とのネットワークパスでジャンボフレームをサポートすると、VPC 内に残っているすべてのトラフィックなどのオーバーヘッドが軽減されます。

- CloudWatch Container Insights には、Fargate タスクのネットワークパフォーマンスメトリクスが含まれます。詳細については、「オブザーバビリティが強化された Container Insights を使用し、Amazon ECS コンテナを監視する」を参照してください。
- タスクメタデータエンドポイントバージョン 4 のサポートを追加しました。これにより、タスクのネットワーク統計情報や、タスクが実行されているアベイラビリティーゾーンなど、Fargate タスクに関する追加情報が提供されます。詳細については、「Amazon ECS タスクメタデータエンドポイントバージョン 4」および「Fargate のタスク用の Amazon ECS タスクメタデータエンドポイントバージョン 4」を参照してください。
- コンテナの定義に SYS_PTRACE Linux パラメータのサポートを追加しました。詳細については、「Linux パラメータ」を参照してください。
- Fargateコンテナエージェントは、Amazon ECS コンテナエージェントの使用をすべての Fargate タスクに置き換えます。通常、この変更は、タスクの実行方法には影響しません。
- コンテナランタイムは Docker の代わりに Containerd を使用するようになりました。ほとんどの場合、この変更はタスクの実行方法には影響しません。コンテナランタイムで発生するいくつかのエラーメッセージは、より一般的な内容になり、Docker には言及されなくなります。詳細については、「Amazon ECS の停止したタスクのエラーメッセージ」を参照してください。
- Amazon Linux 2 に基づく。

AWS Fargate Linux プラットフォームバージョンの廃止

次の表は、AWS Fargate で廃止された、または廃止が予定されている Linux プラットフォームのバージョンを示しています。これらのプラットフォームのバージョンは、公開された廃止日までは引き続き利用できます。

廃止予定のプラットフォームのバージョンごとに、強制更新日が提供されます。強制更新日に、廃止予定のプラットフォームバージョンを指すLATEST プラットフォームバージョンを使用するすべてのサービスは、強制新規デプロイオプションを使用して更新されます。強制新規デプロイオプションを使用してサービスを更新すると、廃止予定のプラットフォームバージョンで実行されているすべてのタスクが停止され、新しいタスクはLATESTタグがその時に指すプラットフォームのバージョンを使用して起動されます。明示的なプラットフォームバージョンが設定されているスタンドアロンのタスクまたはサービスは、強制更新日の影響を受けません。

最新のプラットフォームバージョンに移行する方法については、「Amazon ECS 上の Linux プラットフォームバージョン 1.4.0 への移行」を参照してください。

プラットフォームバージョンが廃止日に達すると、そのプラットフォームバージョンは新しいタスクやサービスで使用できなくなります。非推奨のプラットフォームバージョンを明示的に使用するスタ

ンドアロンのタスクまたはサービスは、タスクが停止されるまでそのプラットフォームバージョンを使用し続けます。廃止日を過ぎると、非推奨のプラットフォームバージョンはセキュリティのアップデートやバグの修正を受けなくなります。

プラットフォームバージョン	強制更新日	廃止日
1.0.0	2020 年 10 月 26 日	2020 年 12 月 14 日
1.1.0	2020 年 10 月 26 日	2020 年 12 月 14 日
1.2.0	2020 年 10 月 26 日	2020 年 12 月 14 日
1.3.0		2026 年 6 月 15 日

現在のプラットフォームバージョンの詳細については、「Amazon ECS 向け Fargate プラットフォームバージョン」を参照してください。

廃止された Fargate Linux バージョンに関する変更ログ

1.3.0

以下は、プラットフォームバージョン `1.3.0` の変更履歴です。

- 2019 年 9 月 30 日以降、起動されたすべての新しい Fargate タスクでは、`awsfirelens`ログドライバーをサポートします。タスク定義パラメータを使用して、AWS のサービスまたは AWS パートナーネットワーク (APN) の送信先に、ログをルーティングし保存および分析を行うように、FireLens for Amazon ECS を設定します。詳細については、「Amazon ECS ログを AWS サービスまたは AWS Partner に送信する」を参照してください。
- Fargateタスクのタスクリサイクルが追加されました。これは、Amazon ECS サービスの一部であるタスクを更新するプロセスです。詳細については、「AWS Fargate on Amazon ECS のタスクの廃止とメンテナンス」。
- 2019 年 3 月 27 日以降、起動されたすべての新しい Fargate タスクでは、プロキシ設定、コンテナのスタートアップとシャットダウンの依存関係、コンテナ別の起動と停止のタイムアウト値を定義するための追加タスク定義パラメータを使用できます。詳細については、「プロキシ設定」、「コンテナの依存関係」、および「コンテナのタイムアウト」を参照してください。
- 2019 年 4 月 2 日以降、起動されたすべての新しい Fargate タスクでは、機密データをAWS Secrets Managerシークレットまたは AWS Systems Managerパラメーターストアのパラメーター

のいずれかに保存し、コンテナ定義でそれらを参照することにより、コンテナへの機密データの挿入をサポートします。詳細については、「Amazon ECS コンテナに機密データを渡す」を参照してください。

- 2019 年 5 月 1 日以降、起動された新しい Fargate タスクでは、`secretOptions` コンテナ定義パラメータを使用してコンテナのログ設定内の機密データを参照できます。詳細については、「Amazon ECS コンテナに機密データを渡す」を参照してください。
- 2019 年 5 月 1 日以降、起動されるすべての新しいFargate タスクでは、`splunk`ログドライバーに加えて `awslogs`ログドライバーがサポートされます。詳細については、「ストレージとログ記録」を参照してください。
- 2019 年 7 月 9 日以降に新しく起動された Fargate タスクでは、 CloudWatch Container Insights がサポートされています。詳細については、「オブザーバビリティが強化された Container Insights を使用し、Amazon ECS コンテナを監視する」を参照してください。
- 2019 年 12 月 3 日より、Fargate Spot キャパシティープロバイダーがサポートされます。詳細については、「Fargate 用の Amazon ECS クラスター」を参照してください。
- Amazon Linux 2 に基づく。

1.2.0

以下は、プラットフォームバージョン `1.2.0` の変更履歴です。

> **Note**
>
> プラットフォームバージョン `1.2.0` は今後利用できません。プラットフォームバージョンの廃止については、「AWS Fargate Linux プラットフォームバージョンの廃止」を参照してください。

- AWS Secrets Manager を使用したプライベートレジストリ認証のサポートが追加されました。詳細については、「Amazon ECS での AWS 以外のコンテナイメージの使用」を参照してください。

1.1.0

以下は、プラットフォームバージョン `1.1.0` の変更履歴です。

> **Note**
>
> プラットフォームバージョン `1.1.0` は今後利用できません。プラットフォームバージョンの廃止については、「AWS Fargate Linux プラットフォームバージョンの廃止」を参照してください。

- Amazon ECSタスクメタデータエンドポイントのサポートを追加しました。詳細については、「Fargate のタスクで使用できる Amazon ECS タスクメタデータ」を参照してください。
- コンテナ定義に追加された Docker ヘルスチェックのサポート。詳細については、「ヘルスチェック」を参照してください。
- Amazon ECSサービス検出のサポートを追加しました。詳細については、「サービス検出を使用して Amazon ECS サービスを DNS 名で接続する」を参照してください。

1.0.0

以下は、プラットフォームバージョン `1.0.0` の変更履歴です。

> **Note**
>
> プラットフォームバージョン `1.0.0` は今後利用できません。プラットフォームバージョンの廃止については、「AWS Fargate Linux プラットフォームバージョンの廃止」を参照してください。

- Amazon Linux 2017.09 に基づく。
- 初回リリース。

Fargate Windows プラットフォームバージョンの変更ログ

Windows コンテナに使用できるプラットフォームのバージョンは以下のとおりです。

1.0.0

以下は、プラットフォームバージョン `1.0.0` の変更履歴です。

- 以下の Microsoft Windows Server オペレーティングシステムに対応した初期リリース。

- Windows Server 2019 Full
- Windows Server 2019 Core
- Windows Server 2022 Full
- Windows Server 2022 Core

Amazon ECS における Fargate 上の Windows コンテナに関する考慮事項

AWS Fargate で Windows コンテナを実行する場合の相違点と考慮事項は次のとおりです。

Linux と Windows のコンテナでタスクを実行する必要がある場合は、オペレーティングシステムごとに別々のタスク定義を作成する必要があります。

AWSオペレーティングシステムのライセンス管理は、 が担当します。したがって、お客様は、追加の Microsoft ライセンスの必要はありません。

AWS Fargate の Windows コンテナは以下のオペレーティングシステムをサポートしています。

- Windows Server 2019 Full
- Windows Server 2019 Core
- Windows Server 2022 Full
- Windows Server 2022 Core

AWS Fargate の Windows コンテナは、awslogs ドライバをサポートしています。詳細については、「the section called “CloudWatch にログを送信する ”」を参照してください。

Fargate の Windows コンテナでは、以下の機能はサポートされていません。

- Amazon FSx
- ENI トランキング
- Windows コンテナ向け gMSA
- タスク用 App Mesh サービスとプロキシの統合
- タスク用 Firelens ログルーターの統合
- EFS ボリューム
- EBS ボリューム

- 以下に示すタスク定義パラメータ。
  - `maxSwap`
  - `swappiness`
  - `environmentFiles`
- Fargate Spot キャパシティープロバイダー
- イメージボリューム

  Dockerfile `volume` オプションは無視されます。代わりに、タスクの定義でバインドマウントを使用します。詳細については、「Amazon ECS でのバインドマウントの使用」を参照してください。
- Windows コンテナーでは、タスクレベルの CPU およびメモリパラメーターは無視されます。Windows コンテナではコンテナレベルリソースを指定することをお勧めします。
- タスクのメモリ
- コンテナのメモリ予約
- コンテナのポリシーを再起動する
- サービスの platformFamily を更新することはできません。

Fargate 上の Linux コンテナにおける Amazon ECS のコンテナイメージのプル動作

すべての Fargate タスクは、専用のシングルユース、シングルテナントのインスタンスで実行されます。Fargate 上で Linux コンテナを実行すると、コンテナイメージまたはコンテナイメージレイヤーはインスタンスにキャッシュされません。したがって、タスクで定義されたコンテナイメージごとに、コンテナイメージ全体を各 Fargate タスクのコンテナイメージレジストリからプルする必要があります。イメージをプルするのにかかる時間は、Fargate タスクを開始するのにかかる時間と直接相関しています。

イメージのプル時間を最適化するために、次の点を考慮してください。

コンテナイメージの近接性

コンテナイメージのダウンロードにかかる時間を短縮するには、データをコンピューティングにできるだけ近い場所に配置します。コンテナイメージをインターネット経由または AWS リージョン 間でプルすると、ダウンロード時間に影響する可能性があります。コンテナイメージは、タスクを実行するのと同じリージョンに保存することをお勧めします。コンテナイメージ

を Amazon ECR に保存する場合は、VPC インターフェイスエンドポイントを使用することでイメージのプル時間をさらに短縮できます。詳細については、「Amazon ECR ユーザーガイド」の「Amazon ECR インターフェイス VPC エンドポイント (AWS PrivateLink)」を参照してください。

コンテナイメージのサイズ削減

コンテナイメージのサイズは、ダウンロード時間に直接影響します。コンテナイメージのサイズまたはコンテナイメージレイヤーの数を減らすことで、イメージのダウンロードにかかる時間を短縮できます。軽量なベースイメージ (最小の Amazon Linux 2023 コンテナイメージなど) は、従来のオペレーティングシステムのベースイメージに基づくイメージよりも大幅に小さくすることができます。最小イメージの詳細については、「Amazon Linux 2023 ユーザーガイド」の「AL2023 の最小コンテナイメージ」を参照してください。

代替の圧縮アルゴリズム

多くの場合、コンテナイメージレイヤーは、コンテナイメージレジストリにプッシュされるときに圧縮されます。コンテナイメージレイヤーを圧縮することで、ネットワーク経由で転送され、コンテナイメージレジストリに保存されるデータの量を削減できます。コンテナランタイムによってコンテナイメージレイヤーがインスタンスにダウンロードされた後、そのレイヤーは解凍されます。使用される圧縮アルゴリズムとランタイムに使用できる vCPU の量は、コンテナイメージの解凍にかかる時間に影響します。Fargate では、タスクのサイズを増やすか、よりパフォーマンスの高い zstd 圧縮アルゴリズムを利用することで、解凍にかかる時間を短縮できます。詳細については、GitHub の「zstd」を参照してください。Fargate のイメージを実装する方法については、「Reducing AWS Fargate Startup Times with zstd Compressed Container Images」を参照してください。

コンテナイメージの遅延読み込み

大きなコンテナイメージ (> 250 mb) の場合、すべてのコンテナイメージをダウンロードするのではなく、コンテナイメージを遅延読み込みするのが最適である場合があります。Fargate では、Seekable OCI (SOCI) を使用して、コンテナイメージレジストリからコンテナイメージを遅延読み込みできます。詳細については、GitHub の「soci-snapshotter」と「Seekable OCI (SOCI) を使ったコンテナイメージの遅延読み込み」を参照してください。

Fargate 上の Windows コンテナにおける Amazon ECS のコンテナイメージのプル動作

Fargate Windows は、Microsoft から提供された直近の月と前月のサーバーコアのベースイメージをキャッシュします。これらのイメージは、毎週火曜日のパッチで更新される KB /ビルド番号のパッチと一致します。例えば、Microsoft は 2024 年 4 月 9 日に Windows Server 2019 用の KB5036896 (17763.5696) をリリースしました。前月の KB (2024 年 3 月 12 日) は KB5035849 (17763.5576) でした。そのため、プラットフォーム `WINDOWS_SERVER_2019_CORE` および `WINDOWS_SERVER_2019_FULL` では、以下のコンテナイメージがキャッシュされました。

- `mcr.microsoft.com/windows/servercore:ltsc2019`
- `mcr.microsoft.com/windows/servercore:10.0.17763.5696`
- `mcr.microsoft.com/windows/servercore:10.0.17763.5576`

さらに、Microsoft は 2024 年 4 月 9 日に Windows Server 2022 用の KB5036909 (20348.2402) をリリースしました。前月の KB (2024 年 3 月 12 日) は KB5035857 (20348.2340) でした。そのため、プラットフォーム `WINDOWS_SERVER_2022_CORE` および `WINDOWS_SERVER_2022_FULL` では以下のコンテナイメージがキャッシュされました。

- `mcr.microsoft.com/windows/servercore:ltsc2022`
- `mcr.microsoft.com/windows/servercore:10.0.20348.2402`
- `mcr.microsoft.com/windows/servercore:10.0.20348.2340`

Amazon ECS 向けの Fargate タスクエフェメラルストレージ

AWS Fargate の Linux コンテナでホストされている各 Amazon ECS タスクはプロビジョニング時にバインドマウントのために次のエフェメラルストレージを受け取ります。これらをマウントし、タスク定義内で `volumes`、`mountPoints` および `volumesFrom` パラメータを使用しているコンテナ間で共有することが可能です。この設定は AWS Fargate の Windows コンテナではサポートされません。

Fargate Linux コンテナプラットフォームのバージョン

バージョン 1.4.0 以降

プラットフォームバージョン `1.4.0` 以降を使用している Fargate でホストされている Amazon ECS タスクは、デフォルトで、最低 20 GiB のエフェメラルストレージを受け取ります。エフェメラルストレージの総量は、最大 200 GiB まで増やすことができます。これを行うには、タスク定義内で `ephemeralStorage` パラメータを設定します。

プル、圧縮、および非圧縮されたコンテナイメージは、エフェメラルストレージに格納されます。タスクを使用するエフェメラルストレージの総量を判断するには、タスクが割り当てられているエフェメラルストレージの総量から、コンテナイメージが使用するストレージの容量を差し引く必要があります。

2020 年 5 月 28 日以降に開始されたプラットフォームバージョン `1.4.0` 以降を使用するタスクでは、エフェメラルストレージが AES-256 暗号化アルゴリズムにより暗号化されます。このアルゴリズムは AWS が所有する暗号化キーを使用するか、独自のカスタマーマネージドキーを作成できます。詳細については、「Customer managed keys for AWS Fargate ephemeral storage」を参照してください。

2022 年 11 月 18 日以降に開始されたプラットフォームバージョン `1.4.0` 以降を使用するタスクでは、エフェメラルストレージの使用量がタスクメタデータエンドポイントを通じて報告されます。タスク内のアプリケーションは、タスクメタデータエンドポイントのバージョン 4 に対してクエリを実行して、エフェメラルストレージの予約サイズと使用量を取得できます。

さらに、Container Insights をオンにすると、エフェメラルストレージの予約サイズと使用量が Amazon CloudWatch Container Insights に送信されます。

> **Note**
>
> Fargate はディスク上のスペースを予約します。スペースは Fargate によってのみ使用されます。これには課金されることはありません。これらのメトリクスには表示されません。ただし、この追加ストレージは、`df` などの他のツールでも確認できます。

バージョン 1.3.0 以前

プラットフォームバージョン `1.3.0` 以前を使用する Fargate タスクでの Amazon ECS の場合、各タスクは次のエフェメラルストレージを受け取ります。

- 10 GB の Docker Layer ストレージ

> **Note**
>
> この量には、圧縮および非圧縮のコンテナイメージのアーティファクトの両方が含まれます。

- ボリュームマウント用の追加 4 GB。これらをマウントし、タスク定義内で `volumes`、`mountPoints` および `volumesFrom` パラメータを使用しているコンテナ間で共有することが可能です。

Fargate Windows コンテナプラットフォームのバージョン

バージョン 1.0.0 以降

プラットフォームバージョン `1.0.0` 以降を使用している Fargate でホストされている Amazon ECS タスクは、デフォルトで、最低 20 GiB のエフェメラルストレージを受け取ります。エフェメラルストレージの総量は、最大 200 GiB まで増やすことができます。これを行うには、タスク定義内で `ephemeralStorage` パラメータを設定します。

プル、圧縮、および非圧縮されたコンテナイメージは、エフェメラルストレージに格納されます。タスクが使用するエフェメラルストレージの総量を判断するには、タスクに割り当てられたエフェメラルストレージの総量から、コンテナイメージが使用するストレージの容量を差し引く必要があります。

詳細については、「Amazon ECS でのバインドマウントの使用」を参照してください。

Amazon ECS 向け AWS Fargate エフェメラルストレージ用のカスタマーマネージドキー

AWS Fargate は、エフェメラルストレージに保存されている Amazon ECS タスクのデータを暗号化するカスタマーマネージドキーをサポートし、規制の影響を受けるお客様が内部セキュリティポリシーを満たすことができるよう支援しています。お客様は、Fargate のサーバーレスのメリットを引き続き享受しながら、コンプライアンス監査者に対してセルフマネージドストレージ暗号化の強化された可視性を提供できます。Fargate は、デフォルトで Fargate 管理のエフェメラルストレージ暗号化を備えていますが、お客様は財務情報や健康関連情報などの機密データを暗号化する際に、独自のセルフマネージドキーを使用することもできます。

AWS KMS に独自のキーをインポートするか、AWS KMS でキーを作成できます。これらのセルフマネージドキーは AWS KMS に保存され、ローテーション、無効化、削除などの標準的な AWS KMS のライフサイクルアクションを実行します。CloudTrail ログで、キーのアクセスと使用状況を監査できます。

デフォルトでは、KMS キーはキーごとに 50,000 の許可をサポートします。Fargate は、各カスタマーマネージドキーのタスクに 1 つの AWS KMS 許可を使用するため、1 つのキーに対して最大 50,000 の同時タスクをサポートします。この数を増やしたい場合は、ケースバイケースで承認される制限の引き上げをリクエストすることができます。

Fargate では、カスタマーマネージドキーの使用に追加の料金は発生しません。ストレージと API リクエストに AWS KMS キーを使用するための標準料金のみが請求されます。

トピック

- Amazon ECS 向け Fargate エフェメラルストレージの暗号化キーを作成する
- Amazon ECS 向け Fargate エフェメラルストレージの AWS KMS キーを管理する

Amazon ECS 向け Fargate エフェメラルストレージの暗号化キーを作成する

Fargate エフェメラルストレージに保存されているデータを暗号化するためのカスタマーマネージドキーを作成します。

> **Note**
>
> カスタマーマネージドキーによる Fargate エフェメラルストレージの暗号化は、Windows タスククラスターでは利用できません。
> カスタマーマネージドキーによる Fargate エフェメラルストレージの暗号化は、`1.4.0` より前のバージョンの `platformVersions` では利用できません。
> Fargate は、Fargate のみが使用するエフェメラルストレージのスペースを予約します。そのスペースに対して料金は発生しません。割り当ては、カスタマー管理以外のキータスクとは異なる場合がありますが、スペースの合計は変わりません。この変更については、`df` などのツールで確認できます。
> マルチリージョンキーは、Fargate エフェメラルストレージではサポートされていません。
> KMS キーエイリアスは、Fargate エフェメラルストレージではサポートされていません。

AWS KMS で Fargate のエフェメラルストレージを暗号化するカスタマーマネージドキー (CMK) を作成するには、次のステップに従ってください。

1. https://console.aws.amazon.com/kms に移動します。
2. 手順については、「AWS Key Management Service デベロッパーガイド」の「Creating Keys」を参照してください。
3. AWS KMS キーを作成するときは、キーポリシーで Fargate サービスに関連する AWS KMS オペレーションのアクセス許可を必ず付与してください。Amazon ECS クラスターリソースでカスタマー管理キーを使用するには、ポリシーで次の API オペレーションを許可する必要があります。

   - `kms:GenerateDataKeyWithoutPlainText` - `GenerateDataKeyWithoutPlainText` を呼び出し、提供された AWS KMS キーから暗号化されたデータキーを生成します。
   - `kms:CreateGrant` - カスタマーマネージドキーに許可を追加します。指定された AWS KMS キーへのアクセス制御を付与します。これは、Amazon ECS Fargate が必要なグラントオペレーションへのアクセスを許可するものです。詳細については、「AWS Key Management Service デベロッパーガイド」の「AWS KMS でのグラント」を参照してください。これにより、Amazon ECS Fargate で以下を行うことができます。
     - `Decrypt` から AWS KMS を呼び出し、エフェメラルストレージのデータを復号するための暗号化キーを取得します。
     - `RetireGrant` へのサービスを許可するために、削除プリンシパルを設定します。
   - `kms:DescribeKey` — カスタマーマネージドキーの詳細を提供し、キーが対称、かつ有効である場合に、Amazon ECS がキーを検証できるようにします。

   次の例は、暗号化するターゲットキーに適用する AWS KMS キーポリシーを示しています。ポリシーステートメントの例を使用する際は、*###############*を独自の情報に置き換えます。通常どおり、必要なアクセス許可のみを設定しますが、エラーを回避するために、少なくとも 1 人のユーザーにアクセス許可付きの AWS KMS を提供する必要があります。

```
{
      "Sid": "Allow generate data key access for Fargate tasks.",
      "Effect": "Allow",
      "Principal": { "Service":"fargate.amazonaws.com" },
      "Action": [
        "kms:GenerateDataKeyWithoutPlaintext"
      ],
      "Condition": {
        "StringEquals": {
          "kms:EncryptionContext:aws:ecs:clusterAccount": [
            "customerAccountId"
```

```
          ],
          "kms:EncryptionContext:aws:ecs:clusterName": [
             "clusterName"
          ]
        }
      },
      "Resource": "*"
    },
    {
      "Sid": "Allow grant creation permission for Fargate tasks.",
      "Effect": "Allow",
      "Principal": { "Service":"fargate.amazonaws.com" },
      "Action": [
        "kms:CreateGrant"
      ],
      "Condition": {
        "StringEquals": {
          "kms:EncryptionContext:aws:ecs:clusterAccount": [
            "customerAccountId"
          ],
          "kms:EncryptionContext:aws:ecs:clusterName": [
             "clusterName"
          ]
        },
       "ForAllValues:StringEquals": {
          "kms:GrantOperations": [
             "Decrypt"
          ]
       }
      },
      "Resource": "*"
    },
    {
      "Sid": "Allow describe key permission for cluster operator - CreateCluster
 and UpdateCluster.",
      "Effect": "Allow",
      "Principal": { "AWS":"arn:aws:iam::customerAccountId:role/customer-chosen-
role" },
      "Action": [
        "kms:DescribeKey"
      ],
      "Resource": "*"
    }
```

Fargate タスクでは、キーを使用する暗号化オペレーションに aws:ecs:clusterAccount および aws:ecs:clusterName 暗号化コンテキストキーを使用します。特定のアカウントやクラスターへのアクセスを制限するには、これらのアクセス許可を追加する必要があります。クラスターを指定するときは、ARN ではなくクラスター名を使用してください。

詳しくは、AWS KMS デベロッパーガイドの Encryption context を参照してください。

クラスターを作成または更新する場合、条件キー fargateEphemeralStorageKmsKeyId を使用するオプションを利用できます。この条件キーにより、お客様は IAM ポリシーをよりきめ細かく制御できます。fargateEphemeralStorageKmsKeyId 設定の更新は、新しいサービスのデプロイでのみ有効になります。

以下は、お客様が特定の承認された AWS KMS キーのセットにのみアクセス許可を付与できるようにする例です。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:CreateCluster",
        "ecs:UpdateCluster"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "ecs:fargate-ephemeral-storage-kms-key": "arn:aws:kms:us-
west-2:111122223333:key/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
        }
      }
    }
  ]
}
```

次に、クラスターに既に関連付けられている AWS KMS キーの削除を拒否する例を示します。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Effect": "Deny",
    "Action": [
        "ecs:CreateCluster",
        "ecs:UpdateCluster"
    ],
    "Resource": "*",
    "Condition": {
      "Null": {
        "ecs:fargate-ephemeral-storage-kms-key": "true"
      }
    }
  }
}
```

お客様は、AWS CLI、`describe-tasks`、`describe-cluster` または `describe-services` コマンドを使用して、管理されていないタスクまたはサービスタスクが、キーを使用して暗号化されているかどうかを確認できます。

詳細については、「AWS KMS デベロッパーガイド」の「Condition keys for AWS KMS」を参照してください。

AWS マネジメントコンソール

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
2. 左側のナビゲーションで [クラスター]を選択し、右上で [クラスターの作成] を選択するか、既存のクラスターを選択します。既存のクラスターの場合は、右上の [クラスターの更新] を選択します。
3. ワークフローの [暗号化] セクションでは、[マネージド型ストレージ] と [Fargate エフェメラルストレージ] で AWS KMS キーを選択することもできます。ここで [AWS KMS キーの作成] を選択することもできます。
4. 新しいクラスターの作成が完了したら [作成] を選択し、既存のクラスターを更新する場合は [更新] を選択します。

AWS CLI

以下は、クラスターを作成し、AWS CLI を使用して Fargate エフェメラルストレージを設定する例です (*##*の値は独自の値に置き換えてください)。

```
aws ecs create-cluster --cluster clusterName \
--configuration '{"managedStorageConfiguration":
{"fargateEphemeralStorageKmsKeyId":"arn:aws:kms:us-
west-2:012345678901:key/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"}}'
{
    "cluster": {
        "clusterArn": "arn:aws:ecs:us-west-2:012345678901:cluster/clusterName",
        "clusterName": "clusterName",
        "configuration": {
            "managedStorageConfiguration": {
                "fargateEphemeralStorageKmsKeyId": "arn:aws:kms:us-
west-2:012345678901:key/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
            }
        },
        "status": "ACTIVE",
        "registeredContainerInstancesCount": 0,
        "runningTasksCount": 0,
        "pendingTasksCount": 0,
        "activeServicesCount": 0,
        "statistics": [],
        "tags": [],
        "settings": [],
        "capacityProviders": [],
        "defaultCapacityProviderStrategy": []
    },
    "clusterCount": 5
}
```

CloudFormation

以下は、クラスターを作成し、CloudFormation を使用して Fargate エフェメラルストレージを設定するテンプレートの例です (*##*の値は独自の値に置き換えてください)。

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  MyCluster:
    Type: AWS::ECS::Cluster
    Properties:
```

```
      ClusterName: "clusterName"
      Configuration:
        ManagedStorageConfiguration:
          FargateEphemeralStorageKmsKeyId: "arn:aws:kms:us-
west-2:012345678901:key/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

Amazon ECS 向け Fargate エフェメラルストレージの AWS KMS キーを管理する

Fargate エフェメラルストレージを暗号化するための AWS KMS キーを作成またはインポートしたら、他の AWS KMS キーと同じ方法で管理します。

AWS KMS キーの自動ローテーション

キーの自動ローテーションを有効にするか、手動でローテーションすることができます。キーの自動ローテーションは、キーの新しい暗号化マテリアルを生成することによって、キーを毎年ローテーションします。AWS KMS は、暗号化マテリアルの以前のバージョンもすべて保存するため、以前のキーバージョンを使用したデータを復号化することができます。ローテーションされたマテリアルは、キーを削除するまで AWS KMS によって削除されることはありません。

キーの自動ローテーションはオプションであり、いつでも有効または無効にできます。

AWS KMS キーの無効化または取り消し

AWS KMS でカスタマーマネージドキーを無効にした場合でも、タスクの実行に影響はなく、ライフサイクルを通じて引き続き機能します。無効化または取り消しされたキーが新しいタスクで使用されている場合は、キーにアクセスできないためタスクは失敗します。CloudWatch アラームなどを設定し、既に暗号化されたデータを復号するために無効化されたキーが再び必要になることがないことを確認する必要があります。

AWS KMS キーの削除

キーの削除は最終的な手段であり、削除されたキーが再び必要になることがないと確信している場合にのみ実行する必要があります。削除されたキーを使用しようとする新しいタスクは、そのキーにアクセスできないため失敗します。AWS KMS では、キーを削除するのではなく、キーを無効にすることをお勧めしています。キーを削除する必要があると思われる場合は、最初にキーを無効にした後、CloudWatch アラームを設定して、不要であることを確認することをお勧めします。キーを削除する場合、AWS KMS では少なくとも 7 日間の猶予期間が付与されているため、その期間に削除を撤回することも可能です。

AWS KMS キーアクセスの監査

CloudTrail ログを使用して、AWS KMS キーへのアクセスを監査できます。AWS KMS オペレーション CreateGrant、GenerateDataKeyWithoutPlaintext、および Decrypt を確認できます。これらのオペレーションでは、EncryptionContext が口グインした CloudTrail の一部として aws:ecs:clusterAccount および aws:ecs:clusterName も表示されます。

以下は、GenerateDataKeyWithoutPlaintext、GenerateDataKeyWithoutPlaintext (DryRun)、CreateGrant、CreateGrant (DryRun)、RetireGrant の CloudTrail イベントの例です (*##* の値は独自の値に置き換えてください)。

GenerateDataKeyWithoutPlaintext

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "ec2-frontend-api.amazonaws.com"
    },
    "eventTime": "2024-04-23T18:08:13Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKeyWithoutPlaintext",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "ec2-frontend-api.amazonaws.com",
    "userAgent": "ec2-frontend-api.amazonaws.com",
    "requestParameters": {
        "numberOfBytes": 64,
        "keyId": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111",
        "encryptionContext": {
            "aws:ecs:clusterAccount": "account-id",
            "aws:ebs:id": "vol-xxxxxxx",
            "aws:ecs:clusterName": "cluster-name"
        }
    },
    "responseElements": null,
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "readOnly": true,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
```

```
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "account-id",
    "sharedEventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
    "eventCategory": "Management"
}
```

GenerateDataKeyWithoutPlaintext (DryRun)

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "fargate.amazonaws.com"
    },
    "eventTime": "2024-04-23T18:08:11Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKeyWithoutPlaintext",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "fargate.amazonaws.com",
    "userAgent": "fargate.amazonaws.com",
    "errorCode": "DryRunOperationException",
    "errorMessage": "The request would have succeeded, but the DryRun option is
 set.",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111",
        "dryRun": true,
        "numberOfBytes": 64,
        "encryptionContext": {
            "aws:ecs:clusterAccount": "account-id",
            "aws:ecs:clusterName": "cluster-name"
        }
    },
    "responseElements": null,
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "readOnly": true,
    "resources": [
        {
            "accountId": "AWS Internal",
```

```
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "account-id",
    "sharedEventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
    "eventCategory": "Management"
}
```

CreateGrant

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "ec2-frontend-api.amazonaws.com"
    },
    "eventTime": "2024-04-23T18:08:13Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "ec2-frontend-api.amazonaws.com",
    "userAgent": "ec2-frontend-api.amazonaws.com",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111",
        "granteePrincipal": "fargate.us-west-2.amazonaws.com",
        "operations": [
            "Decrypt"
        ],
        "constraints": {
            "encryptionContextSubset": {
                "aws:ecs:clusterAccount": "account-id",
                "aws:ebs:id": "vol-xxxx",
                "aws:ecs:clusterName": "cluster-name"
            }
        },
        "retiringPrincipal": "ec2.us-west-2.amazonaws.com"
    },
    "responseElements": {
```

```
        "grantId":
 "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855",
        "keyId": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
    },
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "readOnly": false,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "account-id",
    "sharedEventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
    "eventCategory": "Management"
}
```


CreateGrant (DryRun)


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "fargate.amazonaws.com"
    },
    "eventTime": "2024-04-23T18:08:11Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "fargate.amazonaws.com",
    "userAgent": "fargate.amazonaws.com",
    "errorCode": "DryRunOperationException",
    "errorMessage": "The request would have succeeded, but the DryRun option is
 set.",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111",
```

```
        "granteePrincipal": "fargate.us-west-2.amazonaws.com",
        "dryRun": true,
        "operations": [
            "Decrypt"
        ],
        "constraints": {
            "encryptionContextSubset": {
                "aws:ecs:clusterAccount": "account-id",
                "aws:ecs:clusterName": "cluster-name"
            }
        }
    },
    "responseElements": null,
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "readOnly": false,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "account-id",
    "sharedEventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
    "eventCategory": "Management"
}
```

RetireGrant

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2024-04-20T18:37:38Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "RetireGrant",
    "awsRegion": "us-west-2",
```

```
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": null,
    "responseElements": {
        "keyId": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
    },
    "additionalEventData": {
        "grantId":
 "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855"
    },
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "readOnly": false,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "account-id",
    "sharedEventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
    "eventCategory": "Management"
}
```


AWS Fargate on Amazon ECS のタスクの廃止とメンテナンス

AWS は、AWS Fargate の基盤インフラストラクチャを維持する責任を負います。AWS は、プラットフォームバージョンのリビジョンを、インフラストラクチャ用の新しいリビジョンに置き換える必要があるタイミングを決定します。これはタスクの廃止と呼ばれます。プラットフォームバージョンのリビジョンが廃止されると、AWS からタスクの廃止通知が送信されます。サポート対象のプラットフォームバージョンを定期的に更新して、Fargate ランタイムソフトウェアと、オペレーティングシステムやコンテナランタイムなどの内在的な依存関係の更新を含む新しいリビジョンを導入しています。新しいリビジョンが利用可能になった後、すべてのお客様のワークロードが Fargate プラットフォームバージョンの最新リビジョンで実行されるように、古いリビジョンを廃止します。リビジョンが廃止されると、そのリビジョンで実行されていたすべてのタスクが停止します。

Amazon ECS タスクは、サービスタスクまたはスタンドアロンタスクのどちらかに分類されます。サービスタスクは、Amazon ECS サービスの一部としてデプロイされ、Amazon ECS スケジューラによって制御されます。詳細については、「Amazon ECS サービス」を参照してください。スタンドアロンタスクとは、Amazon ECS RunTask API によって開始されるタスクであり、直接または、Amazon EventBridge によって開始されるスケジュールされたタスク、AWS Batch、AWS Step Functions などの外部スケジューラにより開始されます。Amazon ECS スケジューラはタスクを自動的に置き換えるため、サービスタスクのタスク廃止への対応のために措置を講じる必要はありません。

スタンドアロンタスクについては、タスク廃止への対応として追加の処理を実行する必要が生じる場合があります。詳細については、「Amazon ECS はスタンドアロンタスクを自動的に処理できますか?」を参照してください。

サービスタスクについては、AWS よりも前にこれらのタスクを置き換える場合を除き、タスク廃止に対する措置を講じる必要はありません。Amazon ECS スケジューラーはタスクを停止すると、`maximumPercent` を使用して新しいタスクを起動し、サービスに必要な数を維持しようとします。ベストプラクティスに従って、タスクの廃止の影響を最小限に抑えます。REPLICA サービススケジューラを使用するサービスのデフォルトの `maximumPercent` 値は 200% です。そのため、AWS Fargate が廃止タスクを開始すると、Amazon ECS はまず新しいタスクをスケジュールし、それが実行されるのを待ってから、古いタスクを廃止します。`maximumPercent` 値を 100% に設定すると、Amazon ECS は最初にタスクを停止し、次にタスクを置き換えます。

スタンドアロンタスクの廃止では、AWS タスク廃止日およびそれ以降のタスクを停止します。タスクが停止したとき、Amazon ECS は代替のタスクを起動しません。これらのタスクを継続して実行する必要がある場合は、通知で示された時間より前に、実行中のタスクを停止し、代替タスクを起動する必要があります。そのためユーザーには、スタンドアロンタスクの状態をモニタリングすることと、必要に応じて停止したタスクを置き換えるロジックの実装が推奨されます。

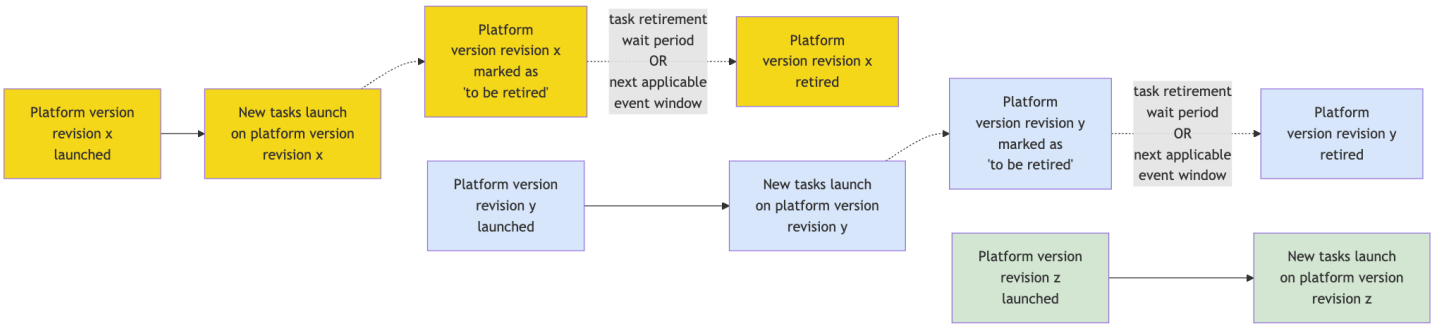
タスクが停止したシナリオに関係なく、`describe-tasks` を実行することができます。レスポンス内の `stoppedReason` は `ECS is performing maintenance on the underlying infrastructure hosting the task` です。

タスクのメンテナンスは、新しいプラットフォームバージョンのリビジョンを新しいリビジョンで置き換える必要があるときに適用されます。内在する Fargate ホストに問題がある場合、Amazon ECS はタスク終了通知を行わず、ホストを交換します。

タスク廃止通知の概要

AWS がプラットフォームバージョンリビジョンを要廃止としてマークすると、そのプラットフォームバージョンリビジョンで実行されているすべてのタスクがすべてのリージョンで特定されます。

以下の図は、新しいリビジョンの起動からプラットフォームリビジョンの廃止までの、Fargate プラットフォームバージョンリビジョンのライフサイクルを示しています。



以下は、ライフサイクルの詳細情報です。

- 新しいプラットフォームバージョンリビジョンが起動されたら、新しいタスクのすべてがこのリビジョンでスケジュールされます。
- 実行されているスケジュール済みの既存のタスクは、タスクの期間が終わるまで元々設定されていたリビジョンに残り、新しいリビジョンには移行されません。

- 新しいタスク (サービスの更新や Fargate のタスク廃止の一環として行われるタスクなど) は、起動時に利用可能な最新のプラットフォームバージョンリビジョンに設定されます。

タスク廃止通知は、AWS Health ダッシュボード、および登録された E メールアドレスへの E メールを通じて送信され、以下の情報が含まれています。

- タスクの廃止日 - タスクは、この日またはそれ以降に停止されます。
- タスクの ID (スタンドアロンタスクの場合)。
- サービスが実行されるクラスターの ID とサービスの ID (サービスタスクの場合)。
- 次に実行する必要があるステップ。

通常、AWS リージョン のサービスタスクとスタンドアロンのタスクについて、それぞれ 1 件の通知が送信されます。ただし、場合によっては、タスクタイプごとに複数のイベントを受け取ることがあります。たとえば、廃止するタスクが多すぎて通知メカニズムの制限を超える場合などです。

廃止がスケジュールされているタスクは次の方法で識別できます。

- Health Dashboard

  AWS Health 通知は Amazon EventBridge 経由で Amazon Simple Storage Service などのアーカイブストレージに送信すること、AWS Lambda 関数の実行などの自動化されたアクションを取ること、または Amazon Simple Notification Service などのその他の通知システムに送信することができます。詳細については「Amazon EventBridge による AWS Health イベントのモニタリング」を参照してください。Amazon Chime、Slack、または Microsoft Teams に通知を送信するための設定例については、GitHub の「AWS Health Aware」リポジトリを参照してください。

  EventBridge イベントの例を次に示します。

```
{
    "version": "0",
    "id": "3c268027-f43c-0171-7425-1d799EXAMPLE",
    "detail-type": "AWS Health Event",
    "source": "aws.health",
    "account": "123456789012",
    "time": "2023-08-16T23:18:51Z",
    "region": "us-east-1",
    "resources": [
        "cluster|service",
        "cluster|service"
```

```
    ],
    "detail": {
        "eventArn": "arn:aws:health:us-east-1::event/ECS/
AWS_ECS_TASK_PATCHING_RETIREMENT/AWS_ECS_TASK_PATCHING_RETIREMENT_test1",
        "service": "ECS",
        "eventScopeCode": "ACCOUNT_SPECIFIC",
        "communicationId":
 "7988399e2e6fb0b905ddc88e0e2de1fd17e4c9fa60349577446d95a18EXAMPLE",
        "lastUpdatedTime": "Wed, 16 Aug 2023 23:18:52 GMT",
        "eventRegion": "us-east-1",
        "eventTypeCode": "AWS_ECS_TASK_PATCHING_RETIREMENT",
        "eventTypeCategory": "scheduledChange",
        "startTime": "Wed, 16 Aug 2023 23:18:51 GMT",
        "endTime": "Fri, 18 Aug 2023 23:18:51 GMT",
        "eventDescription": [
            {
                "language": "en_US",
                "latestDescription": "\\nA software update has been deployed to
 Fargate which includes CVE patches or other critical patches. No action is required
 on your part. All new tasks launched automatically uses the latest software
 version. For existing tasks, your tasks need to be restarted in order for these
 updates to apply. Your tasks running as part of the following ECS Services will
 be automatically updated beginning Wed, 16 Aug 2023 23:18:51 GMT.\\n\\nAfter Wed,
 16 Aug 2023 23:18:51 GMT, the ECS scheduler will gradually replace these tasks,
 respecting the deployment settings for your service. Typically, services should
 see little to no interruption during the update and no action is required. When AWS
 stops tasks, AWS uses the minimum healthy percent (1) and launches a new task in
 an attempt to maintain the desired count for the service. By default, the minimum
 healthy percent of a service is 100 percent, so a new task is started first before
 a task is stopped. Service tasks are routinely replaced in the same way when
 you scale the service or deploy configuration changes or deploy task definition
 revisions. If you would like to control the timing of this restart you can update
 the service before Wed, 16 Aug 2023 23:18:51 GMT, by running the update-service
 command from the ECS command-line interface specifying force-new-deployment for
 services using Rolling update deployment type. For example:\\n\\n$ aws ecs update-
service -service service_name \\\n--cluster cluster_name -force-new-deployment\
\n\\nFor services using Blue/Green deployment type with AWS CodeDeploy:\\nPlease
 refer to create-deployment document (2) and create new deployment using same task
 definition revision.\\n\\nFor further details on ECS deployment types, please
 refer to ECS Deployment Developer Guide (1).\\nFor further details on Fargate's
 update process, please refer to the AWS Fargate User Guide (3).\\nIf you have
 any questions or concerns, please contact AWS Support (4).\\n\\n(1) https://
docs.aws.amazon.com/AmazonECS/latest/developerguide/deployment-types.html\\n(2)
 https://docs.aws.amazon.com/cli/latest/reference/deploy/create-deployment.html\\n(3)
```

```
 https://docs.aws.amazon.com/AmazonECS/latest/userguide/task-maintenance.html\\n(4)
 https://aws.amazon.com/support\\n\\nA list of your affected resources(s) can be
 found in the 'Affected resources' tab in the 'Cluster/ Service' format in the AWS
 Health Dashboard. \\n\\n"
            }
        ],
      "affectedEntities": [
                {
                    "entityValue": "arn:aws:ecs:eu-west-1:111222333444:task/
examplecluster/00805ce1d81940b5a37398e5a2c23333"
                },
                {
                    "entityValue": "arn:aws:ecs:eu-west-1:111222333444:task/
examplecluster/00805ce1d81940b5a37398e5a2c25555"
                }
            }
        ]
    }
}
```

- E メール

  AWS アカウント ID に対して登録された E メールアドレスに E メールが送信されます。

タスク廃止の準備を行う方法については、「Amazon ECS での AWS Fargate タスク廃止に備える」を参照してください。

タスク廃止通知の影響を受けるリソースのステータス

タスク廃止通知に記載されている影響を受ける各リソースには、そのタスクがまだ廃止待ちかどうかを反映するステータスがあります。タスクの廃止が保留中の場合、そのステータスは [障害がある] です。これは、廃止がスケジュールされているプラットフォームバージョンリビジョンでタスクがまだ実行されていることを示します。

タスクの実行が停止すると、ステータスは通常 24 時間以内に自動的に [解決済み] に変わります。これは、タスクメンテナンスの一環として AWS がタスクを廃止したか、スケジュールされた廃止前にご自身でタスクを停止または置き換えたかにかかわらず適用されます。通知の影響を受けるすべてのリソースが [解決済み] になると、 AWS は廃止通知を閉じます。

タスク廃止をオプトアウトすることはできますか?

いいえ。AWS の責任共有モデルの一環として、AWS は AWS Fargate の基盤となるインフラストラクチャの管理と維持に対する責任を担います。これには、セキュリティと安定性を確保するための定期的なプラットフォーム更新の実施が含まれます。これらの更新は AWS が自動的に適用するため、お客様がオプトアウトすることはできません。これは、EC2 インスタンスでワークロードを実行する代わりに AWS Fargate を使用するときの主なメリットであり、基盤となるプラットフォームを維持する責任は AWS が担います。このモデルは、インフラストラクチャのメンテナンスではなく、アプリケーションに集中することを可能にします。AWS は、これらのプラットフォーム更新を自動的に適用することによって、お客様からのアクションを必要とすることなく、Fargate 環境を最新かつセキュアに保つことができます。これは、Fargate でワークロードを実行するための、セキュアで信頼性が高いコンテナ化された環境の提供に役立ちます。

他の AWS サービスを通じてタスク終了通知を受け取ることはできますか?

AWS は、タスク終了通知を、Health Dashboard および AWS アカウント の主要メール連絡先に送信します。Health Dashboard は、EventBridge を含む他の AWS サービスとのさまざまな統合を提供します。EventBridge を使用して、通知の可視性を自動化できます (たとえば、メッセージを ChatOps ツールに転送するなど) 。詳細については、「ソリューションの概要: タスク廃止通知の取得」を参照してください。

タスクの廃止はスケジュールされた後に変更できますか?

いいえ。スケジュールはタスク廃止の待機時間 (デフォルトで 7 日間) に基づいています。さらに時間が必要な場合は、待機期間を 14 日間に設定することもできます。詳細については、「ステップ 2: タスク廃止通知をキャプチャしてチームに警告し、措置を講じる」を参照してください。

2025 年 12 月 18 日より、Amazon ECS で Fargate タスク向けの Amazon EC2 イベントウィンドウを設定できるようになりました。業務時間中の中断を避けるためにタスク廃止を週末にスケジュールするなど、廃止のタイミングを正確に制御する必要がある場合は、タスク、サービス、またはクラスターに Amazon EC2 イベントウィンドウを設定できます。「ステップ 1: タスクの待機時間を設定、または Amazon EC2 イベントウィンドウを使用する」を参照してください。この設定の変更は、今後スケジュールされる廃止に適用されます。現在予定されている廃止には影響しません。また、Fargate タスクに Amazon EC2 イベントウィンドウを設定するときは、このウィンドウがタスク廃止の待機時間設定に優先します。ご不明な点がある場合は、サポート までお問い合わせください。

Amazon ECS はサービスの一部であるタスクをどのように処理しますか?

サービスタスクについては、AWS よりも前にこれらのタスクを置き換える場合を除き、タスク廃止に対応するための措置を講じる必要はありません。Amazon ECS スケジューラは、タスクを停止する際、最小正常稼働率を使用して新しいタスクを起動し、サービスに必要とされる数を維持しようとします。Fargate のタスク廃止による影響を最小限に抑えるには、Amazon ECS のベストプラクティスに従ってワークロードをデプロイする必要があります。例えば、ウェブまたは API サーバーなど、Amazon ECS サービスとしてステートレスアプリケーションをデプロイする場合、お客様は複数のタスクレプリカをデプロイして、minimumHealthyPercent を 100% に設定する必要があります。サービスの最小ヘルス率はデフォルトで 100% です。そのため、Fargate が廃止タスクを開始すると、Amazon ECS はまず新しいタスクをスケジュールし、それが実行されるのを待ってから、古いタスクを廃止します。サービスタスクは、サービスをスケールする、設定変更をデプロイする、またはタスク定義リビジョンをデプロイするときと同じ方法で、タスク廃止の一環として定期的に置き換えられます。タスク廃止プロセスへの準備を行うには、「Amazon ECS での AWS Fargate タスク廃止に備える」を参照してください。

Amazon ECS はスタンドアロンタスクを自動的に処理できますか?

いいえ。AWS は、RunTask が開始したスタンドアロンタスク、スケジュールされたタスク (EventBridge Scheduler の使用など)、AWS Batch、AWS Step Functions に対する交換タスクを作成できません。Amazon ECS はサービスの一部であるタスクのみを管理します。

タスク廃止中のサービス可用性のトラブルシューティング

タスクの廃止中に Amazon ECS が代替タスクを開始できない場合、サービスの可用性に影響が出る可能性があります。これは、次のような誤ったユーザー設定が原因で発生する可能性があります。

- IAM ロールが欠落しているか、正しく設定されていない
- ターゲットサブネットの容量不足
- セキュリティグループの設定ミス
- タスク定義エラー

Amazon ECS が代替タスクを起動できない場合、廃止されたタスクは代替されずに停止され、その結果、サービスの使用可能な容量が減少し、サービスの中断を引き起こす可能性があります。サービスのタスク数と Amazon CloudWatch メトリクスをモニタリングして、廃止イベント中に代替タスクが正常に起動されることを確認します。

Amazon ECS での AWS Fargate タスク廃止に備える

タスク廃止への準備を行うには、以下の手順を実行します。

1. タスク廃止の待機期間を設定するか、Amazon EC2 イベントウィンドウを使用します。
2. タスク廃止通知をキャプチャして、チームメンバーに通知します。
3. 強制デプロイオプションでサービスを更新することにより、すべてのサービスのタスクが最新のプラットフォームリビジョンで実行されるようにすることができます。この手順は省略可能です。

ステップ 1: タスクの待機時間を設定、または Amazon EC2 イベントウィンドウを使用する

Fargate がタスク廃止を開始する時刻の設定には、`fargateTaskRetirementWaitPeriod` と `fargateEventWindows` の 2 つのアカウント設定オプションがあります。

fargateTaskRetirementWaitPeriod アカウント設定の使用

Fargate がタスク廃止を開始する時間を設定できます。デフォルトの待機時間は 7 日間です。アップデートをすぐに適用する必要があるワークロードの場合は、即時設定 (0) を選択します。これより長い時間が必要な場合は、7 日または 14 日オプションを設定します。

新しいプラットフォームバージョンのリビジョンを早く取得できるように、待機時間を短くすることをお勧めします。

待機時間を設定するには、ルートユーザーまたは管理者ユーザーで `put-account-setting-default` または `put-account-setting` を実行します。`fargateTaskRetirementWaitPeriod` オプションを `name` に使用し、`value` オプションを次のいずれかの値に設定します。

- `0` - AWS から通知が送信され、影響を受けたタスクが直ちに廃止されます。
- 7 - AWS から通知が送信され、7 日間待機してから、影響を受けたタスクの廃止が開始されます。これがデフォルトです。
- 14 - AWS から通知が送信され、14 日間待機してから、影響を受けたタスクの廃止が開始されます。

詳細については、「Amazon Elastic Container Service API リファレンス」の「put-account-setting-default」と「put-account-setting」を参照してください。

fargateEventWindows アカウント設定の使用

2025 年 12 月 18 日より、Amazon ECS で Fargate タスク向けの Amazon EC2 イベントウィンドウを設定できるようになりました。業務時間中の中断を避けるためにタスク廃止を週末にスケジュールするなど、廃止のタイミングを正確に制御する必要がある場合は、タスク、サービス、またはクラスターに Amazon EC2 イベントウィンドウを設定できます。

イベントウィンドウを使用する場合、Fargate は、ユーザーが開始したアクション、または基盤ハードウェアの劣化などの重大な正常性イベントによってタスクが停止された場合を除き、タスクが次に使用可能なウィンドウ内で少なくとも 3 日間実行されてから廃止されるようにします。

`fargateEventWindows` アカウント設定を `enabled` に設定します。`put-account-setting-default` API または `put-account-setting` API のいずれかをルートユーザーまたは管理者ユーザーとして使用できます。

各 Amazon EC2 イベントウィンドウの時間枠は少なくとも週に 4 時間 (各時間範囲は少なくとも 2 時間) にする必要があります。大規模なクラスターやサービスの場合は、長時間 (8 時間以上) のイベントウィンドウ、または時間範囲がより頻繁に実施される (少なくとも 3 日に 1 度) イベントウィンドウを設定することが推奨されます。Amazon EC2 イベントウィンドウの考慮事項については、ユーザーガイド で詳細を確認できます。AWS Fargate は、ユーザーが開始したアクション、または基盤ハードウェアの劣化などの重大な正常性イベントによってタスクが停止された場合を除き、タスクが少なくとも 3 日間実行されてから廃止されるようにします。

> **Important**
>
> イベントウィンドウ内でのタスクの置き換えはベストエフォートです。イベントウィンドウ外でタスクが廃止されていることに気付いた場合は、時間を延長する (8 時間以上) か、頻度を増やす (少なくとも 3 日に 1 度) ことを検討してください。

Amazon EC2 イベントウィンドウを Fargate タスクの廃止に適用する:

- `fargateEventWindows` アカウント設定を `enabled` に設定します。`put-account-setting-default` API または `put-account-setting` API のいずれかをルートユーザーまたは管理者ユーザーとして使用できます。これは、Fargate タスクに Amazon EC2 イベントウィンドウ機能を使用するための 1 回限りの有効化であることに注意してください。
- AWS コンソールまたは AWS CLI を使用して Amazon EC2 イベントウィンドウを作成します。CLI を使用してイベントウィンドウを作成するには、時間範囲または cron 式を

設定した EC2 `create-instance-event-window` API を使用します。レスポンス内の `InstanceEventWindowId` を記録しておきます。

```
aws ec2 create-instance-event-window \
                    --time-range
 StartWeekDay=monday,StartHour=2,EndWeekDay=wednesday,EndHour=8 \
                    --tag-specifications "ResourceType=instance-event-
window,Tags=[{Key=K1,Value=V1" \
                    --name myEventWindowName
```

また、EC2 イベントウィンドウの作成時には、cron 式を使用することも可能です。

```
aws ec2 create-instance-event-window \
                    --cron-expression "* 21-23 * * 2,3" \
                    --tag-specifications "ResourceType=instance-event-
window,Tags=[{Key=K1,Value=V1" \
                    --name myEventWindowName
```

- その後、EC2 `associate-instance-event-window` API を使用して、アカウント内の特定のサービス、クラスター、またはすべてのタスクにイベントウィンドウを関連付けることができます。
  - ECS サービスタスクの場合

    ```
    aws ec2 associate-instance-event-window \
    --instance-event-window-id iew-0abcdef1234567890 \
    --association-target "InstanceTags=[{Key=aws:ecs:serviceArn,Value=your-service-
    arn}]"
    ```

  - ECS クラスターの場合

    ```
    aws ec2 associate-instance-event-window \
    --instance-event-window-id iew-0abcdef1234567890 \
    --association-target "InstanceTags=[{Key=aws:ecs:clusterArn,Value=your-cluster-
    arn}]"
    ```

  - イベントウィンドウをアカウント内のすべてのタスクに関連付ける

    ```
    aws ec2 associate-instance-event-window \
    --instance-event-window-id iew-0abcdef1234567890 \
    ```

```
--association-target "InstanceTags=[{Key=aws:ecs:fargateTask,Value=true}]"
```

キーと値のペアを複数使用して、イベントウィンドウを複数のサービスまたはクラスターに関連付けることができます。

Fargate は、各タスクのイベントウィンドウを次の順序で選択します。

- タスクのサービスに関連付けられたイベントウィンドウがある場合は、そのウィンドウが使用されます。これは、スタンドアロンタスクまたはアンマネージドタスクには適用されません。
- タスクのクラスターに関連付けられたイベントウィンドウがある場合は、そのウィンドウが使用されます。
- すべての Fargate タスクに設定されたイベントウィンドウがある場合は、そのウィンドウが使用されます。
- `fargateTaskRetirementWaitPeriod` 設定は、タスクに一致するイベントウィンドウがない場合に使用されます。

Fargate タスクのメンテナンス用イベントウィンドウの設定

異なる可用性要件を持つ複数の ECS サービスが Fargate で実行されている状況を考えてみましょう。こうした状況では、タスクの廃止を正確に制御する必要があります。次のように複数のイベントウィンドウを設定できます。

- すべての Fargate タスク向けのデフォルトメンテナンス: オフピーク時間 (毎日午前 12 時から午前 4 時) に定期的なメンテナンス用のイベントウィンドウを作成し、 `aws:ecs:fargateTask` タグを使用してすべての Fargate タスクに関連付けます。
- 開発クラスター向けの週末限定メンテナンス: 週末の中断に対応できるサービスが含まれる開発クラスターには、24 時間の週末ウィンドウ (土曜日と日曜日の終日) を作成し、`aws:ecs:clusterArn` タグとクラスター ARN を使用してクラスターに関連付けます。
- ミッションクリティカルなサービス向けの制限付きウィンドウ: 平日に高いアップタイムを維持する必要があるミッションクリティカルな決済処理サービスの場合は、メンテナンスを週末の早朝時間 (土曜日と日曜日の午前12 時から午前 4 時) に制限し、`aws:ecs:serviceArn` タグとサービス ARN を使用して特定のサービスに関連付けます。

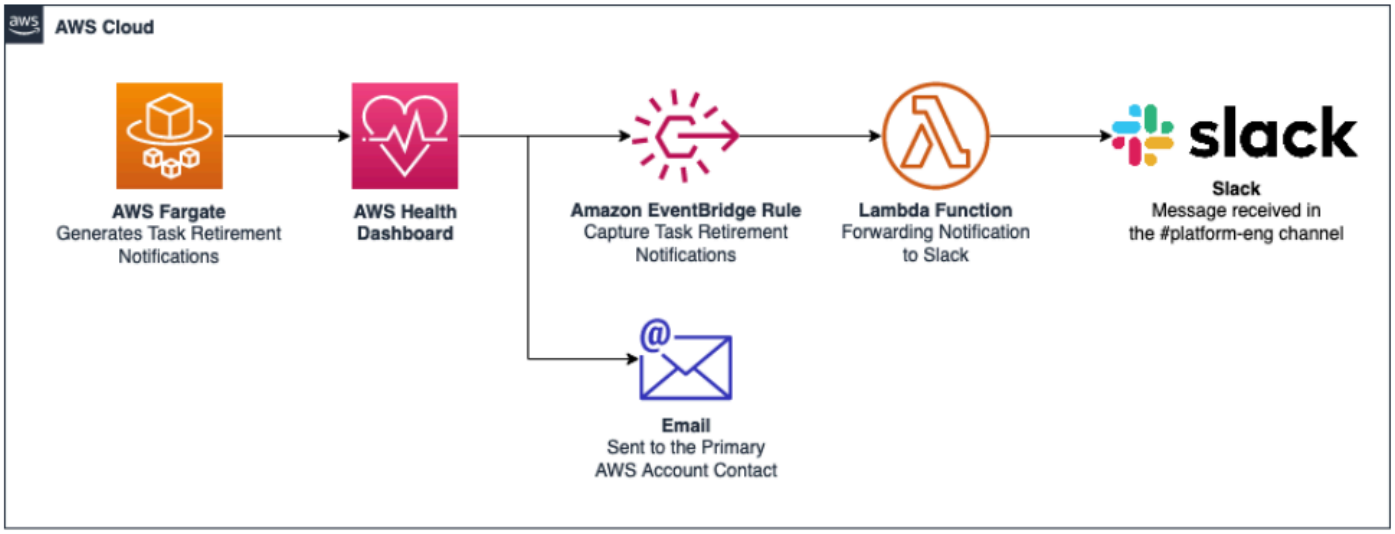
この設定では、決済サービスが特定の週末限定ウィンドウ、開発クラスターのサービスとタスクが週末の 24 時間ウィンドウ、他のすべての Fargate タスクがデフォルトの日次メンテナンスウィンドウを使用します。

詳細については、「Amazon Elastic Container Service API リファレンス」の「put-account-setting-default」と「put-account-setting」を参照してください。

ステップ 2: タスク廃止通知をキャプチャしてチームに警告し、措置を講じる

近日実施されるタスク廃止があるときは、AWS が AWS Health ダッシュボード、および AWS アカウント の主要 E メール連絡先にタスク廃止通知を送信します。AWS Health ダッシュボードは、Amazon EventBridge を含めた他の AWS サービスとの統合を多数提供しています。EventBridge を使用して、ChatOps ツールにメッセージを転送して次回の廃止の視認性を高めるなど、タスク廃止通知からオートメーションを構築することができます。AWS HealthAware は、AWS Health ダッシュボードの能力と、組織全体に通知を配布する方法を説明するリソースです。タスク廃止通知は、Slack などのチャットアプリケーションに転送できます。

以下の図は、このソリューションの概要です。



以下は、ソリューションの詳細情報です。

- Fargate がタスク廃止通知を AWS Health ダッシュボードに送信します。
- AWS Health ダッシュボードが AWS アカウント の主要 E メール連絡先に E メールを送信し、EventBridge に通知します。
- EventBridge には、廃止通知をキャプチャするルールがあります。

このルールは、イベント詳細タイプが "AWS Health Event" and the Event Detail Type Code: "AWS_ECS_TASK_PATCHING_RETIREMENT" のイベントを探します。

- ルールが、Slack Incoming Webhook を使用して情報を Slack に転送する Lambda 関数をトリガーします。詳細については、「Incoming Webhooks」を参照してください。

コード例については、Github で「Capturing AWS Fargate Task Retirement Notifications」を参照してください。

ステップ 3: タスクの置き換えを制御する

タスク廃止の正確なタイミングを制御することはできませんが、待機時間を定義することは可能です。独自のスケジュールに従ってタスクの置き換えを制御する場合は、タスク廃止通知をキャプチャして、タスク廃止の日付を把握することから始めます。その後、サービスを再度デプロイして代替タスクを起動することができます。スタンドアロンタスクも同様に置き換えます。ローリングデプロイを使用するサービスの場合は、廃止の開始時刻の前に、force-deployment オプションがある update-service を使用してサービスを更新します。

次の update-service 例では、force-deployment オプションを使用しています。

```
aws ecs update-service –-service service_name \
    --cluster cluster_name \
     --force-new-deployment
```

ブルー/グリーンデプロイを使用するサービスの場合は、AWS CodeDeploy で新しいデプロイメントを作成する必要があります。デプロイを作成する方法については、AWS Command Line Interfaceリファレンス の「create-deployment」を参照してください。

AWS Fargate で使用する Amazon ECS でサポートされているリージョン

次の表を参照すると、AWS Fargate 上の Linux コンテナおよび AWS Fargate 上の Windows コンテナのリージョンサポートを確認できます。

AWS Fargate 上の Linux コンテナ

AWS Fargate の Amazon ECS Linux コンテナは次の AWS リージョン でサポートされています。該当する場合、サポートされているアベイラビリティーゾーン ID が記載されています。

リージョン名	リージョン
米国東部 (オハイオ)	us-east-2
米国東部 (バージニア北部)	us-east-1
米国西部 (北カリフォルニア)	us-west-1 (usw1-az1 & usw1-az3 のみ)
米国西部 (オレゴン)	us-west-2
カナダ (中部)	ca-central-1
カナダ西部 (カルガリー)	ca-west-1
メキシコ (中部)	mx-central-1
アフリカ (ケープタウン)	af-south-1
アジアパシフィック (香港)	ap-east-1
アジアパシフィック (ムンバイ)	ap-south-1
アジアパシフィック (東京)	ap-northeast-1 (apne1-az1 、apne1-az2 、& apne1-az4 のみ)
アジアパシフィック (ソウル)	ap-northeast-2
アジアパシフィック (大阪)	ap-northeast-3
アジアパシフィック (ハイデラバード)	ap-south-2
アジアパシフィック (シンガポール)	ap-southeast-1
アジアパシフィック (シドニー)	ap-southeast-2
アジアパシフィック (タイ)	ap-southeast-7
アジアパシフィック (ジャカルタ)	ap-southeast-3
アジアパシフィック (メルボルン)	ap-southeast-4
アジアパシフィック (マレーシア)	ap-southeast-5

リージョン名	リージョン
カナダ (中部)	ca-central-1
カナダ西部 (カルガリー)	ca-west-1
中国 (北京)	cn-north-1 (`cnn1-az1` & `cnn1-az2` のみ)
中国 (寧夏)	cn-northwest-1
欧州 (フランクフルト)	eu-central-1
欧州 (チューリッヒ)	eu-central-2
欧州 (アイルランド)	eu-west-1
欧州 (ロンドン)	eu-west-2
欧州 (パリ)	eu-west-3
欧州 (ミラノ)	eu-south-1
欧州 (スペイン)	eu-south-2
欧州 (ストックホルム)	eu-north-1
南米 (サンパウロ)	sa-east-1
イスラエル (テルアビブ)	il-central-1
中東 (バーレーン)	me-south-1
中東 (UAE)	me-central-1
AWS GovCloud (米国東部)	us-gov-east-1
AWS GovCloud (米国西部)	us-gov-west-1

AWS Fargate 上の Windows コンテナ

AWS Fargate の Amazon ECS Windows コンテナは次の AWS リージョン でサポートされています。該当する場合、サポートされているアベイラビリティーゾーン ID が記載されています。

リージョン名	リージョン
米国東部 (オハイオ)	us-east-2
米国東部 (バージニア北部)	us-east-1 (use1-az1、use1-az2、use1-az4、use1-az5、& use1-az6 のみ)
米国西部 (北カリフォルニア)	us-west-1 (usw1-az1 & usw1-az3 のみ)
米国西部 (オレゴン)	us-west-2
アフリカ (ケープタウン)	af-south-1
アジアパシフィック (香港)	ap-east-1
アジアパシフィック (ムンバイ)	ap-south-1
アジアパシフィック (ハイデラバード)	ap-south-2
アジアパシフィック (大阪)	ap-northeast-3
アジアパシフィック (ソウル)	ap-northeast-2
アジアパシフィック (シンガポール)	ap-southeast-1
アジアパシフィック (シドニー)	ap-southeast-2
アジアパシフィック (メルボルン)	ap-southeast-4
アジアパシフィック (マレーシア)	ap-southeast-5
アジアパシフィック (東京)	ap-northeast-1 (apne1-az1 、apne1-az2 、& apne1-az4 のみ)
カナダ (中部)	ca-central-1 (cac1-az1 & cac1-az2 のみ)

リージョン名	リージョン
カナダ西部 (カルガリー)	ca-west-1
中国 (北京)	cn-north-1 (cnn1-az1 & cnn1-az2 のみ)
中国 (寧夏)	cn-northwest-1
欧州 (フランクフルト)	eu-central-1
欧州 (チューリッヒ)	eu-central-2
欧州 (アイルランド)	eu-west-1
欧州 (ロンドン)	eu-west-2
欧州 (パリ)	eu-west-3
欧州 (ミラノ)	eu-south-1
欧州 (スペイン)	eu-south-2
欧州 (ストックホルム)	eu-north-1
南米 (サンパウロ)	sa-east-1
イスラエル (テルアビブ)	il-central-1
中東 (アラブ首長国連邦)	me-central-1
中東 (バーレーン)	me-south-1

Amazon ECS 用の EC2 キャパシティのアーキテクト

料金を最適化する必要がある大規模なワークロードには EC2 キャパシティを使用します。

EC2 を使用してタスク定義とサービスをモデル化する方法を検討する際には、一緒に実行する必要があるプロセスはどれか、また、各コンポーネントをどのようにスケールするかについて検討することをお勧めします。

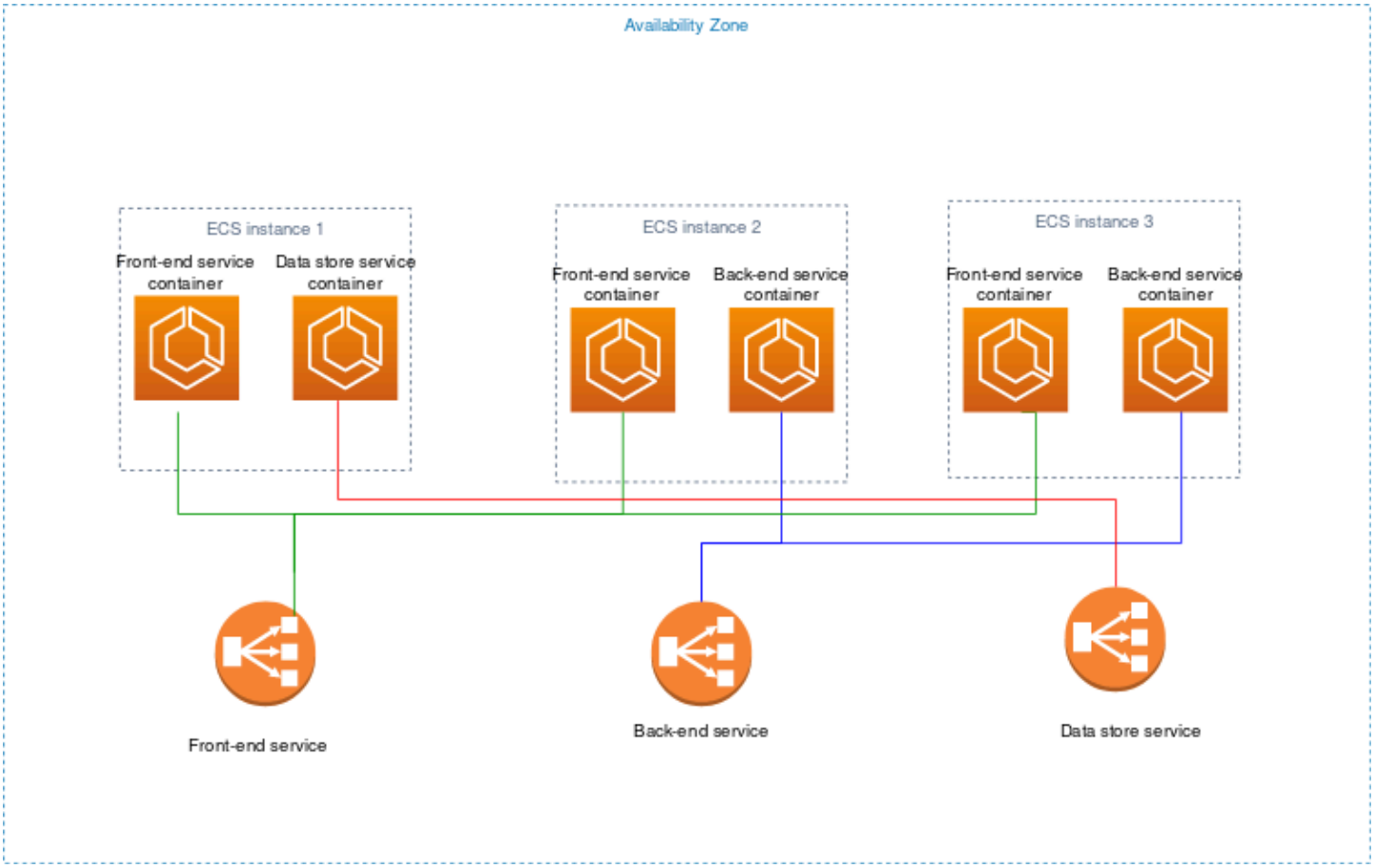
例えば、以下のコンポーネントで構成されるアプリケーションがあるとします。

- ウェブページに情報を表示するフロントエンドサービス
- フロントエンドサービスに API を提供するバックエンドサービス
- データストア

この例では、共通の目的で使用されるコンテナをまとめてグループ化するタスク定義を作成します。異なるコンポーネントは、複数のタスク定義に分割します。以下の例のクラスターでは、3 つのコンテナインスタンスにより、3 つのフロントエンドサービスコンテナ、2 つのバックエンドサービスコンテナ、さらに 1 つのデータストアサービスコンテナを実行しています。

一緒に実行する必要のあるリンクされたコンテナなど、関連するコンテナをタスク定義にグループ化できます。例えば、フロントエンドサービスに追加するログストリーミングコンテナは、同じタスク定義に含めることができます。

タスク定義を作成したら、それらの定義からサービスを作成して、使用可能なタスクの必要数を維持できます。詳細については、「Amazon ECS のローリング更新デプロイの作成」を参照してください。サービスでは、コンテナを Elastic Load Balancing ロードバランサーに関連付けることができます。詳細については、「ロードバランサーを使用して Amazon ECS サービストラフィックを分散する」を参照してください。アプリケーションの要件が変更された際には、サービスを更新してタスクの必要数を増減できます。あるいは、サービスを更新して、タスクに新しいバージョンのコンテナをデプロイすることも可能です。詳細については、「Amazon ECS サービスを更新する」を参照してください。



EC2 と Amazon ECS の外部インスタンスでのコンテナイメージのプル動作

コンテナーの起動にかかる時間は、基礎となるコンテナイメージによって異なります。たとえば、太いイメージ (Debian、Ubuntu、Amazon 1/2 のフルバージョン) は、それぞれのスリムバージョン (Debian-Slim、Ubuntu-Slim、Amazon-Slim) や小さいベースイメージ (Alpine) と比較して、コンテナ内で実行されるサービスの数が多いため、起動に時間がかかる場合があります。

Amazon ECS エージェントはタスクを開始すると、リモートレジストリから Docker イメージを取得し、ローカルコピーをキャッシュします。アプリケーションのリリースごとに新しいイメージタグを使用する場合、この動作は不要です。

`ECS_IMAGE_PULL_BEHAVIOR` エージェントパラメータはイメージのプル動作を決定します。以下のオプションが利用できます。

- `ECS_IMAGE_PULL_BEHAVIOR`: `default`

イメージはリモートでプルされます。プルが失敗した場合、インスタンスにキャッシュされたイメージが使用されます。

- ECS_IMAGE_PULL_BEHAVIOR: always

  イメージはリモートでプルされます。プルに失敗した場合、そのタスクは失敗します。

デプロイをスピードアップするには、Amazon ECS エージェントパラメータを次のいずれかの値に設定します。

- ECS_IMAGE_PULL_BEHAVIOR: once

  同じコンテナインスタンスの以前のタスクによりイメージがプルされていないか、自動クリーンアッププロセスによってキャッシュされたイメージが削除された場合にのみ、イメージがリモートでプルされます。それ以外の場合は、インスタンスにキャッシュされたイメージが使用されます。これにより、不要なイメージのプルがなくなります。

- ECS_IMAGE_PULL_BEHAVIOR: prefer-cached

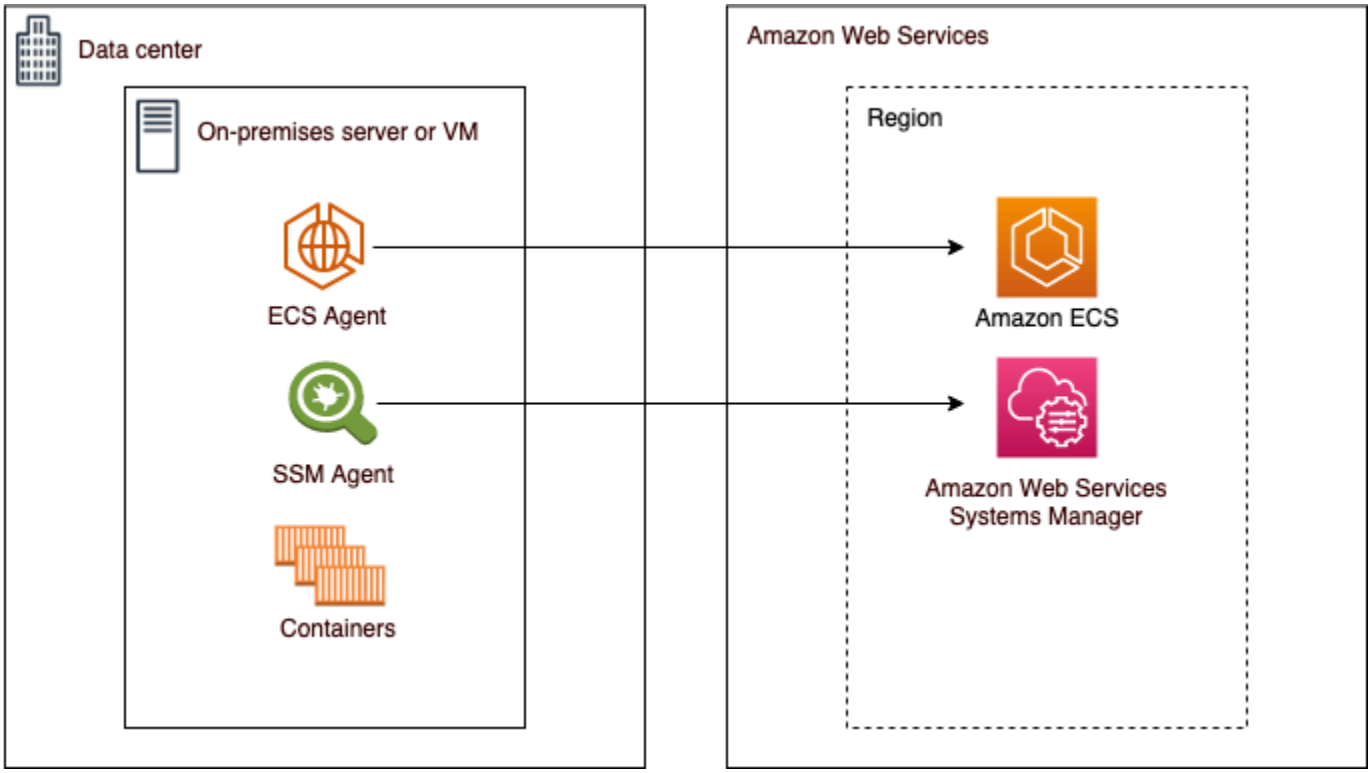
  キャッシュされたイメージがない場合に、リモートでイメージがプルされます。それ以外の場合は、インスタンスにキャッシュされたイメージが使用されます。キャッシュされたイメージが削除されないように、そのコンテナの自動イメージクリーンアップはオフになっています。

ECS_IMAGE_PULL_BEHAVIOR パラメータを前述の値のいずれかに設定すると、Amazon ECS エージェントはダウンロードされた既存のイメージを使用するため、時間を節約できます。Docker イメージのサイズが大きい場合、ダウンロード時間がネットワーク経由で取得されるまでに 10 ~ 20 秒かかることがあります。

Amazon ECS の外部インスタンス (Amazon ECS Anywhere)

Amazon ECS Anywhere は、オンプレミスサーバーや仮想マシン (VM) などの外部インスタンスを Amazon ECS クラスターに登録するためのサポートを提供します。外部インスタンスは、アウトバウンドトラフィックを生成したり、データを処理したりするアプリケーションを実行するために最適化されています。アプリケーションがインバウンドトラフィックを必要とする場合、Elastic Load Balancing のサポートがないため、これらのワークロードの実行効率が低下します。Amazon ECS は、新しいEXTERNAL起動タイプで、サービスを作成したり、外部インスタンスでタスクを実行したりできます。

以下に、Amazon ECS Anywhere の高レベルのシステムアーキテクチャの概要を示します。オンプレミスサーバーには、Amazon ECS エージェントと SSM エージェントの両方がインストールされています。



詳細については、「外部インスタンスの Amazon ECS クラスター」を参照してください。

Amazon ECS Express Mode

Amazon ECS Express Mode サービスは、適切なデフォルトを提供し、補助的な AWS サービスの設定を自動化することで、コンテナ化されたアプリケーションのデプロイの複雑さを軽減します。Express Mode サービスでは、複数のサービスの設定パラメータを管理する必要がなく、開始に必要となるものは 3 つだけです。

- コンテナイメージ
- タスク実行ロール
- インフラストラクチャロール

Amazon ECS Express Mode は、アクセス可能な一意の URL を持つ Fargate ベースの ECS サービス、SSL/TLS を使用するロードバランサー、自動スケーリングポリシー、モニタリング、ネットワークコンポーネントなど、必要なすべてのインフラストラクチャをオーケストレーションして設定します。

以下のシナリオで、Amazon ECS Express Mode サービスを使用します。

- ウェブアプリケーションと API – HTTP リクエストを処理するステートレスコンテナ化アプリケーション
- 迅速なプロトタイピング – インフラストラクチャのセットアップオーバーヘッドなしでアプリケーションを迅速にデプロイしてテストする
- 開発者の生産性 – 深い AWS の知識がなくても、アプリケーションチームが独立してデプロイできるようにする
- プラットフォームチームの効率 – セルフサービスのデプロイ機能を提供することで、メンテナンスのオーバーヘッドを削減する

Amazon ECS Express Mode サービスは、パブリックまたはプライベートの HTTPS アプリケーションをサポートし、使用率またはトラフィックに基づいて自動的にスケールします。

Amazon ECS Express Mode サービスの利点

Amazon ECS Express Mode サービスには、AWS でコンテナ化されたアプリケーションをデプロイして管理する際に、主な利点がいくつかあります。

- 簡素化されたデプロイ – 本稼働に対応するデフォルトを使用して複数の AWS サービスでコンテナ化されたアプリケーションをデプロイできます。
- 機能に妥協なし – きめ細かな制御や高度な機能が必要な場合でも、基盤となるすべての AWS リソースにアクセスして直接管理できます。
- コストの最適化 – 同じネットワーク設定を使用して複数の Express Mode サービス間で Application Load Balancer を共有し、コストを削減します。
- 透過的なインフラストラクチャ – すべてのリソースは AWS アカウントに作成され、AWS コンソールと API を通じて完全に可視化され、アクセスできます。

料金

Amazon ECS Express Mode サービスの使用に追加料金はかかりません。以下を含む、アプリケーションを実行するために作成される基盤の AWS リソースに対してのみ料金が発生します。

- Fargate コンピューティングリソース
- Application Load Balancer
- CloudWatch Logs と Metrics
- データ転送料金

可用性

Amazon ECS Express Mode は、Amazon ECS と Fargate がサポートされているすべての AWS リージョンで使用できます。Express Mode サービスは、次のいずれかを使用して作成、管理できます。

- Amazon ECS コンソール
- AWS AWS CLI
- AWS SDK
- CloudFormation
- Terraform
- AWSAmazon ECS 用の ラボ MCP サーバー

Amazon ECS Express Mode サービスの作成

Amazon ECS Express Mode サービスは、適切なデフォルトを提供し、補助的な AWS サービスの設定を自動化することで、コンテナ化されたアプリケーションのデプロイの複雑さを軽減します。Express Mode サービスでは、複数のサービスの設定パラメータを管理する代わりに、開始に必要となるのはコンテナイメージ、タスク実行ロール、インフラストラクチャロールだけです。

デプロイが完了すると、アプリケーションの一意の URL が返されます。クラスターにデプロイしたサービスがあります。クラスターを指定しなかった場合は `Default` クラスター内にあります。

前提条件

Express Mode サービスを作成する前に、次のいずれかがあることを確認してください。

- Amazon ECR に保存されたコンテナイメージ。
- または、プライベートレジストリに保存されたコンテナイメージ。プライベートレジストリを使用するには、Secrets Manager シークレットを設定します。詳細については、AWS Secrets Manager ユーザーガイドの「Create an AWS Secrets Manager secret」を参照してください。

> **Note**
>
> IAM ロールも必要ですが、これについてはコンソールと CLI ガイドで別途説明されています。Amazon ECS コンソールには、ロール作成のための自動フローがあります。

チュートリアル

- コンソールを使用したシンプルな初回実行エクスペリエンスについては、「the section called “コンソールで最初の Express Mode サービスを作成する”」を参照してください。
- AWS CLI を使用して Express Mode サービスを作成するには、「the section called “AWS CLI を使用して最初の Express Mode サービスを作成する”」を参照してください。
- Express Mode が作成する内容とその仕組みの詳細については、「the section called “Express Mode サービスによって作成されたリソース ”」を参照してください。

カスタマイズ

Express Mode サービスを作成する際、Amazon ECS は Amazon ECS およびその他の AWS リソースのデフォルトを選択します。必要に応じて、オプションのパラメータを使用してアプリケーションをカスタマイズします。Express Mode サービスを活用して、以下をカスタマイズします。

- サービス名
- Amazon ECS クラスター
- コンテナ設定 – ポートとヘルスチェックパス
- 環境変数とシークレット
- カスタムコマンド
- タスクロール – このロールにより、(コンテナ上の) アプリケーションコードが他の AWS サービスを使用できるようになります。

  アプリケーションが Amazon S3 などの他の AWS サービスにアクセスする場合は、このロールが必要です。
- コンピューティングリソース – CPU とメモリの割り当て
- ネットワーク設定 – サブネットとサービスセキュリティグループ
- Application Auto Scaling ポリシー – メトリクス、ターゲット値、タスク制限
- CloudWatch Logs の設定
- リソースタグ
- タスク定義 – 独自のカスタムタスク定義を使用する

コンソールで最初の Amazon ECS Express Mode サービスを作成する

Express Mode サービスのコンソールエクスペリエンスでは、コンテナ化されたアプリケーションを最小限の設定で効率的にデプロイできます。前提条件の詳細については、「Amazon ECS Express Mode」を参照してください。Express Mode が作成する内容とその仕組みの詳細については、「the section called “Express Mode サービスによって作成されたリソース ”」を参照してください。

手順

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。

2. ナビゲーションペインで、[Express Mode] を選択します。
3. [アプリの設定] で以下を実行します。

   a. アプリケーションに使用するイメージを指定します。[イメージ URI] でイメージの URI を入力します。Amazon ECR イメージを参照するには、[ECR イメージを参照] を選択し、次の操作を行います。

      i. [プライベートリポジトリ] で、Amazon ECR プライベートリポジトリを選択します。

      ii. [イメージ] で、イメージを選択します。

      iii. イメージを識別する方法を選択します。[イメージの選択方法] で、以下のオプションのいずれかを選択します。

         - AWS は [イメージダイジェスト] を選択することをお勧めしています。
         - タグを使用するには、[イメージタグ] を選択してからタグを選択します。

   b. プライベートレジストリを使用する場合は、[プライベートレジストリ] を選択します。次に、[Secrets Manager ARN または名前] に前提条件で作成した Secrets Manager ARN を入力します。

   c. [タスク実行ロール] と [インフラストラクチャロール] で、ロールを選択します。Amazon ECS でこれらのロールを作成したことがない場合は、ドロップダウンに [新しいロールを作成する] オプションが表示されます。このオプションを選択した場合、[作成] をクリックするとロールが自動的に作成されます。タスク実行ロールの場合、アクセス許可の詳細には、「AWS マネージドポリシーリファレンスガイド」の AmazonECSTaskExecutionRolePolicy に関する追加の詳細へのリンクが表示されます。インフラストラクチャロールの場合、アクセス許可の詳細には、「AWS マネージドポリシーリファレンスガイド」の AmazonECSInfrastructureRoleforExpressGatewayServices に関する追加の詳細へのリンクが表示されます。

      これらのロールをカスタマイズする場合は、[新しいロールを作成する] ボタンから IAM コンソールに移動し、ロールにアタッチされるアクセス許可を作成、表示、編集できます。Amazon ECS Express Mode コンソールに戻ったら、更新して作成したロールを表示します。

      > **Note**
      >
      > Amazon ECS コンソールで初めてサービスを作成するときに、一部のユーザーに `Invalid Parameter Exception: Unable to assume the service`

`linked role.` が表示されることがあります。これが発生した場合は、数秒待ってからもう一度試してください。

4. [作成] を選択します。最初の Amazon ECS Express Mode サービスを作成しました。

次のステップ

最初の Express Mode サービスを作成した後:

- デプロイが完了したら、提供されたアプリケーション URL を使用してアプリケーションにアクセスします。
- コンソールの [リソース] タブでデプロイの進行状況をモニタリングします。デプロイが完了すると、サービスはトラフィックを受信する準備が整います。詳細については、「the section called “Express Mode サービスの詳細の表示”」を参照してください。
- サービスに戻るには、ナビゲーションの [クラスター] を選択し、作成時にクラスターを指定しなかった場合は `default` クラスターを選択します。Amazon ECS はクラスターと呼ばれる論理グループにサービスを整理します。これによりアカウントでアプリケーションを簡単に整理できます。
- デプロイ中に問題が発生した場合は、the section called “トラブルシューティング” を参照してください。

カスタマイズ

必要に応じて、Express Mode サービスの設定をカスタマイズすることもできます。[追加設定 – オプション] というタイトルのセクションを開きます。

1. サービスをデプロイしたい特定の Amazon ECS クラスターがある場合は、ドロップダウンから [クラスター] を選択します。クラスターが指定されていない場合、Express Mode は `Default` クラスターを使用します。Express Mode コンソールを離れてクラスターを作成する場合は、更新ボタンを使用してドロップダウンを入力します。
2. サービスの名前を入力します。名前が指定されていない場合、Express Mode はコンテナイメージ名から名前を生成します。このサービス名は、アプリケーション URL、Amazon ECS Service、Amazon ECS タスク定義、証明書、スケーリングターゲット、スケーリングポリシーなど、複数のリソースで使用されます。
3. 必要に応じて、コンテナの詳細を指定します。

a. [コンテナポート] に、アプリケーションがリッスンするポートを入力します (デフォルトは 80)。

b. [ヘルスチェックパス] には、ヘルスチェックのパス ("/health" など) を入力します。デフォルトは "/". です。

c. [環境変数] で、アプリケーションに必要な環境変数のキーと値のペアを追加します。[キー] に、環境変数の名前を入力します。[値タイプ] で、[環境変数] または [シークレット] を選択します。[値または値の取得] には、値またはリファレンスを入力します。[環境変数の追加] を選択して、必要に応じて変数を追加します。デフォルトは提供されません。

d. [コマンド] に、オプションでカスタムコマンドを入力して Docker CMD命令を上書きします。デフォルトは提供されません。

e. [タスクロール] で、アプリケーションにアクセス許可を付与する IAM ロールを選択します。デフォルトは提供されません。

4. 必要に応じて、[コンピューティング] と [自動スケーリング] の詳細を指定します。

a. [CPU] で、タスクの vCPU 割り当てを選択します (デフォルトは 1 vCPU)。

b. [メモリ] で、タスクのメモリ割り当てを選択します (デフォルトは 2 GB)。

c. [ECS サービスメトリクス] では、スケールするためのメトリクスを選択します (デフォルトは [平均 CPU 使用率])。

d. [ターゲット値] で、スケーリングのターゲットパーセンテージを選択します (デフォルトは 60 です)。

e. [最小タスク数] と [最大タスク数] で、スケーリング制限を設定します。(デフォルトは 1 と 20)

5. ネットワーク設定をカスタマイズするには、このチェックボックスをオンにします。これらの設定をカスタマイズしない場合、Express Mode はデフォルトの VPC を使用します。

a. [VPC] を選択すると、サービスが実行されるサブネットにすばやく移動できます。必要に応じて、Amazon ECS コンソールを離れて新しい VPC を作成し、コンソールに戻って更新して、ドロップダウンでその VPC を確認します。

b. [セキュリティグループ] で、セキュリティグループを選択または作成して、サービスへの追加のインバウンドネットワークアクセスを許可します。

6. [ログ] で以下を実行します。

a. [Amazon CloudWatch ロググループ] で、アプリケーションログの希望のロググループ名を入力します。デフォルトのロググループは、クラスター名とサービス名に従って名前が付けられます。

b. [Amazon CloudWatch ログストリームプレフィックス] で、ログストリームの希望のプレフィックスを入力します。デフォルトのストリームプレフィックスは `ecs/Main/.` です。

7. [タグ] で、キーと値のペアを追加してリソースにタグを付けます。[Key] (キー) に、タグキーを入力します。[Value] (値) に、タグの値を入力します。[新しい項目を追加] を選択して、必要に応じてタグを追加します。タグは作成時にのみ追加できます。

AWS CLI を使用して最初の Express Mode サービスを作成する

このチュートリアルでは、AWS AWS CLI を使用して Express Mode サービスアプリケーションを作成する方法を紹介します。Express Mode サービスのコア機能を示すシンプルなウェブアプリケーションをデプロイします。

前提条件

開始する前に、以下があることを確認してください。

- 適切なアクセス許可が付与されている AWS アカウント
- AWS AWS CLI がインストールされ、設定されている
- Amazon ECR またはプライベートレジストリに保存されたコンテナイメージ
- サービスを作成する AWS リージョンにパブリックサブネットがあるデフォルトの VPC。デフォルトでは、Express Mode サービスは、アカウントのデフォルト VPC とパブリックサブネットにインターネット向けロードバランサーを作成します。アカウントにデフォルト VPC がない場合は、開始する前に作成するか、サービス作成時に独自のサブネットを指定します。詳細については、Amazon VPC ユーザーガイド の「デフォルト VPC」を参照してください。

ステップ 1: IAM ロールを作成する

Express Mode サービスには 2 つの IAM ロールが必要です。タスク実行ロールを使用すると、Amazon ECS はユーザーに代わってコンテナイメージをプルし、ログを書き込むことができます。次のポリシーを使用して、タスク実行ロールとインフラストラクチャロールを作成します。

```
#Create the roles with ECS trust policies
aws iam create-role --role-name ecsTaskExecutionRole \
```

```
    --assume-role-policy-document '{
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Principal": {
                    "Service": "ecs-tasks.amazonaws.com"
                },
                "Action": "sts:AssumeRole"
            }
        ]
    }'

aws iam create-role --role-name ecsInfrastructureRoleForExpressServices \
    --assume-role-policy-document '{
        "Version": "2012-10-17",
        "Statement": [
            {
                "Sid": "AllowAccessInfrastructureForECSExpressServices",
                "Effect": "Allow",
                "Principal": {
                    "Service": "ecs.amazonaws.com"
                },
                "Action": "sts:AssumeRole"
            }
        ]
    }'
```

```
#Attach the AWS managed policies
aws iam attach-role-policy --role-name ecsTaskExecutionRole \
    --policy-arn arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy
aws iam attach-role-policy --role-name ecsInfrastructureRoleForExpressServices \
    --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonECSInfrastructureRoleforExpressGatewayServices
```

詳細については、「the section called “タスク実行 IAM ロール”」を参照してください。

> **Note**
>
> IAM ロールは結果整合性があるため、新しく作成されたロールはすぐには使用できない場合があります。これらのロールの作成直後に最初の `create-express-gateway-service` 呼び出しが `Unable to assume the service linked role` で失敗した場合 (または同

様の assume-role エラー) 、ロールが伝播されるまでしばらく (約 1 分) 待ってからコマンドを再試行します。

ステップ 2: 最初の Express Mode サービスアプリケーションを作成する

最低限の必要なパラメータを使用して Express Mode サービスアプリケーションを作成します。

```
aws ecs create-express-gateway-service \
    --primary-container "image"="public.ecr.aws/nginx/nginx:latest" \
    --execution-role-arn arn:aws:iam::123456789012:role/ecsTaskExecutionRole \
    --infrastructure-role-arn arn:aws:iam::123456789012:role/
ecsInfrastructureRoleForExpressServices \
    --monitor-resources
```

このコマンドは、以下を備えた Express Mode サービスアプリケーションを作成します。

- イメージ名から生成された一意のサービス名
- 1 vCPU と 2 GB メモリ
- HTTPS トラフィック用のポート 80
- CPU 使用率に基づく自動スケーリング
- デフォルトの VPC およびパブリックサブネットのインターネット向け Application Load Balancer
- このサービスに一意の URL (`servicename.ecs.region.on.aws` 形式)

このコマンドは、プロビジョニング中のリソースを継続的にモニタリングし、ステータスを返します。サービスのデプロイが完了すると、アプリケーション URL はトラフィックを受信する準備が整います。モニタリングを中断すると、サービス ARN や URL など、作成されたサービスに関する情報が返されます。

```
{
    "service": {
        "cluster": "arn:aws:ecs:region:123456789012:cluster/default",
        "serviceName": "nginx-1234",
        "serviceArn": "arn:aws:ecs:region:123456789012:service/default/nginx-1234",
        "infrastructureRoleArn": "arn:aws:iam::123456789012:role/
ecsInfrastructureRoleForExpressServices",
        "status": {
            "statusCode": "ACTIVE"
        },
```

```
        "createdAt": "UNIXTIMESTAMP"
    }
}
```

ステップ 3: カスタム設定で Express Mode サービスアプリケーションを作成する

追加のパラメータを指定することで、Express Mode サービスアプリケーションをカスタマイズできます。

```
aws ecs create-express-gateway-service \
    --execution-role-arn arn:aws:iam::123456789012:role/ecsTaskExecutionRole \
    --infrastructure-role-arn arn:aws:iam::123456789012:role/
ecsInfrastructureRoleForExpressServices \
    --primary-container '{"image":"123456789012.dkr.ecr.region.amazonaws.com/my-
app:latest","containerPort":8080,"environment":[{"name":"ENV","value":"production"},
{"name":"DEBUG","value":"false"}]}' \
    --service-name "my-web-app" \
    --cpu 2 \
    --memory 4 \
    --health-check-path "/health" \
    --scaling-target '{"minTaskCount":3,"maxTaskCount":100}' \
    --monitor-resources
```

これにより、以下を備えたアプリケーションが作成されます。

- カスタム名「my-web-app」
- 2 vCPU と 4 GB メモリ
- アプリケーショントラフィック用のポート 8080
- カスタムのヘルスチェックエンドポイント
- 環境変数
- 最小 3 タスク、最大 100 タスクの自動スケーリング

独自のタスク定義を使用して Express Mode サービスを作成

独自のタスク定義を使用して Express Mode サービスを作成することもできます。独自のタスク定義を使用すれば、タスクレベルの設定を完全に制御できます。Express Mode は、ロードバランシング、自動スケーリング、カナリアデプロイなどのインフラストラクチャを引き続き管理します。

次の AWS Command Line Interface (AWS AWS CLI) コマンドは、カスタムタスク定義を使用して Express Mode サービスを作成します。

```
aws ecs create-express-gateway-service \
    --infrastructure-role-arn arn:aws:iam::123456789012:role/
ecsInfrastructureRoleForExpressServices \
    --task-definition-arn arn:aws:ecs:us-west-2:123456789012:task-definition/my-td:3 \
    --health-check-path "/health" \
    --monitor-resources
```

タスク定義は、以下の要件を満たしている必要があります。

- コンテナ名 `Main`
- `Main` コンテナには、コンテナポートとポート名を含む単一の TCP ポートマッピングが必要です
- タスク定義の互換性には `FARGATE` が必要です

> **Note**
>
> `taskDefinitionArn` パラメータは、同じ API コールの `primaryContainer`、`executionRoleArn`、`taskRoleArn`、`cpu`、または `memory` で指定することはできません。Express Mode は、指定されたタスク定義からこれらの値を取得します。

ステップ 4: デプロイをモニタリングする

`--monitor-resources` フラグは、Express Mode サービスへの Create、Update、Delete 呼び出しで機能します。ただし、変更アクションとは別に、サービスのリソースをいつでもモニタリングできます。デプロイの時間は、プロビジョニングする必要があるリソースによって異なります。ステータスが `ACTIVE` に変わると、アプリケーションはトラフィックを受信する準備が整います。

```
aws ecs monitor-express-gateway-service --service-arn
 arn:aws:ecs:region:123456789012:service/default/app-23d97h88
```

Express Mode サービスアプリケーションの現在の設定とステータスを確認することもできます。

```
aws ecs describe-express-gateway-service --service-arn
 arn:aws:ecs:region:123456789012:service/default/app-23d97h88
```

ステップ 5: アプリケーションにアクセスする

Express Mode サービスアプリケーションがアクティブになったら、提供された URL を使用してアクセスします。URL 形式は次のとおりです。

```
https://<service-name>.ecs.<region>.on.aws/
```

例えば、次のようになります。

```
https://app-23d97h88.ecs.us-west-2.on.aws/
```

以下を備えたアプリケーションが実行されます。

- SSL/TLS の自動終了
- 複数のアベイラビリティーゾーンでのロードバランシング
- CPU 使用率に基づく自動スケーリング
- CloudWatch のログ記録とモニタリング
- 今後の更新のための 5XX ロールバックアラームとカナリアデプロイ

Amazon ECS Express Mode サービスによって作成されたリソース

Express Mode サービスは、適切なデフォルトを提供し、補助的な AWS サービスの設定を自動化することで、コンテナ化されたアプリケーションのデプロイの複雑さを軽減します。Express Mode サービスでは、複数のサービスの何百もの設定パラメータを管理する代わりに、開始に必要となるのはコンテナイメージ、タスク実行ロール、インフラストラクチャロールだけです。

Express Mode サービスでは、本番稼働用ウェブアプリケーションを実行するための優先インフラストラクチャをバンドルする Amazon ECS Service のプロビジョニングと設定の手順が簡素化されています。Express Mode サービスには、運用と設定のベストプラクティスが自動的に組み込まれています。以下のリソースが作成されます。

- Fargate キャパシティプロバイダーを備えた ECS デフォルトクラスター (存在しない場合)
- コンテナ、ログ記録、ネットワーク設定を含むタスク定義
- カナリアデプロイと自動スケーリングを備えたサービス
- HTTPS リスナー、リスナールール、ターゲットグループを含む Application Load Balancer
- 最小限の必要なイングレスを持つセキュリティグループ

- 自動スケーリングとロードバランシングのための、サービスにリンクされたロール
- Application Auto Scaling のスケーラブルターゲットとターゲット追跡スケーリングポリシー。

  スケーリングポリシーは、指定されたターゲット値、またはそれに近い値にメトリクスを維持するため、必要に応じてサービスタスクを追加または削除します。ターゲットの追跡スケーリングポリシーは、メトリクスをターゲット値近くに維持することに加えて、負荷パターンの変動によるメトリクスの変動に合わせて調整し、サービスで実行されているタスク数の急速な変動を最小化します。
- サービスに固有のロググループ
- 障害のあるデプロイを検出するためのメトリクスアラーム
- ACM 証明書

デフォルトを使用して Express Mode サービスを開始するか、最小限のパラメータセットを使用してアプリケーションを設定します。

クラスターのデフォルト

以下のクラスター用のデフォルトは、Express Mode 内から設定できます。

- clusterName: `default` クラスターを使用します。

  このオプションは、コンソールの追加設定オプションを使用するか、`create-express-gateway-service --cluster` を指定することでカスタマイズできます。

以下のデフォルトは Express Mode によって設定されますが、クラスターで直接更新できます。

- capacityProviders: ["FARGATE"] – Fargate を使用します

タスク定義のデフォルト

以下のタスク定義用のデフォルトは、Express Mode 内から設定できます。

- cpu: 1024 – タスクに割り当てられた 1 vCPU 単位

  このオプションは、コンソールの追加設定オプションを使用するか、`create-express-gateway-service --cpu` を実行することでカスタマイズできます。
- メモリ: 2048 – タスクに割り当てられた 2 GB のメモリ

このオプションは、コンソールの追加設定オプションを使用するか、`create-express-gateway-service --memory` を実行することでカスタマイズできます。

以下のデフォルトは Express Mode によって設定されますが、タスク定義で直接更新できます。

- networkMode: "awsvpc" – `awsvpc` ネットワークモードを使用します
- operatingSystemFamily: "LINUX" – Linux オペレーティングシステムで実行されます
- cpuArchitecture: "X86_64" – x86_64 プロセッサアーキテクチャを使用します
- requiresCompatibilities: ["FARGATE"] – Fargate 起動タイプと互換性があります
- platformVersion: "LATEST" – 最新の Fargate プラットフォームバージョンを使用します

`taskDefinitionArn` パラメータを使用してカスタムタスク定義を指定することもできます。Express Mode では、CPU、メモリ、コンテナ定義、および定義した追加のコンテナや設定など、タスク定義をそのまま使用します。

コンテナ定義のデフォルト

以下のコンテナ用のデフォルトは、Express Mode 内から設定できます。

- port: 80 – コンテナトラフィックのデフォルトポート

以下のデフォルトは Express Mode によって設定されますが、タスク定義 – コンテナ定義で直接更新できます。

- essential: true – コンテナはタスクの正常性に必須としてマークされます
- protocol: "tcp" – ロードバランサーの通信のデフォルトプロトコル
- name: "Main" – プライマリコンテナのデフォルト名

  デフォルトコンテナの名前を変更すると、Express Mode がサービスに対して後続の更新を実行できなくなる可能性があります。これは、Express Mode コンソールまたは API を引き続き使用する場合はお勧めしません。
- versionConsistency: enabled – 一貫したデプロイのためにイメージタグをダイジェスト ID に解決します
- initProcessEnabled: true – Docker init プロセスのサポートを有効にします。
- stopTimeout: 30 秒 – SIGTERM シグナルと SIGKILL シグナルの間の時間。

ログ記録のデフォルト

以下の CloudWatch Logs 用のデフォルトは、Express Mode 内から設定できます。

- awslogs-group: "/aws/ecs/<cluster>/<name>-####" – クラスターとサービス名に由来するロググループ名

  このオプションは、コンソールの追加設定オプションを使用するか、`create-express-gateway-service --primary-container “awsLogsConfiguration”=[{“logGroup”=“/my/logGroup/”}]` を実行することでカスタマイズできます。
- awslogs-stream-prefix: "ecs" – ログストリーム名のプレフィックス

  このオプションは、コンソールの追加設定オプションを使用するか、`create-express-gateway-service --primary-container “awsLogsConfiguration”=[{“logStreamPrefix”=“myprefix”}]` を実行することでカスタマイズできます。

以下のデフォルトは Express Mode によって設定されますが、タスク定義 – ログ設定で直接更新できます。

- logDriver: "awslogs" – CloudWatch Logs を使用します
- awslogs-region: 現在のリージョン – Express Mode サービスと同じリージョンを使用します
- mode: "non-blocking" – ログ記録モードがノンブロッキングに設定されます
- max-buffer-size: "25m" – ログ書き込みの最大バッファサイズ

`taskDefinitionArn` パラメータを使用してカスタムタスク定義を指定すると、タスク定義内でログ記録を設定できます。

サービスのデフォルト

以下の Amazon ECS Service 用のデフォルトは、Express Mode 内から設定できます。

- serviceName: Express Mode service → Name – サービスの名前 (お客様が提供したもの、またはイメージ名に由来するもの)

  名前は作成時にのみ設定でき、更新時に設定することはできません。

- cluster: Express Mode service → Cluster – このサービスの ECS クラスター (お客様が提供したもの、またはデフォルト)

  クラスターは作成時にのみ設定でき、更新時に設定することはできません。
- desiredCount: Express Mode service → MinTasks – サービスに必要な数 (デフォルトは 1)

  このオプションは、コンソールの追加設定オプションを使用するか、`create-express-gateway-service --scaling-target '{"minTaskCount"=3}'`, を実行することでカスタマイズできます。
- tags: AWS タグ付け用の Express Mode サービスタグ (お客様が提供したもの)

  このオプションは、コンソールの追加設定オプションを使用するか、`create-express-gateway-service --tags` を実行することでカスタマイズできます。タグは、新しいリソースの作成時にのみ追加できます。

以下のデフォルトは Express Mode によって設定されますが、Amazon ECS Service で直接更新できます。

- availabilityZoneRebalancing: true – 自動の AZ リバランシングを有効にします
- CapacityProviderStrategy: {"base": 1, "capacityProvider": "FARGATE", "weight": 1} – コンピューティングキャパシティに Fargate を使用します
- deploymentConfiguration: デフォルトは Canary – Express Mode サービスはカナリアデプロイを使用します

  Express Mode サービスではデプロイ戦略を更新できないことに注意してください。
- enableECSManagedTags: true – ECS マネージドリソースのタグ付けを有効にします
- healthCheckGracePeriodSeconds: 300 – スケジューラが ELB または Lattice ヘルスチェックを確認する前の猶予期間 (ELB ヘルスチェックの猶予期間のデフォルトと一致)
- launchType: <Not set> – キャパシティプロバイダーを使用します
- loadBalancers: ECS によって処理されるロードバランサーの設定

  Express Mode サービスではロードバランサー設定を更新できないことに注意してください。
- placementStrategy: 未設定 – 起動時に Fargate を使用します (常に AZ スプレッドである必要があります)
- platformVersion: LATEST – Fargate プラットフォームバージョン (LATEST にハードコーディング)

- propagateTags: "SERVICE" – サービスからタスクにタグを伝播します
- schedulingStrategy: "REPLICA" – Express Mode サービスは REPLICA サービスです
- taskDefinition: Express Mode service created – 使用するタスク定義 (Express Mode サービスが提供したもの)

ネットワーク設定のデフォルト

以下の Amazon ECS Service – ネットワーク設定用のデフォルトは、Express Mode 内から設定できます。

- networkConfiguration.Subnets: 何も指定しない場合、Express Mode はデフォルト VPC のデフォルトパブリックサブネットを使用します。

  デフォルト VPC には、2 つ以上のアベイラビリティーゾーンに少なくとも 2 つのパブリックサブネットがあり、サブネットごとに割り当てられた CIDR ブロックあたり使用できる IP が 8 つ以上ある必要があります。

  カスタムパブリックサブネットを指定すると、Express Mode はインターネット向け ALB をプロビジョニングし、タスクの assignPublicIP を有効にします。プライベートサブネット (ルートテーブルにインターネットゲートウェイがないサブネット) を指定すると、Express Mode は内部 ALB をプロビジョニングします。

  サブネットを指定すると、VPC の最初の Express Mode サービスは、その VPC のインターネット向けロードバランサーまたは内部ロードバランサーに関連付けられたサブネットを定義します。同じ VPC で起動する後続の Express Mode サービスには、ロードバランサーでサポートされているアベイラビリティーゾーンと一致するサブネットが必要です。このため、すべてのアベイラビリティーゾーンのサブネットを使用して Express Mode サービスを作成することをお勧めします。
- networkConfiguration.SecurityGroups: 何も指定されていない場合、Express Mode はサービスセキュリティグループとロードバランサーセキュリティグループの両方を作成します。

  サービスセキュリティグループは、パブリックインターネットへのアウトバウンドトラフィックを許可しますが、サブネット設定など、ネットワークの他の側面によっては制限される場合があります。

  ロードバランサーセキュリティグループは、指定したコンテナポート (デフォルトは 80) で、HTTPS 解決のためのインバウンドトラフィックとサービスセキュリティグループへのアウトバウンドトラフィックを許可します。また、Express Mode の Create または Update 呼び出しで提供されるコンテナポートと一致するように自動的に更新されます。

このオプションは、コンソールの追加設定オプションを使用するか、`create-express-gateway-service --network-configuration '{"securityGroup": ["sg-xxxxxxx"]}'` を実行することでカスタマイズできます。セキュリティグループを指定するときに、サービスへの追加のイングレスパスを提供します。

以下のデフォルトは Express Mode によって設定されますが、Amazon ECS Service – ネットワーク設定で直接更新できます。

- assignPublicIp: サブネットタイプに基づく – デフォルト VPC のユーザーがパブリックインターネットにアクセスできるようにパブリックサブネットに対して有効になります。デフォルトで各タスクのパブリック IP を有効にします。これは、プライベートサブネットを指定すると無効になり、タスクでインターネットアクセスが必要な場合は NAT ゲートウェイを設定する必要があります。

IAM ロールのデフォルト

以下の IAM ロールは Express Mode サービスに対して設定でき、必要に応じてサービスにリンクされたロールが自動的に作成されます。

- executionRoleArn: タスク実行ロール (必須パラメータ)

  必要なアクセス許可はマネージドポリシーにありますが、追加のポリシーとアクセス許可をアタッチできます。「AWS マネージドポリシーリファレンスガイド」の「AmazonECSTaskExecutionRolePolicy」で追加の詳細を参照してください。

- infrastructureRoleArn: Express Gateway サービスのインフラストラクチャロール (必須パラメータ)

  必要なアクセス許可はマネージドポリシーにありますが、追加のポリシーとアクセス許可をアタッチできます。「AWS マネージドポリシーリファレンスガイド」の「AmazonECSInfrastructureRoleforExpressGatewayServices」で追加の詳細を参照してください。

- taskRoleArn: コンテナアプリケーションコードからの呼び出し用のタスクロール (オプション)

  このオプションは、コンソールの追加設定オプションを使用するか、`create-express-gateway-service --task-role-arn` を実行することでカスタマイズできます。

Express Mode サービスを作成または更新する前に、新しい IAM ロールを作成する必要があります。

次の IAM ロールは、Express Mode によって自動的に設定されます。

- ecsServiceRoleForECS: Amazon ECS にクラスターの管理を許可します
- AWSServiceRoleForElasticLoadBalancing: 他の AWS サービスをユーザーに代わって呼び出します – Elastic Load Balancing によって自動的に作成されます
- AWSServiceRoleForApplicationAutoScaling_ECSService: ユーザーに代わって他の AWS サービスを呼び出します – Application Auto Scaling によって自動的に作成されます

`taskDefinitionArn` パラメータを使用してカスタムタスク定義を指定すると、タスク実行ロールとタスクロールはタスク定義から読み込まれます。

Application Auto Scaling のデフォルト

次の Application Auto Scaling の設定は、Express Mode で設定できます。

- autoScalingTargetValue: 60 – スケーリングのターゲット CPU 使用率
- autoScalingMetric: "CPUUtilization" – 自動スケーリングの決定に使用されるメトリクス
- desiredMinTaskCount: 1 – 維持するタスクの最小数
- desiredMaxTaskCount: 20 – スケールするタスクの最大数 (増減可能)

以下の設定は、Application Auto Scaling で直接設定できます。

- policyType: "TargetTrackingScaling" – ターゲット追跡スケーリングポリシーを使用します
- disableScaleIn: false – 需要が減少したときのスケールダウンを許可します

アプリケーションスケーリングターゲットのデフォルト

Application Auto Scaling ターゲットは、自動スケーリングオペレーションのスケーラブルなリソースと容量の制限を定義します。Express Mode では、以下を変更できます。

- MaxCapacity: Express Mode service → MaxTasks – サービスがスケールアップする最大タスク数 (デフォルトは CreateLoadBalancedService またはお客様提供)

- MinCapacity: Express Mode service → MinTasks – サービスがスケールダウンする最小タスク数 (デフォルトは CreateLoadBalancedService またはお客様提供)
- ScalableDimension: ecs:service:DesiredCount – スケールするプロパティ
- Tags: CreateLoadBalancedService で渡されたタグに由来 – このリソースのリソースタグ

Application Auto Scaling スケーラブルターゲットでは、次のデフォルトを変更できます。

- DynamicScalingInSuspended: false – スケーリングが中断されているかどうかを示します
- DynamicScalingOutSuspended: false – スケールアウトが中断されているかどうかを示します
- ScheduledScalingSuspended: false – スケジュールされたスケーリングが中断されているかどうかを示します

アプリケーションスケーリングポリシーのデフォルト

スケーリングポリシーは、CPU 使用率をモニタリングし、最適なパフォーマンスを維持するためにタスク数を調整するターゲット追跡アルゴリズムを実装します。Express Mode から以下を変更できます。

- TargetValue: 60% – 追跡ポリシーのターゲット値 (Express Mode service → scaling-target-value)
- PredefinedMetricType: ECSServiceAverageCPUUtilization – 追跡されるメトリクスの名前タイプ (Express Mode service → scaling-metric)

  Express Mode は、平均 CPU 使用率、平均メモリ使用率、ターゲットあたりのリクエスト数のスケーリングメトリクスを提供します。ターゲットあたりのリクエスト数は Application Load Balancer に由来し、1 秒あたりターゲットごとに最大 65536 リクエストまで設定できます。これが Application Load Balancer ターゲットグループの Amazon ECS サービスです。

次のデフォルトは Express Mode によって設定されますが、Application Auto Scaling ポリシーで変更できます。

- PolicyName: <ServiceName><"TargetMetric"> – このスケーリングポリシーの名前 (ECS サービス名とターゲットメトリクスに由来)
- PolicyType: TargetTrackingScaling – 実行されるスケーリングのタイプ (事前定義されたデフォルト)
- ScalableDimension: "ecs:service:DesiredCount" – スケーリングされるプロパティ (事前定義されたデフォルト)

- DisableScaleIn: false – スケールインが無効かどうかを示します

Application Load Balancer のデフォルト

Express Mode では、次の Application Load Balancer のデフォルトを構成できます。

- scheme: サブネットによって異なる – パブリックサブネットが指定された場合、パブリック IP アドレスを持つインターネット向けロードバランサーを作成します。プライベートサブネットが指定された場合、プライベート IP アドレスを持つ内部ロードバランサーを作成します。
- ip-address-type: サブネットによって異なる – アドレスが IPv4 の場合、IPv4 のみの Application Load Balancer を作成します。IPv6 対応のサブネットがある場合、デュアルスタック Application Load Balancer が作成されます。IPv6 対応のサブネットがある場合、VPC の最初の Express Mode サービスは、その VPC の内部ロードバランサーまたはインターネット向けロードバランサーを定義することに注意してください。このため、IPv6 のみのサブネットを最初に作成するか、新しい VPC 内に作成することをお勧めします。

次のデフォルトは Express Mode で設定されますが、Application Load Balancer で構成できます。

- desync-mitigation-mode: Off – HTTP 非同期緩和が無効になります
- access-logs.enabled: false – アクセスログ記録が無効になります
- listener-configurations.protocol: https – 安全な通信に HTTPS プロトコルを使用します
- listener-configurations.port: 443 – 標準 HTTPS ポートでリッスンします
- listener-configurations.rule-type: host-header – ホストヘッダールールに基づいてトラフィックをルーティングします

ターゲットグループのデフォルト

次の Amazon EC2 ターゲットグループのデフォルトは、Express Mode で設定できます。

- health-check-path: (デフォルト "/") Express Mode サービスのヘルスチェックパス – ヘルスチェックリクエストの URL パス
- port: (デフォルト 80) – ターゲットがトラフィックを受信するポート
- health-check-port: (デフォルト 80) – ヘルスチェックリクエストのポート

次のデフォルトは Express Mode で設定されますが、Amazon EC2 ターゲットグループで構成できます。

- protocol: HTTP – ターゲットにトラフィックをルーティングするためのプロトコル
- protocol-version: HTTP1 – 通信用の HTTP プロトコルバージョン
- vpc-id: デフォルトはデフォルト VPC – ターゲットグループの仮想プライベートクラウド識別子ですが、指定されたサブネットに由来します
- health-check-protocol: プロトコルと同じ – ヘルスチェックリクエストのプロトコル
- health-check-enabled: Always enabled – ヘルスチェックは自動的に有効になります
- health-check-interval-seconds: 30 – 各ターゲットのヘルスチェック間隔
- health-check-timeout-seconds: 5 – ヘルスチェック応答のタイムアウト期間
- healthy-threshold-count: 5 – 正常なステータスに必要な連続した正常なヘルスチェック
- unhealthy-threshold-count: 2 – 異常なステータスに必要な連続して失敗したヘルスチェック
- target-type: ip – ターゲットは IP アドレスによって登録されます
- ip-address-type: ipv4 – ターゲット登録に IPv4 アドレスを使用します

リソース共有とコストの最適化

Express Mode サービスは、可能な限りリソースを自動的に共有してコストを最適化します。

- ロードバランサーの共有 – 同じ VPC 内の最大 25 の Express Mode サービスが Application Load Balancer を共有できます。Express Mode は、プロビジョニングした Express Mode サービスの量に基づき、必要に応じて追加の Application Load Balancer をプロビジョニングします。また、VPC の Express Mode サービスの量を減らすと、Express Mode は未使用の Application Load Balancer のプロビジョニングも解除します。このような共有により、Express Mode サービスをデプロイするほど、アプリケーションあたりのコストを削減できます。
- クラスター共有 – Express Mode サービスは、Amazon ECS クラスターでグループ化できます。Express Mode サービスは、Amazon ECS クラスターを Express Mode で管理されていない Amazon ECS サービスと共有することもできます。

Amazon ECS Express Mode サービスの詳細の表示

Express Mode サービスの詳細を表示して、そのステータス、設定、パフォーマンスメトリクスをモニタリングできます。サービスの詳細ページには、サービスの現在の状態、実行中のタスク、デプロイ履歴、運用メトリクスに関する包括的な情報が表示されます。

Express Mode サービスを表示するには (Amazon ECS コンソール)

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。
3. [クラスター] ページで、Express Mode サービスを含むクラスターの名前を選択します。
4. [クラスターの詳細] ページで、[サービス] タブを選択します。
5. Express Mode サービスを表示するようにフィルターを設定します。[リソース管理タイプを絞り込む] で、ECS を選択します。

   Express Mode サービスには、名前の横に Express バッジがあります。
6. お使いの Express Mode サービスの名前を選択して、Express Mode サービスのサービス詳細ページに移動します。
7. サービス詳細ページに表示されるサービス情報を確認します。

   a. Express サービスの概要 – アプリケーション URL、サービスのステータス、保留中および実行中のタスク、アクティブなデプロイなどの基本的なサービス情報が表示されます。

   b. オブザーバビリティ – CPU 使用率 (最小、最大、平均)、メモリ使用率 (最小、最大、平均)、ターゲット応答時間 (p50、p90、p99)、RequestCount、4xx ステータス、5xx ステータスの CloudWatch メトリクスが表示されます。

   c. ログ – タスク内のコンテナからのログが表示されます。

   d. リソース – Express Mode サービス用に作成された AWS リソース。タイムラインビューは、作成または更新後に自動的に更新され、ステータスとイベントが表示されます。テーブルビューには、各リソースのステータスと完全な ARN を含む検索可能なテーブルが表示されます。

   e. デプロイ – デプロイ履歴と現在のデプロイステータスを追跡します。更新中に貴重な情報を提供するサービスリビジョンとタスクが含まれます。イベントパネルは、Express Mode サービスが定常状態に達したことを知らせます。

   f. 設定 – クラスター、ロール、コンテナ設定、コンピューティング、自動スケーリング、ネットワーク設定などのサービス設定が表示されます。

g. タグ – タグを管理します。

Amazon ECS Express Mode サービスの更新

Express Mode サービスは、ネットワーク、ロードバランシング、Application Auto Scaling の設定可能なオプションを提供し、これらのサービス全体の変更をオーケストレーションすることで、サービスの更新プロセスを簡素化します。

Express Mode サービスを更新して、コンテナイメージの変更、リソース割り当ての調整、または設定の変更を行うことができます。更新はカナリアデプロイを使用してデプロイされ、可用性を維持するためにアラームベースのロールバックアラームが使用されます。

デプロイの動作

Express Mode サービスは、デフォルトでカナリアデプロイを使用して、安全な更新と迅速なロールバックを実現します。

- 新しい環境が作成され、変更とともにタスクがデプロイされます
- トラフィックの 5% が新しい環境に移行します
- 4xx と 5xx エラーの割合の合計が 3 分以内に 2 つのデータポイントで 1 より大きい場合、アラームベースのロールバックがトリガーされます
- ヘルスチェックは新しいタスクが正常であることを確認します
- ベイク時間が 3 分経過すると、トラフィックの 100% が新しい環境に移行されます
- モニタリング用のベイク時間が 3 分経過すると、古いタスクは徐々に停止して置き換えられます

詳細については、「Amazon ECS カナリアデプロイ」を参照してください。

手順

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。
3. [クラスター] ページで、Express Mode サービスを含むクラスターの名前を選択します。
4. [クラスターの詳細] ページで、[サービス] タブを選択します。
5. Express Mode サービスを表示するようにフィルターを設定します。[リソース管理タイプを絞り込む] で、ECS を選択します。

Express Mode サービスには、名前の横に Express バッジがあります。

6. [構成] の下:

   a. アプリケーションに使用するイメージを指定します。[イメージ URI] でイメージの URI を入力します。Amazon ECR イメージを参照するには、[ECR イメージを参照] を選択し、次の操作を行います。

      i. [プライベートリポジトリ] で、Amazon ECR プライベートリポジトリを選択します。

      ii. [イメージ] で、イメージを選択します。

      iii. イメージを識別する方法を選択します。[イメージの選択方法] で、以下のオプションのいずれかを選択します。

         - AWS は [イメージダイジェスト] を選択することをお勧めしています。
         - タグを使用するには、[イメージタグ] を選択してからタグを選択します。

   b. プライベートレジストリを使用するには、[プライベートレジストリ] を選択します。次に、[Secrets Manager ARN または名前] に前提条件で作成した Secrets Manager ARN を入力します。

   c. [タスク実行ロール] で、ロールを選択するか、新しいロールを作成して更新します。アクセス許可を追加する必要がある場合は、タスク実行ロールを更新できます。

7. [追加設定] で、サービスをカスタマイズします。

   a. [コンテナ] で以下を実行します。

      i. [コンテナポート] で、アプリケーションがリッスンするポートを更新します (デフォルトは 80)。

      ii. [ヘルスチェックパス] で、ヘルスチェックのパス (`/health` など) を更新します。

   b. [環境変数] で、アプリケーションに必要な環境変数のキーと値のペアを追加します。

      i. [キー] に、環境変数の名前を入力します。

      ii. [値タイプ] で、[環境変数] または [シークレット] を選択します。

      iii. [値または値の取得] には、値またはリファレンスを入力します。

      iv. [環境変数の追加] を選択して、必要に応じて変数を追加します。

   c. [コマンド] に、オプションでカスタムコマンドを入力して Docker CMD 命令を上書きします。

d. [タスクロール] で、コンテナで実行されているアプリケーションにアクセス許可を付与する IAM ロールを追加します。これにより、アプリケーションは AWS サービスに対して API コールを実行できます。

e. [コンピューティング] で以下を実行します。

   i. [CPU] で、タスクの vCPU 割り当てを更新します (1 vCPU など)。

   ii. [メモリ] で、タスクのメモリ割り当てを更新します (2 GB など)。

f. [自動スケーリング] で以下を実行します。

   i. [ECS サービスメトリクス] で、スケールするためのメトリクス (ECS サービスの平均メモリ使用率やターゲットあたりのリクエスト数など) を選択します。

   ii. [ターゲット値] で、スケーリングのターゲット (60 や 1000 など) を入力します。

   iii. [最小タスク数] と [最大タスク数] で、スケーリング制限を更新します。

g. [ログ] で以下を実行します。

   i. [Amazon CloudWatch ロググループ] で、アプリケーションログのロググループ名を更新します。これにより、既存のログは移動されませんが、新しいサービスリビジョンからのログの書き込みが開始されます。

   ii. [Amazon CloudWatch ログストリームプレフィックス] に、ログストリームの新しいプレフィックスを入力します。

8. [更新] を選択して、Express Mode サービスを更新します。

カスタムタスク定義による更新

`taskDefinitionArn` パラメータを指定することで、カスタムタスク定義で Express Mode サービスを更新できます。Express Mode はタスク定義をそのまま使用し、残りのインフラストラクチャの管理を継続します。

以降の更新では、以下のいずれかを選択できます。

- タスク定義の新しいリビジョンを作成し、`taskDefinitionArn` を更新します。Express Mode は、次のサービスデプロイに新しいリビジョンを使用します。
- Express Mode パラメータを直接更新します。
  - タスク定義に関連しないパラメータ (`healthCheckPath`、`networkConfiguration`、`scalingTarget` など) の場合、Express Mode はタスク定義を変更せずに変更を適用します。

- タスク定義関連のパラメータ (`primaryContainer` または `cpu/memory` を使用したイメージ、CPU、メモリなど) の場合、Express Mode は、他のタスクレベルの設定を維持しながら、変更を加えた新しいマネージドタスク定義を作成します。

カスタムタスク定義で更新するには (Amazon ECS コンソール)

1. Amazon ECS コンソールを開き、Express Mode サービスに移動します。
2. サービスの更新 の横にあるドロップダウン矢印を選択し、カスタムタスク定義で更新 を選択します。
3. Task definition family(タスク定義ファミリ) については、既存のタスク定義ファミリを選択してください。新しいタスク定義を作成するには、タスク定義 リンクを選択してください。
4. タスク定義リビジョン では、使用するリビジョンを選択するか、空白のままにして最新のリビジョンを使用します。
5. (オプション) 追加設定 を展開して、ヘルスチェックパス、ネットワーク、または自動スケーリングの設定を更新します。
6. [更新] を選択します。

次の AWS AWS CLI コマンドは、カスタムタスク定義で Express Mode サービスを更新します。

```
aws ecs update-express-gateway-service \
    --service-arn arn:aws:ecs:us-west-2:123456789012:service/default/my-service \
    --task-definition-arn arn:aws:ecs:us-west-2:123456789012:task-definition/my-td:4
```

> **Note**
>
> CloudFormation を使用し、`taskDefinitionArn` を初めて Express Mode サービスに追加する場合は、最初にスタックを再デプロイします。これにより、ロールバックの互換性が確保されます。

Amazon ECS Express Mode サービスを削除する

Express Mode サービスを削除すると、そのサービスが削除され、クラスター、ロードバランサー、その他のインフラストラクチャコンポーネントなどの関連リソースも削除されます。

> **Important**
>
> Express Mode サービスを削除すると、実行中のすべてのタスクが停止し、アプリケーションが使用できなくなります。このアクションを元に戻すことはできません。

選択内容によっては、次のリソースが削除される場合があります。

- Amazon ECS クラスター (他のサービスが実行されていない場合)
- Amazon ECS サービス、タスク定義、および実行中のタスク
- サービスセキュリティグループ
- CloudWatch ロググループ
- メトリクスアラーム
- ACM 証明書
- Application Load Balancer (他のサービスが設定されていない場合)、ターゲットグループ、セキュリティグループ、リスナー、リスナールール
- Auto Scaling ポリシー、スケーラブルターゲット

以下の点を考慮してください。

- 削除する前に、重要なデータをバックアップしていることを確認する
- 削除する前にサービスをゼロタスクにスケールしてトラフィックを適切に停止することを検討する
- 削除の影響を受ける可能性のある依存関係を確認する

Express Mode サービスの削除とサービスの削除の違い

サービスを削除すると、実行中のタスクがない場合、Amazon ECS はサービスを削除します。Amazon ECS は、サービスまたはデフォルトクラスターに関連付けられたターゲットグループを削除しません (使用されている場合)。Express Mode サービスを削除すると、Amazon ECS は Express Mode サービスに特有のサービスおよびリソースを削除します。例えば、Express Mode サービスがロードバランサーを作成し、ロードバランサーを使用する唯一のサービスである場合、関連付けられたロードバランサーは削除されます。ロードバランサーが共有されている場合、Express Mode で管理されていないリソースであっても、ロードバランサーは削除されません。

手順

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. [Clusters] (クラスター) ページで、サービスに使用するクラスターを選択します。
3. [Clusters] (クラスター) ページで、クラスターを選択します。
4. [サービス] タブを選択します。
5. 削除するサービスを選択します。
6. [削除] を選択します。
7. [削除の確認] ダイアログボックスで、
   - **`delete`** と入力して削除を確定します。
8. [Delete] を選択して確定します。
9. このページにとどまり、削除の進行状況をモニタリングし、削除されたリソースのリストと、保持され、手動のクリーンアップが必要になる可能性のあるリソースのリストを確認します。

Amazon ECS Express Mode サービスのベストプラクティス

実稼働環境で Express Mode サービスを効果的に使用するためのベストプラクティスと推奨事項について説明します。

セキュリティのベストプラクティス

シークレットの管理

- シークレットに Secrets Manager を使用する – Secrets Manager に機密データ (プライベートリポジトリやデータベース認証情報など) を保存します。

  Secrets Manager のベストプラクティスの詳細については、「Secrets Manager ユーザーガイド」の「Secrets Manager のベストプラクティス」を参照してください。
- 保管時の暗号化を有効にする – AWS サービスに保存されているときにシークレットが暗号化されていることを確認します。

  Secrets Manager などのサービスを使用すると、AWS マネージドキーまたはお客様提供のキーを使用して暗号化できます。
- シークレットローテーションを実装する – データベースパスワードと API キーの自動ローテーションを使用します。

Secrets Manager などのサービスを使用すると、Amazon Aurora や Amazon RDS などのサービスのシークレットローテーションを管理できます。

Express Mode サービスでのシークレットの使用例:

```
aws ecs update-express-gateway-service \
    --primary-container \
        '{"environment"=[{"name"="DB_PASSWORD","value"="arn:aws:secretsmanager:us-
west-2:123456789012:secret:prod/db/password"}, \
        {"name"="API_KEY","value"="arn:aws:ssm:us-west-2:123456789012:parameter/prod/
api-key"}]}' \
```

ネットワークセキュリティ

- 機密アプリケーションにプライベートサブネットを使用する – 直接的なインターネットアクセスを必要としないアプリケーションをプライベートサブネットにデプロイします。

  推奨されるアーキテクチャの詳細については、「Amazon ECS アプリケーションをインターネットに接続する」を参照してください。
- セキュリティグループに最小限のアクセス許可を設定する – インバウンドトラフィックとアウトバウンドトラフィックを必要なポートとソースのみに制限します。

  Express Mode サービスのセキュリティグループのアウトバウンドトラフィックを制限するには、アウトバウンドルールを変更するか、次のコマンドを使用して、Amazon EC2 セキュリティグループコンソールでこれを直接編集できます。

```
aws ec2 authorize-security-group-egress
    --group-id sg-xxxxxxxx \
    --protocol tcp \
    --port 443 \
    --cidr 0.0.0.0/0

aws ec2 revoke-security-group-egress
    --group-id sg-xxxxxxxx \
    --protocol tcp \
    --port 443 \
```

```
    --cidr 0.0.0.0/0
```

- Amazon VPC フローログを有効にする – セキュリティ分析とトラブルシューティングのためにネットワークトラフィックをモニタリングします。

  VPC サブネットコンソールの Express Mode アプリケーションで使用中の各サブネットで有効にすることも、 `aws ec2 create-flow-logs --resource-ids subnet-xxx` を使用することもできます。
- ウェブアプリケーションに AWS WAF を使用する – 一般的なウェブエクスプロイトや攻撃から保護します。

  これを有効にするには、ウェブ ACL を作成し、それを Express Mode サービスで使用される Application Load Balancer に関連付けます。コンソールで、WAF & Shield Service にウェブ ACL を作成し、Application Load Balancer に関連付けます。または、`aws wafv2 create-web-acl` と `aws wafv2 associate-web-acl --resource-arn <alb>` を使用します。

パフォーマンスとコンピューティングの最適化

リソースのサイズ設定

- 適切なサイズの CPU とメモリ – アプリケーションのパフォーマンスをモニタリングし、実際の使用パターンに基づいて CPU とメモリの割り当てを調整します。

  AWS Compute Optimizer は、Amazon ECS のタスクとコンテナのサイズに関する推奨事項を生成します。詳細については、AWS Compute Optimizer ユーザーガイド の「What is AWS Compute Optimizer? (とは ?)」 を参照してください。
- アプリケーションのパフォーマンステスト – アプリケーションが大規模かつ、指定されたスケーリングしきい値とリソース割り当てで動作することを確認するには、負荷テストを実行します。

自動スケーリング設定

- 適切なスケーリングしきい値を設定する – パフォーマンスが低下する前にスケーリングをトリガーする CPU またはメモリのしきい値を設定します。

  サービスメトリクスのターゲット値は、Express Mode サービスコンソールで変更できます。

特にトラフィックに時間ベースのパターンが見られる場合は、予測的なスケーリングポリシーを追加することを検討してください。詳細については、「予測自動スケーリング」を参照してください。

- 複数のスケーリングメトリクスを使用する – CPU またはメモリとリクエストベースのスケーリング両方を使用して、より応答性の高いスケーリングを行うことを検討してください。

  サービスに複数のポリシーを追加できます。Express Mode はデフォルトで 1 つを追加しますが、追加のポリシーをサービスに直接アタッチできます。
- タスクの最小数と最大数の制限を設定する – コストを制御し、可用性を確保するために合理的な制限を設定します。

  本稼働ワークロードの場合、初期テストが完了したら、3 つのアベイラビリティーゾーンで実行して、アベイラビリティーのベストプラクティスに従うことをお勧めします。Express Mode コンソールでタスクの最小数を更新するか、`update-express-gateway-service --scaling-target '{“minTaskCount”=3}'` を使用します。

ヘルスチェック

- 有意義なヘルスチェックを実装する – 重要なアプリケーションの依存関係を検証するヘルスチェックエンドポイントを作成します。

  ヘルスチェックパスは、Express Mode コンソールで更新できます。または、`update-express-gateway-service --health-check-path "/health"` を使用します。

  アプリケーションのヘルスチェック形成の詳細については、「ヘルスチェックの実装」を参照してください。
- ヘルスチェックを軽量に保つ – ヘルスチェックエンドポイントでの高価な操作を回避します。

  例としては、外部 API コール、CPU またはメモリを大量に消費する操作、タイムアウトになる可能性がある長時間実行される操作などがあります。
- 適切なタイムアウトを使用する – 障害をすばやく検出しながら正常な応答時間を実現するヘルスチェックタイムアウトを設定します。

  Express Mode のヘルスチェックタイムアウトは、Application Load Balancer ターゲットグループで構成できます。Amazon EC2 コンソールで、[ターゲットグループ] セクションに移動し、Express Mode ターゲットグループを選択します。[ヘルスチェック] タブを選択し、[編集] をクリックします。[ヘルスチェックの詳細設定] でタイムアウトを調整できます。または、aws

`elbv2 modify-target-group --target-group-arn <targetgroup> --health-check-timeout` を使用します。

- 適切な HTTP ステータスコードを返す – 正常な状態では 200、異常な状態では 4xx/5xx を使用します。

運用のベストプラクティス

モニタリングとログ記録

- 強化された Container Insights を有効にする – CloudWatch の強化された Container Insights を使用して、Express Mode サービスアプリケーションを包括的にモニタリングします。

  詳細については、「Amazon ECS での Container Insights のセットアップ」を参照してください。
- カスタムメトリクスを設定する – ビジネスロジックのモニタリングのために、アプリケーション固有のメトリクスを CloudWatch に発行します。

  詳細については、「CloudWatch ユーザーガイド」の「カスタムメトリクスの発行」を参照してください。
- ログの保持を設定する – 適切なログ保持期間を設定し、コストとコンプライアンス要件のバランスを取ります。

  Express Mode によって作成された CloudWatch ロググループは、有効期限が切れないように設定され、Express Mode サービスが削除されても保持されます。この設定は CloudWatch ロググループで調整できます。
- ダッシュボードとアラートを作成する – CloudWatch のダッシュボードとアラームをプロアクティブなモニタリング用に設定します。

デプロイ戦略

- ベイク時間を実装する – Express Mode は Canary ベイク時間を実装して、問題のあるデプロイの影響範囲を減らしながらデプロイを安定させる時間を確保します。アプリケーションの安定化にさらに時間が必要な場合は、Express Mode サービスの Amazon ECS Service 定義で構成できます。詳細については、「Amazon ECS カナリアデプロイの作成」を参照してください。
- ロールバック手順を実装する – 問題が発生した場合は、以前のバージョンにすばやく戻す計画を立てます。

有意義なヘルスチェックとアラームベースのロールバックはどちらもロールバックに役立ちます。Express Mode のカナリアデプロイ戦略を 4xx および 5xx トラフィックのアラームベースのロールバックと組み合わせると、アプリケーションコードまたは設定に障害が発生した場合に迅速にロールバックするようにデプロイをセットアップできます。

カスタムタスク定義を使用するタイミング

以下の場合にのみ、独自のカスタムタスク定義を提供することを検討してください。

- チームには、すべてのサービスに存在する必要があるセキュリティエージェント、ログ記録サイドカー、またはコンプライアンス設定を含む標準化されたタスク定義があります。
- コンテナのヘルスチェック、カスタムログドライバー (FireLens など)、Linux パラメータ (`initProcessEnabled`、`tmpfs` マウント)、追加のサイドカーコンテナなど、Express Mode で直接設定できないタスクレベルの設定が必要です。
- インフラストラクチャをコードとして使用し、タスク定義をテンプレートで定義して、Express Mode サービスに渡します。

Amazon ECS Express Mode サービスのトラブルシューティング

このセクションは、Express Mode サービスをデプロイして管理する際に一般的な問題を特定して解決するのに役立ちます。

デプロイに関する問題

サービスが ACTIVE または DRAINING ステータスでスタックする

症状: DescribeServiceRevisions は、リソースがまだプロビジョニング中またはプロビジョニング解除中であることを示します。DescribeServices はデプロイが安定していないことを示します

考えられる原因と解決策:

- IAM アクセス許可が不十分 – タスク実行ロールとインフラストラクチャロールに、それぞれの管理ポリシーに示されているように必要なアクセス許可があることを確認します。

```
# Check if the role has the required managed policy
```

```
aws iam list-attached-role-policies --role-name ecsTaskExecutionRole
```

- イメージプルの失敗 – コンテナイメージが存在し、アクセス可能であることを確認します。

```
# Test image pull manually
docker pull 123456789012.dkr.ecr.us-west-2.amazonaws.com/my-app:latest
```

- ネットワーク接続の問題 – サブネットにインターネットアクセスまたは AWS サービスの Amazon VPC エンドポイントがあることを確認します。
- リソース制限 – アカウントに十分な Fargate キャパシティがあり、Service Quotas に達していないことを確認します。

診断手順:

1. DescribeExpressGatewayService を使用して、ServiceRevision の現在のサービスリビジョンを取得した後、DescribeServiceRevisions を使用してプロビジョニングまたはプロビジョニング解除のステータスを取得します。
2. 詳細なエラーメッセージについては、Amazon ECS コンソールのサービスイベントを確認してください。
3. コンテナポートが正しく設定されていることを確認します。
4. Amazon ECS と Fargate の AWS Service Quotas を確認します。

タスクスタートアップの失敗

症状: タスクが開始に失敗する、または開始直後に停止する。

一般的な原因:

- アプリケーションエラー – コンテナアプリケーションは、設定エラーまたはランタイムエラーにより終了します。
- ヘルスチェックの失敗 – アプリケーションが、予想されるポートまたはパスでのヘルスチェックに応答していません。
- リソース制約 – アプリケーションの CPU またはメモリの割り当てが不十分です。
- 環境変数またはシークレットがない – アプリケーションが必要な設定を使用できません。

解決の手順:

1. CloudWatch Logs でアプリケーションログを確認し、DescribeServiceRevisions からロググループ名を取得します。

```
aws logs describe-log-streams --log-group-name /ecs/express-service-my-app
aws logs get-log-events --log-group-name /ecs/express-service-my-app --log-stream-
name stream-name
```

2. ヘルスチェックパスが HTTP 200 ステータスを返すことを確認します。
3. コンテナイメージをローカルでテストして、正しく起動することを確認します。
4. 必要に応じて CPU とメモリの割り当てを確認して調整します。

カスタムタスク定義エラー

`taskDefinitionArn` パラメータを使用する場合、サービスの作成または更新は `InvalidParameterException` で失敗します。

「taskDefinitionArnには、primaryContainer、executionRoleArn、taskRoleArn、cpu、または memory を指定することはできません。」

`taskDefinitionArn` パラメータは、同じ API コールの `primaryContainer`、`executionRoleArn`、`taskRoleArn`、`cpu`、または `memory` で指定することはできません。Express Mode は、指定されたタスク定義からこれらの値を取得します。

「タスク定義には、単一の TCP ポートマッピングを持つ「Main」という名前のコンテナを含める必要があります。」

タスク定義が Express Mode 要件を満たしていません。以下があることを確認します。

- コンテナ名 `Main`
- `Main` コンテナには、コンテナポートとポート名が定義された TCP ポートマッピングが 1 つだけあります。
- タスク定義の互換性には `FARGATE` があります

接続の問題

ロードバランサー経由でアプリケーションにアクセスできない

症状: アプリケーション URL がタイムアウトまたは接続エラーを返す。

トラブルシューティングのステップ:

1. リソースのプロビジョニングが完了していることを確認します。
2. タスクが実行中であり、正常であることを確認します。

```
aws ecs describe-services --cluster my-cluster --services my-express-service
```

3. Application Load Balancer ターゲットグループのヘルスをチェックします。

```
aws elbv2 describe-target-health --target-group-arn
 arn:aws:elasticloadbalancing:region:account:targetgroup/name/id
```

4. アプリケーションがコンテナ内の正しいポートでリッスンしていることを確認します。

パフォーマンスの問題

応答時間が遅い

症状: アプリケーションの応答が想定よりも遅い。

診断アプローチ:

1. CPU とメモリの使用率をモニタリングします。

```
# Check CloudWatch metrics for the service
aws cloudwatch get-metric-statistics \
    --namespace AWS/ECS \
    --metric-name CPUUtilization \
    --dimensions Name=ServiceName,Value=my-express-service Name=ClusterName,Value=my-
cluster \
    --start-time 2024-01-01T00:00:00Z \
```

```
    --end-time 2024-01-01T01:00:00Z \
    --period 300 \
    --statistics Average
```

2. アプリケーションログにエラーやパフォーマンスの警告がないか確認します。
3. 自動スケーリングが負荷に適切に応答しているかどうかを確認します。
4. リクエスト分散のロードバランサーメトリクスを分析します。

最適化戦略:

- リソースに制約がある場合は、CPU またはメモリの割り当てを増やします。
- 先にスケールアウトするよう、自動スケーリングのしきい値を調整します。
- アプリケーションコードとデータベースクエリを最適化します。

自動スケーリングが想定どおりに機能しない

症状: サービスが高負荷時にスケールアップしないか、低負荷時にスケールダウンしない。

トラブルシューティングのステップ:

1. 自動スケーリングポリシーとその設定を確認します。

```
aws application-autoscaling describe-scaling-policies \
    --service-namespace ecs \
    --resource-id service/my-cluster/my-express-service
```

2. CloudWatch メトリクスを確認して、スケーリングトリガーが満たされていることを確認します。
3. サービスにスケールする権限があることを確認します (IAM ロールを確認)。
4. スケーリングアクティビティとその結果を確認します。

モニタリングとデバッグツール

CloudWatch Container Insights の使用

Container Insights を有効にして包括的なモニタリングを行います。

```
aws ecs put-account-setting --name containerInsights --value enabled
```

Container Insights は以下を提供します。

- CPU、メモリ、ディスク、ネットワークメトリクス
- パフォーマンスモニタリングダッシュボード
- ログの相関関係と分析
- 異常検出

Express Mode 以外のリソースの更新

Express Mode は、ユーザーに代わって AWS リソースを自動的にプロビジョニングし管理することでデプロイを簡素化しますが、責任モデルを理解することが大切です。Express Mode は Amazon ECS サービス、ロードバランサー、自動スケーリングポリシーなどのリソースを作成し設定しますが、すべてのリソースは AWS アカウントに残り、アクセスして直接管理できます。これらのリソースは、AWS コンソール、AWS CLI、または API を使用して Express Mode の外部で変更できますが、Express Mode のサービスを管理する機能に影響する可能性があります。このセクションでは、Express Mode が管理するリソース、Express Mode の外部で安全に変更を行う方法、および変更による影響について説明します。

責任共有モデルを理解する

Express Mode の責任境界は、API の使用によって定義されます。Express Mode API は、複数の AWS サービス間で調整された更新により、ユーザーに代わってリソースを管理します。標準 AWS サービス API を使用すると、個々のリソースを直接制御できます。これにより、Express Mode が設定用に提供するパラメータのセットを超えて、すべてのリソースのすべてのパラメータを制御できます。どちらの API セットも、アカウント内の同じ基盤となるインフラストラクチャで動作し、どちらもリクエストしたオペレーションを実行します。AWS には、リクエストしたオペレーションを実行する責任があります。Express Mode は、直接 API を使用したリソースの変更が Express Mode の設定と競合するかどうかを検証せず、サービス機能を中断する可能性のある変更を妨げることもありません。つまり、変更を加えることができ、その変更は保持されます。Express Mode の更新の一部としてリクエストされない限り、Express Mode は変更を上書きしません。ユーザーには、直接 API を使用した変更が Express Mode の設定にどのように影響するかを理解し、結果として生じる競合やサービスの問題を解決する責任があります。

Express Mode が設定するデフォルトと、それがオーケストレーションするリソースの詳細については、「the section called “Express Mode サービスによって作成されたリソース ”」を参照してください。

Express Mode サービスで Amazon ECS 機能を利用する前に、何かする必要はありますか?

いいえ。Express Mode にはグラデュエーションパスやブレークグラスはありません。Amazon ECS 機能セットはすべて、Express Mode サービスで常に使用できます。サービスでこれ以上 Express Mode の更新が発生しないようにするには、いくつかオプションがあります。リソースのマネージドタグを削除することで、Amazon ECS は Express Mode サービスとしてそれを識別して操作できなくなります。あるいは、IAM を使用して、ユーザーのアクセス許可を Express Mode API に制限することもできます。

Express Mode 外での更新の例

Express Mode サービスへのサイドカーの追加

任意のサイドカーコンテナをサービスに追加できます。この例は、Fluent Bit ログ記録サイドカーを追加する方法を示しています。

サイドカーコンテナを使用してタスク定義を定義し、`taskDefinitionArn` パラメータを使用して Express Mode サービスに提供できます。Express Mode では、設定したサイドカーコンテナを含め、タスク定義をそのまま使用します。詳細については、「the section called “独自のタスク定義を使用して Express Mode サービスを作成”」を参照してください。

または、次の手順で説明するように、Express Mode が作成したタスク定義を直接編集してサイドカーを追加することもできます。

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。
3. Express Mode サービスがあるクラスターを選択し、[サービス] パネルで Express Mode サービスを選択します。
4. [リソース] タブに移動し、タスク定義を選択します。
5. [JSON] タブを選択し、[新しいリビジョンを作成] というラベルが付いた上部ボタンの下矢印を選択します。ここには、JSON で新しいリビジョンを作成するというアクションがあります。
6. プライマリコンテナ定義の前後にサイドカーコンテナを追加します。詳細については、「Amazon ECS タスク定義の例: FireLens にログをルーティングする」を参照してください。

7. この例のように、ログ記録サイドカーがある場合は、プライマリコンテナで `logDriver` を変更します。

```
"logConfiguration": {
    "logDriver": "awsfirelens"
}
```

8. Express Mode サービスのデフォルト設定を前提として、プライマリコンテナ memoryReservation を 1998 に設定し、Fluent Bit コンテナ memoryReservation を 50 に設定します。これにより、2 つのコンテナの合計メモリ予約数は Fargate タスクで実行している量 (デフォルトの Express Mode サービスの場合は 2048) を超えません。
9. 新しいリビジョンを作成します。
10. [デプロイ] ドロップダウンをクリックし、[サービスの更新] を選択します。
11. 更新するクラスターとサービスを選択し、[新しいデプロイを適用する] を選択します。先ほど作成したタスク定義のリビジョンが自動で選択されるはずです。[更新] を選択します。
12. デプロイが行われていることを確認します。このサービスのタスクに 2 つのコンテナが追加されました。Express Mode コンソールまたは API による追加の更新は、直接競合しない限り、タスク定義への変更を上書きしません。この場合だと、例えば、Express Mode に新しいロググループとログストリーム名を指定した場合、logDriver は `awslogs` に更新されます。

サービスへのカスタムドメインの追加

前提条件: このガイドでは、ドメインが ACM 証明書を使用して Route 53 によって管理されるホストゾーンであることを前提としています。ドメインが他の場所でホストされている場合は、Application Load Balancer DNS レコードを指す CNAME が必要です。

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。
3. Express Mode サービスがあるクラスターを選択し、[サービス] パネルで Express Mode サービスを選択します。
4. [リソース] タブに移動し、リスナールールを選択します。
5. [アクション]、[ルールを編集] の順に選択します。
6. 現在のホストヘッダー値 (アプリケーション URL) をコピーし、ルールを削除します。これは各タイプのいずれか 1 つしか持てないためです。

7. ホストヘッダータイプの条件を追加し、アプリケーション URL をホストヘッダー条件値として貼り付けます。
8. [OR 条件値の追加] をクリックします。
9. ここにカスタムドメインを入力します。
10. [次へ]、[変更を保存] の順にクリックします。
11. 次に、Application Load Balancer リスナーの [証明書] タブを選択します。
12. [証明書の追加] をクリックします。ここでは、証明書をリスナーにインポートできます。
13. Route 53 ホストゾーンで、[レコードの作成] をクリックして、サービスへのルーティングレコードを作成します。
14. シンプルルーティングレコードを選択します。
15. レコード名を追加します。
16. レコードタイプを選択します。
17. トラフィックを Application Load Balancer と Classic Load Balancer へのエイリアスにルーティングします。
18. リージョンを選択します。
19. Express Mode サービスが使用するロードバランサーを選択します。

数分かけて伝播した後、サービスはカスタムドメインでトラフィックを受信し始めます。Application Load Balancer に多くの証明書を追加する場合、または Application Load Balancer からの多くの Express Mode サービスを処理する場合は、Application Load Balancer の証明書のサービス制限を引き上げることを検討してください。

Amazon ECSの タスク定義

タスク定義はアプリケーションのブループリントです。これは、アプリケーションを形成するパラメータと 1 つ以上のコンテナを記述する JSON 形式のテキストファイルです。

以下に示したのは、タスク定義の中で指定できるパラメータの一部です。

- タスクをホストするインフラストラクチャを決定するキャパシティ
- タスクの各コンテナで使用する Docker イメージ
- 各タスクで、またはタスク内の各コンテナで使用する CPU とメモリの量
- メモリと CPU の要件
- タスクが実行されるコンテナのオペレーションシステム
- タスクのコンテナで使用する Docker ネットワーキングモード
- タスクで使用するログ記録設定
- コンテナが終了または失敗した場合にタスクを実行し続けるかどうか
- コンテナの開始時に実行するコマンド
- タスク内でコンテナが使用するデータボリューム
- タスクで使用される IAMロール

タスク定義パラメータの完全なリストについては、「Fargate での Amazon ECS タスク定義パラメータ」を参照してください。

タスク定義を作成したら、タスク定義をタスクまたはサービスとして実行できます。

- タスクはクラスター内のタスク定義のインスタンス化です。Amazon ECS でアプリケーションのタスク定義を作成後、クラスターで実行するタスクの数を指定できます。
- Amazon ECS サービスは、Amazon ECS クラスターで必要な数のタスクを同時に実行して維持します。仕組みとしては、タスクがいずれかの理由で失敗または停止した場合に、Amazon ECS サービススケジューラがタスク定義に基づいて別のインスタンスを起動することによって動作します。これは、それを置き換え、サービス内の必要な数のタスクを維持するために行われます。

トピック

- Amazon ECS タスク定義の状態
- Amazon ECS 用のアプリケーションの構築

- コンソールを使用した Amazon ECS タスク定義の作成
- Amazon Q Developer を使用して Amazon ECS コンソールでタスク定義のレコメンデーションを生成する
- コンソールを使用した Amazon ECS タスク定義の更新
- 新しいコンソールを使用した Amazon ECS タスク定義リビジョンの登録解除
- コンソールを使用した Amazon ECS タスク定義リビジョンの削除
- Amazon ECS タスク定義のユースケース
- Amazon ECS マネージドインスタンスの Amazon ECS タスク定義パラメータ
- Fargate での Amazon ECS タスク定義パラメータ
- Amazon EC2 における Amazon ECS タスク定義パラメータ
- Amazon ECS タスク定義テンプレート
- Amazon ECS のタスク定義の例

Amazon ECS タスク定義の状態

タスク定義を作成、登録解除、または削除すると、タスク定義の状態が変わります。タスク定義の状態は、コンソール上で、または `DescribeTaskDefinition` を使用して確認できます。

タスク定義は、以下の状態を取ることがあります。

ACTIVE

Amazon ECS に登録された後のタスク定義は `ACTIVE` の状態になります。`ACTIVE` 状態にあるタスク定義では、タスクを実行することやサービスを作成することができます。

INACTIVE

タスク定義の登録を解除すると、そのタスク定義の状態は `ACTIVE` から `INACTIVE` に遷移します。`DescribeTaskDefinition` を呼び出すと、`INACTIVE` 状態のタスク定義を取得できます。`INACTIVE` 状態のタスク定義を使用して、新しいタスクを実行することや新しいサービスを作成することはできません。既存のサービスやタスクには影響を与えません。

DELETE_IN_PROGRESS

タスク定義の削除をリクエストすると、そのタスク定義の状態が `INACTIVE` から `DELETE_IN_PROGRESS` に遷移します。タスク定義の状態が `DELETE_IN_PROGRESS` に遷移した後、Amazon ECS は、そのタスク定義がアクティブなタスクやデプロイによって参照されていな

いことを周期的に確認した上で、タスク定義を完全に削除します。DELETE_IN_PROGRESS 状態のタスク定義を使用して、新しいタスクを実行することや新しいサービスを作成することはできません。タスク定義の削除は、既存のタスクやサービスに影響を与えることなく、任意のタイミングでリクエストできます。

DELETE_IN_PROGRESS 状態にあるタスク定義はコンソールに表示できます。また、DescribeTaskDefinition を呼び出すと、そのタスク定義を取得できます。

すべての INACTIVE タスク定義リビジョンを削除すると、タスク定義名はコンソールで表示されず、API でも返されません。タスク定義リビジョンが DELETE_IN_PROGRESS 状態にある場合、タスク定義名はコンソールに表示され、API で返されます。タスク定義名は Amazon ECS によって保持され、次回その名前を使用してタスク定義を作成するときにリビジョンがインクリメントされます。

タスク定義の管理に AWS Config を使用してしている場合は、すべてのタスク定義の登録について AWS Config による課金が行われます。登録解除については、ACTIVE の状態にある最新のタスク定義の解除に対してのみ課金されます。タスク定義の削除には料金はかかりません。料金の詳細については、「AWS Config 料金表」を参照してください。

削除をブロックできる Amazon ECS リソース

タスク定義リビジョンに依存する Amazon ECS リソースがある場合、タスク定義の削除リクエストは完了しません。次のリソースが原因で、タスク定義が削除されない場合があります。

- Amazon ECS スタンドアロンタスク – タスクを正常に動作させるには、タスク定義が必要です。
- Amazon ECS サービスタスク - タスクを正常に動作させるには、タスク定義が必要です。
- Amazon ECS サービスのデプロイとタスクセット - Amazon ECS のデプロイまたはタスクセットのスケーリングイベントが開始される場合は、タスク定義が必要です。

タスク定義が DELETE_IN_PROGRESS の状態のままである場合は、コンソールまたは AWS CLI を使用して、タスク定義の削除をブロックしているリソースを特定し、停止できます。

ブロックされたリソースが削除された後のタスク定義の削除

タスク定義の削除をブロックするリソースを削除すると、次のルールが適用されます。

- Amazon ECS タスク - タスク定義の削除は、タスクが停止されてから完了するまでに最大 1 時間かかる場合があります。

- Amazon ECS サービスのデプロイとタスクセット - タスク定義の削除は、デプロイまたはタスクセットが削除されてから完了するまでに最大 24 時間かかる場合があります。

Amazon ECS 用のアプリケーションの構築

アプリケーションのタスク定義を作成して、アプリケーションを構築します。タスク定義には、次のようなアプリケーションに関する情報を定義するパラメータが含まれます。

- タスクをホストするインフラストラクチャを決定するキャパシティ。

  EC2 キャパシティプロバイダーを使用する場合は、インスタンスタイプも選択します。Amazon ECS マネージドインスタンスのキャパシティプロバイダーを使用すると、Amazon ECS がコンピューティングキャパシティを管理するためのインスタンス要件を提供できます。GPU などの一部のインスタンスタイプでは、特定のパラメータを設定する必要があります。詳細については、「Amazon ECS タスク定義のユースケース」を参照してください。
- コンテナイメージには、アプリケーションコードと、アプリケーションコードの実行に必要なすべての依存関係が保持されます。
- タスク内のコンテナで使用するネットワーキングモード。

  ネットワークモードにより、タスクがネットワーク上で通信する方法が決定します。

  EC2 インスタンスおよび Amazon ECS マネージドインスタンスで実行するタスクには、オプションが複数存在します。その中でも awsvpc ネットワークモードの使用を推奨します。awsvpc ネットワークモードでは、VPC 内でのアプリケーション同士や他のサービスとの通信方法に対してより詳細な制御を可能にすることで、コンテナネットワークが簡素化されます。

  Fargate で実行されるタスクについては、awsvpc ネットワークモードを使用する必要があります。
- タスクで使用するログ記録設定。
- タスク内のコンテナで使用するデータボリューム。

タスク定義パラメータの完全なリストについては、「Fargate での Amazon ECS タスク定義パラメータ」を参照してください。

タスク定義を作成する際には、次のガイドラインに従ってください。

- 各タスク定義ファミリーは 1 つのビジネス目的にのみ使用してください。

複数の種類のアプリケーションコンテナをまとめて同じタスク定義にグループ化する場合、これらのコンテナを個別にスケールすることはできません。たとえば、ウェブサイトと API では通常、異なるスケーリングパターンが必要となります。トラフィックが増加すると、API コンテナとは異なる数のウェブコンテナが必要になる場合があります。これらの 2 つのコンテナが同じタスク定義にデプロイされる場合、各タスクは同じ数のウェブコンテナと API コンテナを実行します。

- 各アプリケーションのバージョンを、タスク定義ファミリー内のタスク定義リビジョンと一致させます。

  タスク定義ファミリー内では、各タスク定義のリビジョンは、特定のコンテナイメージのその時点での設定のスナップショットを表します。これは、コンテナがアプリケーションコードの特定のバージョンを実行するために必要なすべての構成要素のスナップショットであることと似ています。

  アプリケーションコードのバージョン、コンテナイメージタグ、およびタスク定義のリビジョンの間に、1 対 1 のマッピングを作成します。一般的なリリースプロセスには、git commit SHA でタグ付けされたコンテナイメージに変換される git commit が含まれます。その後、そのコンテナイメージタグは、独自の Amazon ECS タスク定義リビジョンを取得します。最後に、Amazon ECS サービスを更新し、新しいタスク定義のリビジョンをデプロイします。
- タスク定義ファミリーごとに異なる IAM ロールを使用します。

  各タスク定義を独自の IAM ロールで定義します。このプラクティスを実装するのに併せて、各ビジネスコンポーネントにそれ独自のタスク定義ファミリーを提供します。これらのベストプラクティスを両方実装することで、各サービスが AWS アカウント内のリソースにどの程度アクセスできるかを制限できます。例えば、パスワードデータベースに接続するためのアクセス権を認証サービスに付与できます。同時に、注文サービスのみがクレジットカードの支払情報にアクセスできることを確認できます。

Amazon ECS タスクサイズのベストプラクティス

コンテナとタスクのサイズ決定は、スケーリングとキャパシティプランニングに不可欠な要素です。Amazon ECS では、CPU とメモリがキャパシティーに使用される 2 つのリソースメトリクスです。CPU 量 は、フル vCPU の 1024 分の 1 の単位で測定されます (1024 ユニットは 1 つの vCPU 全体に相当します)。メモリはメビバイト単位で測定されます。タスク定義では、リソースの予約と制限を設定できます。

予約を設定すると、タスクに最低限必要な量のリソースが設定されます。タスクは少なくとも要求された最低限の量のリソースを受け取ります。アプリケーションは、宣言された予約量よりも多く

の CPU またはメモリを使用できる可能性があります。ただし、これには宣言した制限が適用されます。予約量を超えるリソースを使用することをバーストと呼びます。Amazon ECS では、予約量が保証されます。たとえば、Amazon EC2 インスタンスを使用してキャパシティを提供する場合、Amazon ECS は予約量を満足できないインスタンスにタスクを課しません。

制限は、コンテナまたはタスクが使用できる CPU ユニットまたはメモリの最大量です。この制限を超える CPU 量 を使用しようとすると、スロットリングが発生します。それ以上メモリを使おうとすると、コンテナは停止します。

これらの値の決定は簡単ではありません。これは、各アプリケーションに最も適した値は、そのアプリケーションのリソース要件に大きく依存するためです。適切にリソース要件を計画し、アプリケーション要件を正確に把握するには、アプリケーションの負荷テストを行うことが重要です。

ステートレスアプリケーション

ロードバランサーの背後にあるアプリケーションなど、水平スケーリングするタイプのステートレスアプリケーションでは、まずそのアプリケーションがリクエストを処理するときに消費するメモリの量を確認することをお勧めします。これは、`ps` や `top` などの従来のツール、あるいは CloudWatch Container Insights などのモニタリングソリューションを使用して実施できます。

CPU 予約を決定する際には、ビジネス要件を満たすにはアプリケーションをどのようにスケーリングすべきかを考慮すること。256 CPU ユニット (つまりフル vCPU の 4 分の 1) などの小さめの CPU 予約を行うことで、コストを最小限に抑えながらきめ細かくスケールアウトできます。ただしその場合、需要が急増した時にスケーリング速度が追い付かない可能性があります。CPU 予約量を増やすとより迅速にスケールイン/スケールアウトできるため、需要の急増にもスムーズに対応できます。ただし、CPU の予約量が多いほどコストが高くなりますのでご注意ください。

その他のアプリケーション

シングルトンワーカーやデータベースサーバーなど、水平スケーリングしないタイプのアプリケーションでは、利用可能な容量とコストが最も重要な考慮事項となります。サービスレベル目標達成のためのトラフィック処理にはどの程度のメモリと CPU 量が必要かを負荷テストで確認した上でメモリおよび CPU 量を決定する必要があります。Amazon ECS は、アプリケーションが十分な容量のあるホストに配置されるよう保証します。

Amazon ECS マネージドインスタンスの Amazon ECS タスクネットワーキング

Amazon ECS マネージドインスタンスが実行する Amazon ECS タスクのネットワーク動作は、タスク定義で指定されたネットワークモードに左右されます。タスク定義からネットワークモードを指定

してください。タスク定義でネットワークモードを指定しないと、Amazon ECS マネージドインスタンスでタスクを実行することはできません。Amazon ECS マネージドインスタンスは以下のネットワークモードに対応しており、ワークロードを Amazon EC2 上の Fargate または Amazon ECS から移行できる下位互換性を提供します。

ネットワークモード	説明
awsvpc	各タスクには、独自の Elastic Network Interface (ENI) とプライベート IPv4 アドレスが割り当てられます。これにより、Amazon EC2 インスタンスと同一のネットワークプロパティを提供し、従来の Fargate タスクとの互換性を実現します。ENI トランキングを使用してタスク密度を高めます。
host	タスクはホストのネットワーク名前空間を直接共有します。コンテナネットワークは、基盤となるホストイ

ネットワークモード	説明	
	ンスタンスに紐づけられます。	

IPv6-only モードで VPC を使用する

IPv6-only 設定では、Amazon ECS タスクは IPv6 経由でのみ通信します。VPC およびサブネットを IPv6-only モードに設定するには、IPv6 CIDR ブロックを VPC に追加し、IPv6 CIDR ブロックのみを含むサブネットを作成してください。詳しくは「Amazon VPC ユーザーガイド」の「VPC の IPv6 サポートを追加する」および「サブネットを作成する」を参照してください。また、IPv6 ターゲットでルートテーブルを更新し、IPv6 ルールでセキュリティグループを設定する必要があります。詳しくは「Amazon VPC ユーザーガイド」の「ルートテーブルを設定する」および「セキュリティグループルールを設定する」を参照してください。

以下の考慮事項に注意してください。

- Amazon ECS サービスを IPv6-only サブネットを使用する設定に直接更新するか、別の IPv6-only サービスを作成し、Amazon ECS ブルー/グリーンデプロイを用いて新サービスへのトラフィック移行を行うことで、IPv4-only 設定の更新および IPv6-only 設定のデュアルスタックを有効にすることが可能です。Amazon ECS におけるブルー/グリーンデプロイについて、詳しくは「Amazon ECS ブルー/グリーンデプロイ」を参照してください。
- IPv6-only Amazon ECS サービスは、IPv6 ターゲットグループを保有するデュアルスタックロードバランサーを使用する必要があります。Application Load Balancer または Network Load Balancer の背後で稼働する既存の Amazon ECS サービスを移行する場合は、新しいデュアルスタックロードバランサーを作成して旧ロードバランサーからトラフィックを切り替えるか、既存のロードバランサーの IP アドレスタイプを更新します。

  Network Load Balancer について、詳しくは「Network Load Balancer ユーザーガイド」の「Network Load Balancer を作成する」および「Network Load Balancer の IP アドレスタイプを更新する」を参照してください。Application Load Balancer について、詳しくは「Application Load Balancer ユーザーガイド」の「Application Load Balancer を作成する」および「Application Load Balancer の IP アドレスタイプを更新する」を参照してください。
- IPv4-only エンドポイントと交信する IPv6-only 設定の Amazon ECS タスクにおいては、IPv6 から IPv4 へのネットワークアドレス変換に DNS64 と NAT64 を設定できます。詳細については、Amazon VPC ユーザーガイドの DNS64 および NAT64 を参照してください。

- IPv6-only 設定の Amazon ECS ワークロードでは、Amazon ECR からイメージを取得する際に Amazon ECR デュアルスタックイメージ URI エンドポイントを使用する必要があります。詳しくは「Amazon Elastic Container Registry ユーザーガイド」の「Getting started with making requests over IPv6」を参照してください。

  > **Note**
  >
  > Amazon ECR は、IPv6-only 設定のタスクが使用するデュアルスタックインターフェイス VPC エンドポイントには対応していません。詳しくは「Amazon Elastic Container Registry ユーザーガイド」の「Getting started with making requests over IPv6」を参照してください。

- Amazon ECS Exec は IPv6-only 設定には対応していません。

Amazon ECS マネージドインスタンスタスクにネットワークインターフェイスを割り当てる

Amazon ECS マネージドインスタンスで awsvpc ネットワークモードを利用することで、アプリケーション間の通信および VPC 内のその他サービスとの通信における管理能力が向上し、よりシンプルなコンテナネットワークが実現します。awsvpc ネットワークモードによりコンテナのセキュリティも強化されます。これは、セキュリティグループやネットワークモニタリングツールを、タスク内でより詳細なレベルで利用できるためです。

デフォルトでは、インスタンスタイプがトランキングをサポートしている場合、すべての Amazon ECS マネージドインスタンスには、起動時にトランク Elastic Network Interface (ENI) がプライマリ ENI としてアタッチされます。ENI トランキングをサポートするインスタンスタイプについて、詳しくは「Amazon ECS コンテナネットワークインターフェイスの増加でサポートされるインスタンス」を参照してください。

> **Note**
>
> 選択したインスタンスタイプがトランク ENI をサポートしていない場合、インスタンスは通常の ENI で起動されます。

インスタンスで実行される各タスクは、プライマリプライベート IP アドレスを持つ、トランク ENI にアタッチされた独自の ENI を受け取ります。VPC がデュアルスタックモード対応に設定されており、かつ IPv6 CIDR ブロックを備えたサブネットを利用する場合は、ENI に対しても IPv6 アドレ

スが割り当てられます。パブリックサブネットを使用する場合、サブネットの IPv4 パブリックアドレス指定を有効にすることで、オプションでパブリック IP アドレスを Amazon ECS マネージドインスタンスのプライマリ ENI に割り当てることができます。詳しくは「Amazon VPC ユーザーガイド」の「サブネットの IP アドレス指定属性を変更する」を参照してください。タスクには、一度に 1 つの ENI しか関連付けられません。

また、同じタスクに属するコンテナでも、`localhost` インターフェイス経由での通信が可能になります。VPC とサブネットについて、詳しくは「Amazon VPC ユーザーガイド」の「Amazon VPC の仕組み」を参照してください。

以下のオペレーションを行うには、インスタンスにアタッチされたプライマリ ENI が必要です。

- イメージのダウンロード – コンテナイメージは、プライマリ ENI を介して Amazon ECR からダウンロードされます。
- シークレットの取得 – Secrets Manager のシークレットおよびその他の認証情報は、プライマリ ENI を介して取得されます。
- ログのアップロード – ログはプライマリ ENI を介して CloudWatch にアップロードされます。
- 環境ファイルのダウンロード – 環境ファイルはプライマリ ENI を介してダウンロードされます。

アプリケーショントラフィックはタスク ENI を経由します。

各タスクにはそれぞれ独自の ENI が提供されるため、VPC フローログなどのネットワーキング機能を使用して、タスクとの間で送受信されるトラフィックをモニタリングできるようになります。詳細については、「Amazon VPC ユーザーガイド」の「VPC フローログを使用した IP トラフィックのログ記録」を参照してください。

AWS PrivateLink を活用することもできます。VPC インターフェースエンドポイントを設定することで、プライベート IP アドレスを通じて Amazon ECS の API にアクセスができます。AWS PrivateLink は、VPC と Amazon ECS 間のすべてのネットワークトラフィックを Amazon ネットワークに制限します。インターネットゲートウェイ、NAT デバイス、または仮想プライベートゲートウェイは必要ありません。詳細は「Amazon ECS インターフェイス VPC エンドポイント (AWS PrivateLink)」を参照してください。

`awsvpc` ネットワークモードでは、Amazon VPC トラフィックミラーリングを活用して、トランク ENI がアタッチされていないインスタンスタイプを使用する場合に、ネットワークトラフィックのセキュリティとモニタリングを行うこともできます。詳しくは「Amazon VPC トラフィックミラーリングガイド」の「What is Traffic Mirroring?」を参照してください。

awsvpc モードに関する考慮事項

- タスクには、Amazon ECS サービスに紐づいた ENI 管理用ロールが必要です。このロールは、クラスターまたはサービスの作成時に自動で生成されます。
- タスク ENI は Amazon ECS によって管理され、手動で削除または変更することはできません。
- スタンドアロンタスク (RunTask) の実行時、またはサービス (CreateService/UpdateService) の作成または更新時の、assignPublicIp を使用したタスク ENI へのパブリック IP アドレス割り当てはサポート対象外です。
- タスクレベルで awsvpc ネットワークを設定する場合は、Amazon ECS マネージドインスタンスのキャパシティプロバイダーの起動テンプレートの一部として指定したのと同じ VPC を使用する必要があります。起動テンプレートで指定されたサブネットやセキュリティグループとは異なるサブネットやセキュリティグループを使用できます。
- awsvpc ネットワークモードタスクの場合は、ロードバランサーターゲットグループを設定するときに ip ターゲットタイプを使用します。Amazon ECS は、サポートされているネットワークモードのターゲットグループ登録を自動的に管理します。

デュアルスタックモードでの VPC の使用

デュアルスタックモードで VPC を使用する場合、タスクは IPv4 または IPv6、あるいはその両方を経由して通信できます。IPv4 と IPv6 アドレスは、互いに独立しています。そのため、VPC 内で IPv4 と IPv6 のルーティングとセキュリティを別々に設定する必要があります。VPC をデュアルスタックモード用に設定する方法については、「Amazon VPC ユーザーガイド」の「既存の VPC を IPv4 から IPv6 に移行する」を参照してください。

VPC でインターネットゲートウェイまたは送信専用インターネットゲートウェイを設定した場合は、その VPC をデュアルスタックモードで使用できます。この設定により、IPv6 アドレスが割り当てられたタスクは、インターネットゲートウェイまたは送信専用インターネットゲートウェイを介して、インターネットへのアクセスが可能になります。NAT ゲートウェイはオプションです。詳細については、Amazon VPC ユーザーガイドの「インターネットゲートウェイ」および「Egress-only インターネットゲートウェイ」を参照してください。

以下の条件が満たされた場合、Amazon ECS タスクには IPv6 アドレスが割り当てられます。

- タスクをホストする Amazon ECS マネージドインスタンスのインスタンスは、バージョン 1.45.0 以降のコンテナエージェントを使用します。インスタンスが使用しているエージェントのバージョンを確認し、必要に応じて更新する方法については、「Amazon ECS コンテナエージェントをアップデートする」を参照してください。

- dualStackIPv6 アカウント設定が有効になります。詳細については、「アカウント設定による Amazon ECS 機能へのアクセス」を参照してください。
- タスクは awsvpc ネットワークモードを使用しています。
- VPC とサブネットは、IPv6 に対して有効になっています。この設定には、指定されたサブネットで作成されたネットワークインターフェイスが含まれます。VPC をデュアルスタックモードに設定する方法について、詳しくは「Amazon VPC ユーザーガイド」の「IPv6 へ移行する」および「サブネットの IP アドレス指定属性を変更する」を参照してください。

ホストネットワークモード

host モードでは、タスクはホストのネットワーク名前空間を直接共有します。コンテナのネットワーク設定は、Amazon ECS マネージドインスタンスのキャパシティプロバイダーを作成するときに networkConfiguration パラメータを使用して指定する、基盤となる Amazon ECS マネージドインスタンスのホストインスタンスに関連付けられます。

このネットワークモードを使用することには重大な欠点があります。各ホストで 1 つのタスクを複数インスタンス化することはできません。これは、Amazon EC2 インスタンスの必要なポートにバインドできるのは最初のタスクだけだからです。また、host ネットワークモードを使用している場合、コンテナポートを再マップする方法はありません。例えば、アプリケーションが特定のポート番号でリッスンする必要がある場合、ポート番号を直接再マップすることはできません。代わりに、アプリケーション構成を変更してポートの競合を管理する必要があります。

host ネットワークモードを使用する際には、セキュリティ上の問題もあります。このモードでは、コンテナがホストになりすますことができ、コンテナはホスト上のプライベートループバックネットワークサービスに接続できます。

ホストモードは、ホストネットワークへの直接アクセスが必要な場合、またはホストレベルのネットワークアクセスを必要とするアプリケーションを移行する場合にのみ使用します。

EC2 の Amazon ECS タスクネットワークオプション

Amazon EC2 インスタンスでホストされる Amazon ECS タスクのネットワーク動作は、タスク定義で定義されているネットワークモードに左右されます。Amazon ECS では、別のネットワークモードを使用する特別の必要性がある場合を除き、awsvpc ネットワークモードの使用を推奨します。

使用可能なネットワークモードは次のとおりです。

ネットワークモード	EC2 の Linux コンテナ	EC2 の Windows コンテナ	[Description] (説明)
`awsvpc`	可能	はい	このタスクには、独自の Elastic Network Interface (ENI) とプライマリプライベート IPv4 または IPv6 アドレスが割り当てられます。これにより、タスクに Amazon EC2 インスタンスと同じネットワークプロパティが与えられます。
`bridge`	はい	いいえ	タスクは、そのタスクをホストする各 Amazon EC2 インスタンス内で実行される、Linux 上の Docker の組み込み仮想ネットワークを使用します。Linux の組み込み仮想ネットワークでは、`bridge` Docker ネットワークドライバーが使用されます。これは、タスク定義でネットワークモードが指定されていない場合の、Linux のデフォルトのネットワークモードです。
`host`	はい	いいえ	タスクは、そのタスクをホストする Amazon EC2 インスタンスの ENI にコンテナポートを直接マッピングすることで Docker の組み込み仮想ネットワークをバイパスする、ホストのネットワークを使用します。ダイナミックポートマッピングは、このネットワークモードでは使用できません。このモードを使用するタスク定義内のコンテナには、特定の `hostPort` 番号を指定する必要があります。ホストのポート番号は、複数のタスクで使用できません。その結果として、1 つの Amazon EC2 インスタンスで同じタスク定義のタスクを複数実行することはできません。
`none`	はい	いいえ	このタスクには外部ネットワーク接続がありません。

ネットワークモード	EC2 の Linux コンテナ	EC2 の Windows コンテナ	[Description] (説明)
`default`	いいえ	はい	タスクは、そのタスクをホストする各 Amazon EC2 インスタンス内で実行される、Windows 上の Docker の組み込み仮想ネットワークを使用します。Windows の組み込み仮想ネットワークは `nat` Docker ネットワークドライバーを使用します。これは、タスク定義でネットワークモードが指定されていない場合の、Windows のデフォルトのネットワークモードです。

Linux の Docker ネットワークについて、詳しくは「Docker ドキュメント」の「ネットワーク機能の概要」を参照してください。

Windows の Docker ネットワークについて詳しくは、Microsoft の Containers on Windows ドキュメント内、「Windows コンテナネットワーク」を参照してください。

IPv6-only モードで VPC を使用する

IPv6-only 設定では、Amazon ECS タスクは IPv6 経由でのみ通信します。IPv6-only 設定の VPC とサブネットを設定するには、IPv6 CIDR ブロックを VPC に追加し、IPv6 CIDR ブロックのみを含む新しいサブネットを作成する必要があります。詳しくは「Amazon VPC ユーザーガイド」の「VPC の IPv6 サポートを追加する」および「サブネットを作成する」を参照してください。

また、IPv6 ターゲットでルートテーブルを更新し、IPv6 ルールでセキュリティグループを設定する必要があります。詳しくは「Amazon VPC ユーザーガイド」の「ルートテーブルを設定する」および「セキュリティグループルールを設定する」を参照してください。

以下の考慮事項に注意してください。

- Amazon ECS サービスを IPv6-only サブネットを使用する設定に直接更新するか、別の IPv6-only サービスを作成し、Amazon ECS ブルー/グリーンデプロイを用いて新サービスへのトラフィック移行を行うことで、IPv4-only 設定の更新および IPv6-only 設定のデュアルスタックを有効にすることが可能です。Amazon ECS におけるブルー/グリーンデプロイについて、詳しくは「Amazon ECS ブルー/グリーンデプロイ」を参照してください。

- IPv6-only Amazon ECS サービスは、IPv6 ターゲットグループを保有するデュアルスタックロードバランサーを使用する必要があります。Application Load Balancer または Network Load Balancer の背後で稼働する既存の Amazon ECS サービスを移行する場合は、新しいデュアルスタックロードバランサーを作成して旧ロードバランサーからトラフィックを切り替えるか、既存のロードバランサーの IP アドレスタイプを更新します。

  Network Load Balancer について、詳しくは「Network Load Balancer ユーザーガイド」の「Network Load Balancer を作成する」および「Network Load Balancer の IP アドレスタイプを更新する」を参照してください。Application Load Balancer について、詳しくは「Application Load Balancer ユーザーガイド」の「Application Load Balancer を作成する」および「Application Load Balancer の IP アドレスタイプを更新する」を参照してください。

- IPv6-only 設定は Windows ではサポートされていません。IPv6-only 設定でタスクを実行するには、Amazon ECS 最適化 Linux AMI を使用する必要があります。Amazon ECS 最適化 AMI について、詳しくは「Amazon ECS に最適化された Linux AMI」を参照してください。

- IPv6-only 設定でタスクを実行するコンテナインスタンスを起動するときは、`--enable-primary-ipv6` EC2 パラメータを使用してインスタンスのプライマリ IPv6 アドレスを設定する必要があります。

  > **Note**
  >
  > プライマリ IPv6 アドレスがない場合、ホストまたはブリッジネットワークモードのコンテナインスタンスで実行されているタスクは、ロードバランサーまたは AWS Cloud Map に登録されません。

  Amazon EC2 インスタンスを実行する `--enable-primary-ipv6` について、詳しくは「AWS CLI Command Reference」の「run-instances」を参照してください。

  AWS マネジメントコンソールを利用したコンテナインスタンス起動について、詳しくは「Amazon ECS Linux コンテナインスタンスの起動」を参照してください。

- デフォルトでは、Amazon ECS コンテナエージェントは、インスタンスのデフォルト IPv4 および IPv6 ルートを確認することで、そのコンテナインスタンスが IPv6-only 設定に対応しているかどうかを検出します。この動作を上書きするには、インスタンスの `/etc/ecs/ecs.config` ファイルで `ECS_INSTANCE_IP_COMPATIBILITY` パラメータを `ipv4` または `ipv6` に設定してください。

- タスクはコンテナエージェントのバージョン 1.99.1 以降を実行します。インスタンスが使用するエージェントのバージョンを確認し、必要に応じて更新する方法について、詳しくは「Amazon ECS コンテナエージェントをアップデートする」を参照してください。
- IPv4-only エンドポイントと交信する IPv6-only 設定の Amazon ECS タスクにおいては、IPv6 から IPv4 へのネットワークアドレス変換に DNS64 と NAT64 を設定できます。詳細については、Amazon VPC ユーザーガイドの DNS64 および NAT64 を参照してください。
- IPv6-only 設定の Amazon ECS ワークロードでは、Amazon ECR からイメージを取得する際に Amazon ECR デュアルスタックイメージ URI エンドポイントを使用する必要があります。詳しくは「Amazon Elastic Container Registry ユーザーガイド」の「Getting started with making requests over IPv6」を参照してください。

  > **Note**
  >
  > Amazon ECR は、IPv6-only 設定のタスクが使用するデュアルスタックインターフェイス VPC エンドポイントには対応していません。詳しくは「Amazon Elastic Container Registry ユーザーガイド」の「Getting started with making requests over IPv6」を参照してください。

- Amazon ECS Exec は IPv6-only 設定には対応していません。

AWS リージョンAmazon ECS の IPv6-only モードをサポートしている

IPv6-only 設定でのタスクは、Amazon ECS が利用可能な以下の AWS リージョンで実行できます。

- 米国東部(オハイオ)
- 米国東部 (バージニア北部)
- 米国西部 (北カリフォルニア)
- 米国西部 (オレゴン)
- アフリカ (ケープタウン)
- アジアパシフィック (香港)
- アジアパシフィック (ハイデラバード)
- アジアパシフィック (ジャカルタ)
- アジアパシフィック (メルボルン)
- アジアパシフィック (ムンバイ)
- アジアパシフィック (大阪)

- アジアパシフィック (ソウル)
- アジアパシフィック (シンガポール)
- アジアパシフィック (シドニー)
- アジアパシフィック (東京)
- カナダ (中部)
- カナダ西部 (カルガリー)
- 中国 (北京)
- 中国 (寧夏)
- 欧州 (フランクフルト)
- 欧州 (ロンドン)
- 欧州 (ミラノ)
- 欧州 (パリ)
- 欧州 (スペイン)
- イスラエル (テルアビブ)
- 中東 (バーレーン)
- 中東 (アラブ首長国連邦)
- 南米 (サンパウロ)
- AWS GovCloud (米国東部)
- AWS GovCloud (米国西部)

Amazon ECS タスクにネットワークインターフェイスを割り当てる

awsvpc ネットワークモードで利用できるタスクネットワーキング機能により、Amazon EC2 インスタンスと同じネットワーキングプロパティが Amazon ECS タスクに提供されます。awsvpc ネットワークモードを使用すると、コンテナネットワークが簡素化されます。また、アプリケーション間およびそのアプリケーションと VPC 内の他のサービスとの相互通信をより強力にコントロールできます。awsvpc ネットワークモードによりコンテナのセキュリティも強化されます。これは、セキュリティグループやネットワークモニタリングツールを、タスク内でより詳細なレベルで利用できるためです。VPC フローログなどの各種 Amazon EC2 ネットワーキング機能により、タスクが送受信するトラフィックをモニタリングできます。さらに、同じタスクに属するコンテナが、localhost インターフェイス経由で通信できるようになります。

タスク向け Elastic Network Interface (ENI) は、Amazon ECS のフルマネージド型機能です。Amazon ECS により ENI が作成され、指定されたセキュリティグループが関連付けられているホスト Amazon EC2 インスタンスにアタッチされます。タスクは、Amazon EC2 インスタンスがプライマリネットワークインターフェイスで実行する場合と同じ方法で、ENI を介してネットワークトラフィックを送受信します。各タスクの ENI には、デフォルトでプライベート IPv4 アドレスが割り当てられます。VPC がデュアルスタックモードに対応していて、IPv6 CIDR ブロックを備えたサブネットを使用する場合、タスクの ENI も IPv6 アドレスを受け取ります。各タスクは、ENI を 1 つだけ持つことができます。

これらの ENI は、アカウントの Amazon EC2 コンソールに表示されます。アカウント側では ENI をデタッチしたり変更したりすることはできません。これは、実行中のタスクに関連付けられている ENI が誤って削除されないようにするためです。タスクの ENI のアタッチに関する情報は、Amazon ECS コンソールか、DescribeTasks API オペレーションで確認できます。タスクが停止した場合やサービスがスケールダウンした場合は、ENI がデタッチされて削除されます。

ENI 密度を上げる必要がある場合は、`awsvpcTrunking` アカウント設定を使用してください。また Amazon ECS は、コンテナインスタンスのトランクネットワークインターフェイスを作成およびアタッチします。トランクネットワークは Amazon ECS によって完全に管理されます。コンテナインスタンスを Amazon ECS クラスターから削除または登録解除するときに、トランク ENI は削除されます。`awsvpcTrunking` アカウント設定の詳細については、「前提条件」を参照してください。

タスク定義の `networkMode` パラメータで `awsvpc` を指定します。詳細については、「ネットワークモード」を参照してください。

次に、タスクの実行時またはサービスの作成時に、タスクを配置する 1 つ以上のサブネットと、ENI にアタッチする 1 つ以上のセキュリティグループを含む `networkConfiguration` パラメータを使用します。詳細については、「ネットワーク設定」を参照してください。タスクは、これらのサブネットとして、同じアベイラビリティーゾーン内の適切な Amazon EC2 インスタンスに配置されます。また、指定されたセキュリティグループが、タスク用にプロビジョニングされた ENI に関連付けられます。

Linux に関する考慮事項

Linux オペレーティングシステムを使用する場合は、以下の点を考慮してください。

- `awsvpc` モードで p5.48xlarge インスタンスを使用する場合、インスタンスで複数のタスクを実行することはできません。
- `awsvpc` ネットワークモード使用するタスクとサービスには、Amazon ECS サービスにリンクされたロールが必要です。このロールにより、ユーザーに代わってその他の AWS サービスを呼び

出す許可を、Amazon ECS に付与できるようになります。このロールは、クラスターの作成、またはサービスの作成や更新を AWS マネジメントコンソール から行う際に、自動的に作成されます。詳細については、「Amazon ECS のサービスリンクロールの使用」を参照してください。サービスにリンクされたロールは、次の AWS CLI コマンドを使用して作成することもできます。

```
aws iam create-service-linked-role --aws-service-name ecs.amazonaws.com
```

- Amazon EC2 Linux インスタンスでは、awsvpc ネットワークモードを使用するタスクを実行するには、コンテナエージェントのバージョン 1.15.0 以降が必要です。Amazon ECS 最適化 AMI を使用している場合、インスタンスには、少なくとも ecs-init パッケージの 1.15.0-4 バージョン以降も必要です。
- enableDnsHostnames と enableDnsSupport オプションの両方が VPC で有効になっている場合、Amazon ECS は Amazon が提供する (内部) DNS ホスト名でタスクのホスト名を設定します。これらのオプションが有効でない場合、タスクの DNS ホスト名にはランダムな名前が付けられます。VPC の DNS 設定について、詳細は「Amazon VPC ユーザーガイド」の「VPC で DNS を使用する」を参照してください。
- awsvpc ネットワークモードを使用する Amazon ECS タスクには、それぞれ独自の Elastic Network Interface (ENI) が提供されます。この ENI は、ENI をホストする Amazon EC2 インスタンスにアタッチされています。Amazon EC2 Linux インスタンスにアタッチできるネットワークインターフェイスの数には、デフォルトのクォータが設定されます。そのクォータに対して、プライマリネットワークインターフェイスは 1 個としてカウントされます。例えば、c5.large インスタンスのデフォルトでは、アタッチ可能な ENI の数は最大 3 個までです。このインスタンスのプライマリネットワークインターフェイスも、1 個としてカウントされます。さらに 2 つの ENI をインスタンスにアタッチできます。awsvpc ネットワークモードを使用する各タスクには ENI が必要なため、通常このインスタンスタイプでは、このようなタスクを 2 つのみ実行できます。各インスタンスタイプのデフォルトの ENI 制限については、「Amazon EC2 ユーザーガイド」の「インスタンスタイプのネットワークインターフェイスあたりの IP アドレス数」を参照してください。
- Amazon ECS では、高い ENI 密度がサポートされているインスタンスタイプを使用して、Amazon EC2 Linux インスタンスを起動します。awsvpcTrunking アカウント設定にオプトインし、これらのインスタンスタイプを使用して Amazon EC2 Linux インスタンスをクラスターに登録すると、対象のインスタンスで ENI のクオータが引き上げられます。これらの、クォータが引き上げられたインスタンスを使用することで、各 Amazon EC2 Linux インスタンスにさらに多くのタスクを配置できます。高い ENI 密度をトランキング機能で使用するには、Amazon EC2 インスタンスにコンテナエージェントのバージョン 1.28.1 以降が必要です。Amazon ECS 最適化 Linux AMI を使用している場合には、インスタンスに ecs-init パッケージの 1.28.1-2 バー

ジョン以降も必要です。awsvpcTrunking アカウント設定のオプトインの詳細については、「アカウント設定による Amazon ECS 機能へのアクセス」を参照してください。ENI トランキングの詳細については、「Amazon ECS Linux コンテナインスタンスのネットワークインターフェイスを増やす」を参照してください。

- Amazon EC2 Linux インスタンスで awsvpc ネットワークモードを使用するタスクをホストする場合、タスク ENI にはパブリック IP アドレスが付与されません。インターネットにアクセスするには、NAT ゲートウェイを使用するよう設定されたプライベートサブネットでタスクを起動する必要があります。詳細については、「Amazon VPC ユーザーガイド」の「NAT ゲートウェイ」を参照してください。インバウンドのネットワークアクセスは、プライベート IP アドレスを使用する VPC 内からのものか、その VPC からロードバランサーを経由させルーティングされたものである必要があります。パブリックサブネット内で起動されたタスクは、インターネットにアクセスできません。
- Amazon ECS は、ユーザーの Amazon EC2 Linux インスタンスにアタッチする ENI のみを認識します。ENI をインスタンスに手動でアタッチした場合は、十分なネットワークアダプタを持たないインスタンスに対しても、Amazon ECS がタスクの追加を試みる可能性があります。これはタスクでのタイムアウトを引き起こし、ステータスがプロビジョニング解除へ、さらに停止状態へと移行する可能性があります。インスタンスに対する ENI の手動によるアタッチは推奨されません。
- Amazon EC2 インスタンスは、awsvpc ネットワークモードでのタスク配置を検討する ecs.capability.task-eni 機能に登録する必要があります。1.15.0-4 の ecs-init バージョン以降を実行するインスタンスは、自動的にこの属性に登録されます。
- Amazon EC2 Linux インスタンスに作成およびアタッチされた ENI は、手動でデタッチしたり、ユーザーのアカウントを使って変更したりすることはできません。これは、実行中のタスクに関連付けられている ENI が誤って削除されないようにするためです。タスクの ENI を解放するには、タスクを停止します。
- タスクを実行するときや、awsVpcConfiguration ネットワークモードを使用するサービスを作成するときは、awsvpc に指定できるサブネットには 16 個、セキュリティグループには 5 個という制限があります。詳細については、Amazon Elastic Container Service API リファレンス の「AwsVpcConfiguration」を参照してください。
- awsvpc ネットワークモードを使用してタスクが開始されると、タスク定義内のコンテナが開始される前に、各タスクに Amazon ECS コンテナエージェントによって追加の pause コンテナが作成されます。次に、amazon-ecs-cni-plugins CNI プラグインを実行して pause コンテナのネットワーク名前空間が設定されます。その後、エージェントによってタスク内の残りのコンテナが開始されます。こうすることで pause コンテナのネットワークスタックが共有されます。つまり、タスク内のすべてのコンテナは ENI の IP アドレスによってアドレス可能であり、localhost インターフェイス経由で相互に通信できます。

- サービスに含まれるタスクに、awsvpc ネットワークモードを使用するものがある場合は、Application Load Balancer と Network Load Balancer のみがサポートされます。このようなサービス用にターゲットグループを作成する場合は、ターゲットタイプとして ip を選択する必要があります。instance を使用しないでください。これは、awsvpc ネットワークモードを使用するタスクは、Amazon EC2 Linux インスタンスではなく、ENI に関連付けられているためです。詳細については、「ロードバランサーを使用して Amazon ECS サービストラフィックを分散する」を参照してください。
- 使用中の DHCP オプション設定を変更するように VPC が更新された場合も、既存のタスクにこれらの変更を適用することはできません。これらのネットワーク設定を安全に変更するためには、変更内容を適用して新しいタスクを開始し、それらの動作が正常なことを確認した上で、既存のタスクを停止します。

Windows に関する考慮事項

Windows オペレーティングシステムを使用するときは、考慮事項を次に示します:

- Amazon ECS 最適化 Windows Server 2016 AMI を使用するコンテナインスタンスは、awsvpc ネットワークモードを使用するタスクをホストしません。Amazon ECS 最適化 Windows Server 2016 AMI、および awsvpc ネットワークネットワークモ ードをサポートする Windows AMI を含むクラスターを使用する場合、awsvpc ネットワークモードを使用するタスクは Windows 2016 Server インスタンスでは起動されません。代わりに、awsvpc ネットワークモードがサポートされるインスタンスで起動されます。
- Amazon EC2 Windows インスタンスでは、awsvpc ネットワークモードを使用する Windows コンテナに CloudWatch メトリクスを使用するには、バージョン 1.57.1 以降のコンテナエージェントが必要です。
- awsvpc ネットワークモード使用するタスクとサービスには、Amazon ECS サービスにリンクされたロールが必要です。このロールにより、ユーザーに代わってその他の AWS サービスを呼び出す許可を、Amazon ECS に付与できるようになります。このロールは、クラスターを作成する際、または AWS マネジメントコンソール でサービスを作成または更新すると、自動的に作成されます。詳細については、「Amazon ECS のサービスリンクロールの使用」を参照してください。サービスにリンクされたロールは、次の AWS CLI コマンドを使用して作成することもできます。

```
aws iam create-service-linked-role --aws-service-name ecs.amazonaws.com
```

- Amazon EC2 Windows インスタンスでは、awsvpc ネットワークモードを使用するタスクを実行するには、コンテナエージェントのバージョン 1.54.0 以降が必要です。インスタンスをブート

ストラップするときは、awsvpc ネットワークモードに必要なオプションを設定する必要があります。詳しくは、「the section called “コンテナインスタンスのブートストラップ”」を参照してください。

- VPC で enableDnsHostnames と enableDnsSupport オプションの両方が有効になっている場合、Amazon ECS は、Amazon が提供する (内部) DNS ホスト名を使用してタスクのホスト名を設定します。これらのオプションが有効になっていない場合は、タスクの DNS ホスト名はランダムな名前に設定されます。VPC の DNS 設定について、詳細は「Amazon VPC ユーザーガイド」の「VPC で DNS を使用する」を参照してください。
- awsvpc モードを使用する Amazon ECS タスクには、それぞれ独自の Elastic Network Interface (ENI) が提供されます。この ENI は、ENI をホストする Amazon EC2 Windows インスタンスにアタッチされています。Amazon EC2 Windows インスタンスへのアタッチが可能な、ネットワークインターフェイスの数には、デフォルトでクォータが設定されます。このクォータに対して、プライマリネットワークインターフェイスは 1 個としてカウントされます。例えば、 c5.large インスタンスのデフォルトでは、最大 3 個までの ENI がアタッチできます。このインスタンスのプライマリネットワークインターフェイスも、それら ENI の内の 1 個としてカウントされます。さらに 2 つの ENI をインスタンスにアタッチできます。awsvpc ネットワークモードを使用する各タスクには ENI が必要なため、通常、このインスタンスタイプでは、このようなタスクを 2 つのみ実行できます。各インスタンスタイプのデフォルトの ENI 制限については、「Amazon EC2 ユーザーガイド」の「インスタンスタイプのネットワークインターフェイスあたりの IP アドレス数」を参照してください。
- Amazon EC2 Windows インスタンスで awsvpc ネットワークモードを使用するタスクをホストする場合、タスク ENI にはパブリック IP アドレスが付与されません。インターネットにアクセスするには、NAT ゲートウェイを使用するよう設定されたプライベートサブネットで、タスクを起動します。詳細については、「Amazon VPC ユーザーガイド」の「NAT ゲートウェイ」を参照してください。インバウンドのネットワークアクセスは、プライベート IP アドレスを使用する VPC 内からのものか、VPC 内のロードバランサーを経由してルーティングされたものである必要があります。パブリックサブネット内で起動されたタスクは、インターネットにアクセスできません。
- Amazon ECS は、Amazon EC2 Windows インスタンスにアタッチした ENI だけを認識します。ENI をインスタンスに手動でアタッチした場合は、十分なネットワークアダプタを持たないインスタンスに対しても、Amazon ECS がタスクの追加を試みる可能性があります。これはタスクでのタイムアウトを引き起こし、ステータスがプロビジョニング解除へ、さらに停止状態へと移行する可能性があります。インスタンスに対する ENI の手動によるアタッチは推奨されません。
- Amazon EC2 Windows インスタンスは、awsvpc ネットワークモードでのタスク配置を検討する ecs.capability.task-eni 機能に登録する必要があります。

- Amazon EC2 Windows インスタンスに作成およびアタッチされた ENI は、手動で変更したり、デタッチしたりはできません。これは、実行中のタスクに関連付けられている ENI を、誤って削除してしまわないようにするためです。タスクの ENI を解放するには、タスクを停止します。
- awsvpc ネットワークモードを使用するタスクの実行時、もしくはサービスの作成時は、最大 16 個までのサブネットと 5 個までのセキュリティグループを awsVpcConfiguration 内で指定できます。詳細については、Amazon Elastic Container Service API リファレンス の「AwsVpcConfiguration」を参照してください。
- awsvpc ネットワークモードを使用してタスクが開始されると、タスク定義内のコンテナが開始される前に、各タスクに Amazon ECS コンテナエージェントによって追加の pause コンテナが作成されます。次に、amazon-ecs-cni-plugins CNI プラグインを実行して pause コンテナのネットワーク名前空間が設定されます。その後、エージェントによってタスク内の残りのコンテナが開始されます。こうすることで pause コンテナのネットワークスタックが共有されます。つまり、タスク内のすべてのコンテナは ENI の IP アドレスによってアドレス可能であり、localhost インターフェイス経由で相互に通信できます。
- サービスに含まれるタスクに、awsvpc ネットワークモードを使用するものがある場合は、Application Load Balancer と Network Load Balancer のみがサポートされます。このようなサービス用にターゲットグループを作成する場合は、ターゲットタイプとして instance ではなく、ip を選択する必要があります。これは、awsvpc ネットワークモードを使用するタスクは、Amazon EC2 Windows インスタンスではなく、ENI に関連付けられているためです。詳細については、「ロードバランサーを使用して Amazon ECS サービストラフィックを分散する」を参照してください。
- 使用中の DHCP オプション設定を変更するように VPC が更新された場合も、既存のタスクにこれらの変更を適用することはできません。これらのネットワーク設定を安全に変更するためには、変更内容を適用して新しいタスクを開始し、それらの動作が正常なことを確認した上で、既存のタスクを停止します。
- EC2 Windows 構成で awsvpc ネットワークモードを使用する場合、以下はサポートされません。
  - デュアルスタック設定
  - IPv6
  - ENI トランキング

デュアルスタックモードでの VPC の使用

デュアルスタックモードで VPC を使用する場合、タスクは IPv4 または IPv6、あるいはその両方を経由して通信できます。IPv4 と IPv6 アドレスは、互いに独立しています。そのため、VPC 内で IPv4 と IPv6 のルーティングとセキュリティを別々に設定する必要があります。VPC をデュアルス

タックモード用に設定する方法については、「Amazon VPC ユーザーガイド」の「既存の VPC を IPv4 から IPv6 に移行する」を参照してください。

VPC でインターネットゲートウェイまたは送信専用インターネットゲートウェイを設定した場合は、その VPC をデュアルスタックモードで使用できます。この設定により、IPv6 アドレスが割り当てられたタスクは、インターネットゲートウェイまたは送信専用インターネットゲートウェイを介して、インターネットへのアクセスが可能になります。NAT ゲートウェイはオプションです。詳細については、Amazon VPC ユーザーガイドの「インターネットゲートウェイ」および「Egress-only インターネットゲートウェイ」を参照してください。

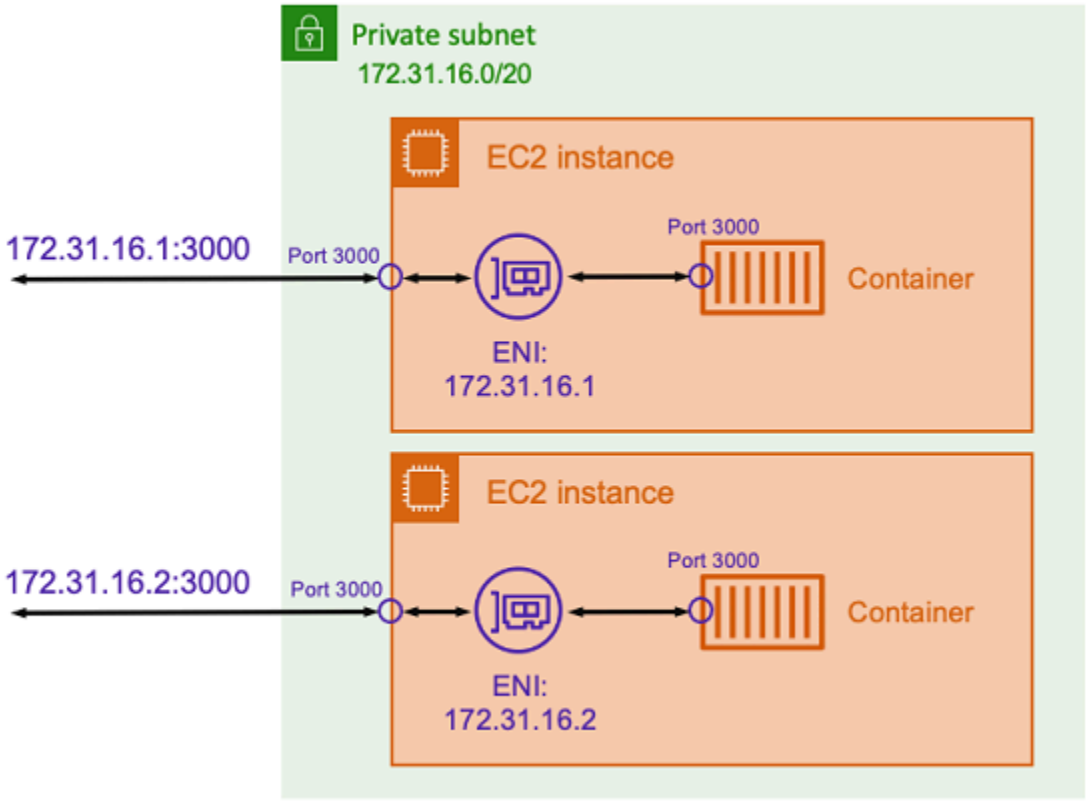
以下の条件が満たされた場合、Amazon ECS タスクには IPv6 アドレスが割り当てられます。

- タスクをホストしている Amazon EC2 インスタンスは、バージョン `1.45.0` 以降のコンテナエージェントを使用しています。インスタンスが使用しているエージェントのバージョンを確認し、必要に応じて更新する方法については、「Amazon ECS コンテナエージェントをアップデートする」を参照してください。
- `dualStackIPv6` アカウント設定が有効になります。詳細については、「アカウント設定による Amazon ECS 機能へのアクセス」を参照してください。
- タスクは `awsvpc` ネットワークモードを使用しています。
- VPC とサブネットは、IPv6 に対して有効になっています。この設定には、指定されたサブネットで作成されたネットワークインターフェイスが含まれます。VPC をデュアルスタックモードに設定する方法について、詳しくは「Amazon VPC ユーザーガイド」の「IPv6 へ移行する」および「サブネットの IP アドレス指定属性を変更する」を参照してください。

Amazon ECS コンテナポートを EC2 インスタンスネットワークインターフェイスにマッピングする

`host` ネットワークモードは Amazon EC2 インスタンスでホストされている Amazon ECS タスクにのみサポートされています。Fargate で Amazon ECS を使用する場合はサポートされません。

`host` ネットワークモードは、Amazon ECS でサポートされている最も基本的なネットワークモードです。host モードでは、コンテナを稼働しているホストに直接関連付けられます。



上の図に示したようなポート `3000` でリッスンする Express アプリケーションで Node.js コンテナを実行していると仮定します。`host` ネットワークモードを使用すると、コンテナは基盤となるホストの Amazon EC2 インスタンスの IP アドレスを使用してポート 3000 でトラフィックを受信します。このモードを使用しないことをお勧めします。

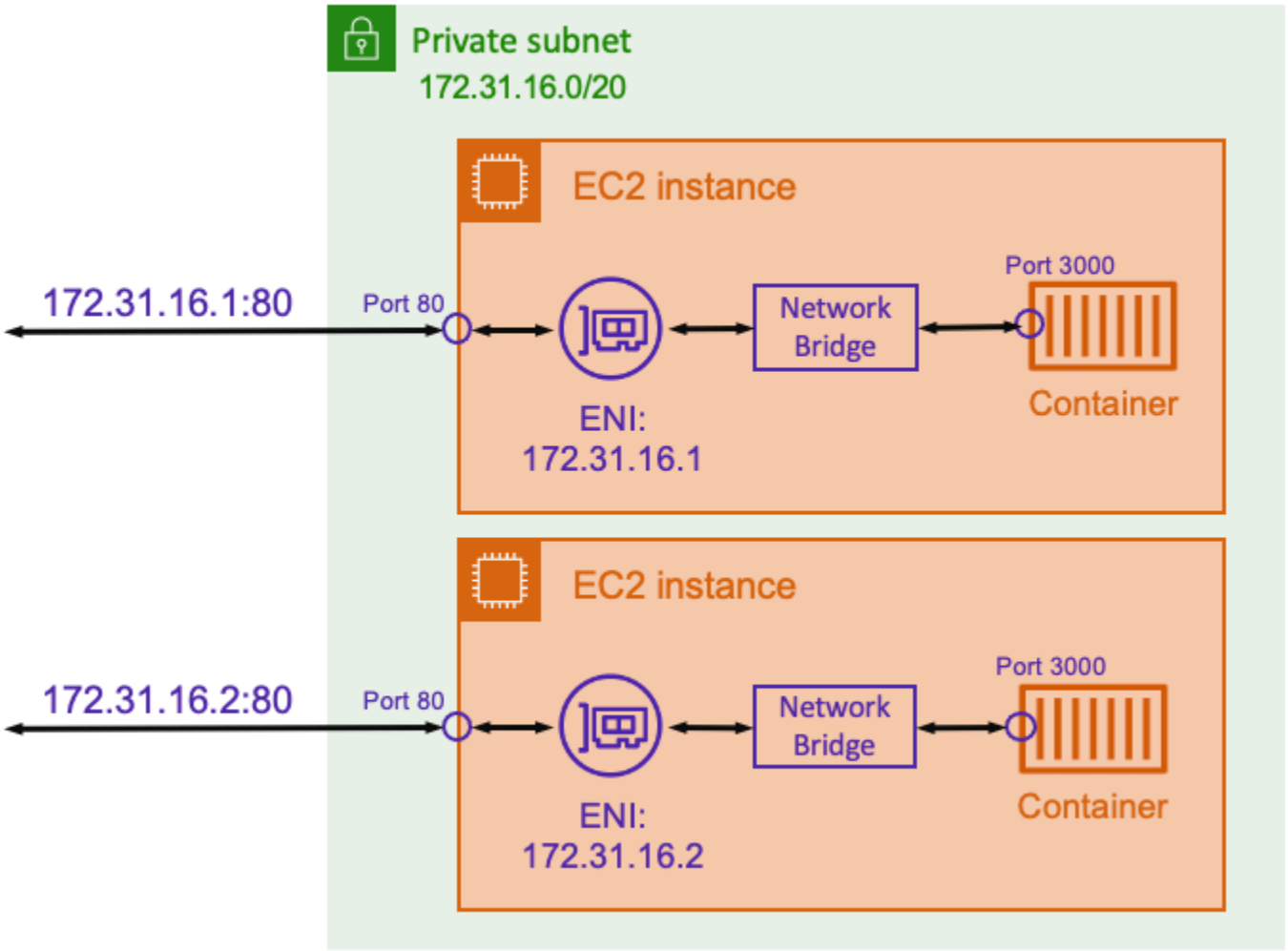
このネットワークモードを使用することには重大な欠点があります。各ホストで 1 つのタスクを複数インスタンス化することはできません。これは、Amazon EC2 インスタンスの必要なポートにバインドできるのは最初のタスクだけだからです。また、`host` ネットワークモードを使用している場合、コンテナポートを再マップする方法はありません。例えば、アプリケーションが特定のポート番号でリッスンする必要がある場合、ポート番号を直接再マップすることはできません。代わりに、アプリケーション構成を変更してポートの競合を管理する必要があります。

`host` ネットワークモードを使用する際には、セキュリティ上の問題もあります。このモードでは、コンテナがホストになりすますことができ、コンテナはホスト上のプライベートループバックネットワークサービスに接続できます。

Amazon ECS Linux タスクに Docker の仮想ネットワークを使用する

`bridge` ネットワークモードは Amazon EC2 インスタンスでホストされている Amazon ECS タスクにのみサポートされています。

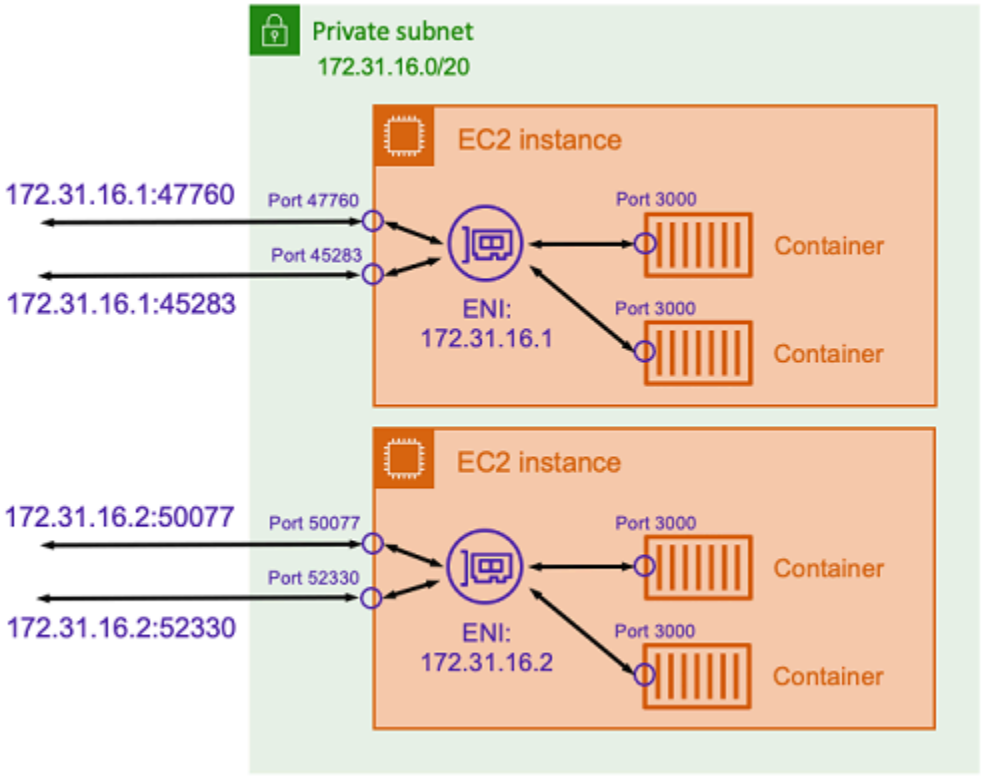
`bridge` モードでは、仮想ネットワークブリッジを使用してホストとコンテナのネットワーク間にレイヤーを作成します。これにより、ホストポートをコンテナポートに再マップするポートマッピングを作成できます。マッピングは静的または動的に実行することができます。



スタティックポートマッピングでは、どのホストポートをコンテナポートにマッピングするかを明示的に定義できます。前述の例を使用すると、ホスト上のポート `80` はコンテナのポート `3000` にマップされます。コンテナ化されたアプリケーションと通信するには、Amazon EC2 インスタンスの IP アドレスのポート `80` にトラフィックを送信します。コンテナ化されたアプリケーションの観点から見ると、ポート `3000` 上のインバウンドトラフィックがわかります。

トラフィックポートのみを変更する場合は、静的ポートマッピングが適しています。ただし、これには host ネットワークモードを使用する場合と同じ欠点があります。各ホストで 1 つのタスクを複数インスタンス化することはできません。これは、静的ポートマッピングでは、1 つのコンテナしかポート 80 にマッピングできないためです。

この問題を解決するには、次の図に示すように、bridge ネットワークモードと動的ポートマッピングを使用することを検討してください。



ポートマッピングでホストポートを指定しないことで、Docker にエフェメラルポート範囲から未使用のポートをランダムに選択させ、それをコンテナのパブリックホストポートとして割り当てることができます。例えば、コンテナのポート 3000 をリッスンしている Node.js アプリケーションには、Amazon EC2 ホスト上の 47760 のようにランダムに大きい番号のポートが割り当てられる場合があります。これにより、そのコンテナの複数のコピーをホスト上で実行できるようになります。さらに、各コンテナにはホスト上の独自のポートを割り当てることができます。コンテナの各コピーは、ポート 3000 でトラフィックを受信します。ただし、これらのコンテナにトラフィックを送信するクライアントは、ランダムに割り当てられたホストポートを使用します。

Amazon ECS は、各タスクにランダムに割り当てられたポートを追跡するのに役立ちます。これは、ロードバランサーのターゲットグループと AWS Cloud Map サービス検出を自動的に更新して、

タスクの IP アドレスとポートのリストを取得することによって行われます。このため、ダイナミックポートで bridge モードを使用するときのみにサポートされます。

ただし、bridge ネットワークモードを使用するデメリットの 1 つは、サービス間の通信をロックダウンするのが難しいことです。サービスはランダムで未使用のポートに割り当てられる可能性があるため、ホスト間で幅広いポート範囲を開く必要があります。ただし、特定のサービスが他の 1 つの特定のサービスとしか通信できないように特定のルールを作成するのは簡単ではありません。サービスには、セキュリティグループのネットワークルールに使用する特定のポートはありません。

IPv6-only ワークロードのブリッジネットワークモードの設定

IPv6 経由の通信において bridge モードを設定するには、Docker のデーモン設定を更新する必要があります。以下の手順に従って /etc/docker/daemon.json を更新してください。

```
{
  "ipv6": true,
  "fixed-cidr-v6": "2001:db8:1::/64",
  "ip6tables": true,
  "experimental": true
}
```

Docker デーモン設定を更新したら、デーモンを再起動してください。

> **Note**
>
> デーモンを更新して再起動すると、Docker はインスタンスで IPv6 転送を有効にします。これにより、Amazon Linux 2 AMI を使用するインスタンスでデフォルトルートが失われる可能性があります。これを回避するには、次のコマンドを使用して、サブネットの IPv6 ゲートウェイを介してデフォルトルートを追加します。
>
> ```
> ip route add default via FE80:EC2::1 dev eth0 metric 100
> ```

Fargate における Amazon ECS タスクのネットワークオプション

デフォルトでは、Fargate 上のすべての Amazon ECS タスクには、プライマリプライベート IP アドレスを備えた Elastic Network Interface (ENI) が提供されます。パブリックサブネットを使用する際には、オプションで、タスクの ENI にパブリック IP アドレスを割り当てることができます。VPC

がデュアルスタックモード向けに設定されていて、IPv6 CIDR ブロックを備えたサブネットを使用する場合、タスクの ENI にも IPv6 アドレスが割り当てられます。タスクには、一度に 1 つの ENI しか関連付けられません。また、同じタスクに属するコンテナでも、`localhost` インターフェイス経由での通信が可能になります。VPC とサブネットの詳細については、「Amazon VPC ユーザーガイド」の「Amazon VPC の仕組み」を参照してください。

Fargate のタスクがコンテナイメージをプルできるようにするには、そのタスクにインターネットへのルートが必要です。次に、タスクにインターネットへのルートがあるか検証する方法について説明します。

- パブリックサブネットを使用する場合、タスク ENI にパブリック IP アドレスを割り当てることができます。
- プライベートサブネットを使用する場合、サブネットに NAT ゲートウェイをアタッチできます。
- Amazon ECR でホストされるコンテナイメージを使用する場合、インターフェイスの VPC エンドポイントを使用するように Amazon ECR を設定でき、イメージのプルはタスクのプライベート IPv4 アドレス上で実行されます。詳細については、Amazon Elastic コンテナレジストリ ユーザーガイド の Amazon ECR Interface VPC エンドポイント(AWS PrivateLink) を参照してください。

各タスクにはそれぞれ独自の ENI が提供されるため、VPC フローログなどのネットワーキング機能を使用して、タスクとの間で送受信されるトラフィックをモニタリングできるようになります。詳細については、「Amazon VPC ユーザーガイド」の「VPC フローログを使用した IP トラフィックのログ記録」を参照してください。

AWS PrivateLink を活用することもできます。VPC インターフェースエンドポイントを設定することで、プライベート IP アドレスを通じて Amazon ECS の API にアクセスができます。AWS PrivateLink は、VPC と Amazon ECS 間のすべてのネットワークトラフィックを Amazon ネットワークに制限します。インターネットゲートウェイ、NAT デバイス、または仮想プライベートゲートウェイは必要ありません。詳細は「Amazon ECS インターフェイス VPC エンドポイント (AWS PrivateLink)」を参照してください。

CloudFormation で `NetworkConfiguration` リソースを使用する方法の例については、「the section called “CloudFormation テンプレートの例”」を参照してください。

作成した ENI は、AWS Fargate によって完全に管理されます。加えて、Fargate へのアクセス許可を付与するために使用される IAM ポリシーが関連付けられます。Fargate プラットフォームバージョン `1.4.0` 以降を使用するタスクは、単一の ENI (タスク ENI と呼ばれる) を受け取ります。すべてのネットワークトラフィックは、VPC 内でこの ENI を通過します。このトラフィックは VPC フローログに記録されます。Fargate プラットフォームバージョン `1.3.0` 以前を使用するタスクに

は、タスク ENI に加えて Fargate が所有する ENI も別に割り当てられます。この ENI は、VPC フローログに表示されない一部のネットワークトラフィックに使用されます。次の表で、ネットワークトラフィックの動作と、プラットフォームバージョンごとに必要な IAM ポリシーについて説明します。

アクション	Linux プラットフォームバージョン **1.3.0** 以前でのトラフィックフロー	Linux プラットフォームバージョン **1.4.0** でのトラフィックフロー	Windows プラットフォームバージョン **1.0.0** でのトラフィックフロー	IAM アクセス許可
Amazon ECR ログイン認証情報の取得	Fargate が所有する ENI	タスク ENI	タスク ENI	タスク実行IAM ロール
イメージプル	タスク ENI	タスク ENI	タスク ENI	タスク実行IAM ロール
ログドライバーによるログの送信	タスク ENI	タスク ENI	タスク ENI	タスク実行IAM ロール
FireLens for Amazon ECS を介したログの送信	タスク ENI	タスク ENI	タスク ENI	タスク IAM ロール
Secrets Manager またはSystems Manager からシークレットの取得	Fargate が所有する ENI	タスク ENI	タスク ENI	タスク実行IAM ロール
Amazon EFS ファイルシステムトラフィック	利用不可	タスク ENI	タスク ENI	タスク IAM ロール

アクション	Linux プラットフォームバージョン **1.3.0** 以前でのトラフィックフロー	Linux プラットフォームバージョン **1.4.0** でのトラフィックフロー	Windows プラットフォームバージョン **1.0.0** でのトラフィックフロー	IAM アクセス許可
アプリケーションのトラフィック	タスク ENI	タスク ENI	タスク ENI	タスク IAM ロール

考慮事項

タスクネットワーキングを使用する際は、以下の点を考慮してください。

- Amazon ECS のサービスにリンクされたロールは、ユーザーの代わりにその他の AWS サービスを呼び出すアクセス許可 を Amazon ECS に付与します。このロールは、クラスターを作成する際、または AWS マネジメントコンソール 内のサービスを作成または更新する際に自動的に作成されます。詳細については、「Amazon ECS のサービスリンクロールの使用」を参照してください。サービスにリンクされたロールは、次の AWS CLI コマンドを使用して作成することもできます。

  ```
  aws iam create-service-linked-role --aws-service-name ecs.amazonaws.com
  ```

- VPC で `enableDnsHostnames` と `enableDnsSupport` オプションの両方が有効になっている場合、Amazon ECS は、Amazon が提供する (内部) DNS ホスト名を使用してタスクのホスト名を設定します。これらのオプションが有効でない場合、タスクの DNS ホスト名にはランダムな名前が付けられます。VPC の DNS 設定について、詳細は「Amazon VPC ユーザーガイド」の「VPC で DNS を使用する」を参照してください。
- `awsVpcConfiguration` には、最大 16 個のサブネットまでと、5 個のセキュリティグループまでが指定可能です。詳細については、Amazon Elastic Container Service API リファレンス の「AwsVpcConfiguration」を参照してください。
- Fargate によって作成およびアタッチされた ENI は、手動でデタッチしたり変更したりできません。これは、実行中のタスクに関連付けられている ENI が誤って削除されないようにするためです。タスクの ENI を解放するには、タスクを停止します。
- VPC サブネットが更新され、使用する DHCP オプションセットが変更された場合でも、VPC を使用する既存のタスクにこれらの変更を適用することはできません。新しい設定で新しいタスクを

開始し、変更内容をテストしてロールバックの必要性がないことを確認した上で、古いタスクを停止しスムーズに移行を行います。

- 以下は、Linux の場合は Fargate プラットフォームバージョン `1.4.0` 以降、Windows の場合は `1.0.0` 以降で実行されるタスクに適用されます。デュアルスタックサブネットで起動されたタスクは、IPv4 アドレスおよび IPv6 アドレスを受け取ります。IPv6-only サブネットで起動されたタスクは、IPv6 アドレスのみを受け取ります。
- プラットフォームバージョン `1.4.0` 以降 (Linux の場合)、またはバージョン `1.0.0` (Windows の場合) を使用するタスクの場合、タスク ENI はジャンボフレームをサポートします。ネットワークインターフェイスは、最大転送単位 (MTU) で設定されます。MTU は、1 つのフレームに収まるペイロードの最大サイズです。MTU が大きいほど、1 つのフレーム内に収まるアプリケーションのペイロードが増えるため、フレームあたりのオーバーヘッドが減少し、効率が向上します。ジャンボフレームをサポートし、タスクと転送先とのネットワークパスでジャンボフレームをサポートすると、オーバーヘッドを削減できます。
- Fargate を使用するタスクのサービスは、Application Load Balancer と Network Load Balancer のみをサポートします。Classic Load Balancer はサポートされていません。ターゲットグループを作成する場合は、ターゲットタイプとして `instance` ではなく、`ip` を選択する必要があります。詳細については、「ロードバランサーを使用して Amazon ECS サービストラフィックを分散する」を参照してください。

デュアルスタックモードでの VPC の使用

デュアルスタックモードで VPC を使用する場合、タスクは IPv4 または IPv6、あるいはその両方を経由して通信できます。IPv4 アドレスと IPv6 アドレスは互いに独立しています。また VPC で IPv4 と IPv6 のルーティングとセキュリティを設定する必要があります。VPC をデュアルスタックモードに設定する方法の詳細については、Amazon VPC ユーザーガイドの「IPv6 への移行」を参照してください。

以下の条件が満たされた場合、Fargate の Amazon ECS タスクには IPv6 アドレスが割り当てられます。

- タスクを起動するリージョンで、タスクを起動する IAM プリンシパルの Amazon ECS `dualStackIPv6` アカウント設定がオン (`enabled`) になっていること。この設定は、API もしくは AWS CLI を使用してのみ変更できます。アカウントの特定の IAM プリンシパルに対してのみこの設定を有効にしてもよいし、あるいはアカウントのデフォルト設定を変更してアカウント全体に対して有効にすることもできます。詳細については、「アカウント設定による Amazon ECS 機能へのアクセス」を参照してください。

- VPC とサブネットが IPv6 に対して有効になっています。VPC をデュアルスタックモード用に設定する方法については、「Amazon VPC ユーザーガイド」の「既存の VPC を IPv4 から IPv6 に移行する」を参照してください。
- サブネットが、IPv6 アドレスの自動割り当てに有効になっています。サブネットの設定方法について、詳細は「Amazon VPC ユーザーガイド」の「サブネットの IPv6 アドレス指定属性を変更する」を参照してください。
- タスクまたはサービスは、Linux 用の Fargate プラットフォームバージョン `1.4.0` 以降を使用します。

デュアルスタックモードを用いて VPC 内で実行されている Fargate 上の Amazon ECS タスクの場合、ECR、SSM、SecretManager などのタスク起動プロセスで使用される依存関係サービスとの通信には、パブリックサブネットのルートテーブルにインターネットゲートウェイへの IPv4 (0.0.0.0/0) ルート、プライベートサブネットのルートテーブルに NAT ゲートウェイへの IPv4 (0.0.0.0/0) ルートが必要です。詳細については、「Amazon VPC ユーザーガイド」の「インターネットゲートウェイ」および「NAT ゲートウェイ」を参照してください。

デュアルスタック VPC を設定する方法の例については、「デュアルスタック VPC 設定の例」を参照してください。

IPv6-only モードで VPC を使用する

IPv6-only 設定では、Amazon ECS タスクは IPv6 経由でのみ通信します。VPC およびサブネットを IPv6-only モードに設定するには、IPv6 CIDR ブロックを VPC に追加し、IPv6 CIDR ブロックのみを含むサブネットを作成してください。詳しくは「Amazon VPC ユーザーガイド」の「VPC の IPv6 サポートを追加する」および「サブネットを作成する」を参照してください。また、IPv6 ターゲットでルートテーブルを更新し、IPv6 ルールでセキュリティグループを設定する必要があります。詳しくは「Amazon VPC ユーザーガイド」の「ルートテーブルを設定する」および「セキュリティグループルールを設定する」を参照してください。

以下の考慮事項に注意してください。

- Amazon ECS サービスを IPv6-only サブネットを使用する設定に直接更新するか、別の IPv6-only サービスを作成し、Amazon ECS ブルー/グリーンデプロイを用いて新サービスへのトラフィック移行を行うことで、IPv4-only 設定の更新および IPv6-only 設定のデュアルスタックを有効にすることが可能です。Amazon ECS におけるブルー/グリーンデプロイについて、詳しくは「Amazon ECS ブルー/グリーンデプロイ」を参照してください。
- IPv6-only Amazon ECS サービスは、IPv6 ターゲットグループを保有するデュアルスタックロードバランサーを使用する必要があります。Application Load Balancer または Network Load Balancer

の背後で稼働する既存の Amazon ECS サービスを移行する場合は、新しいデュアルスタックロードバランサーを作成して旧ロードバランサーからトラフィックを切り替えるか、既存のロードバランサーの IP アドレスタイプを更新します。

Network Load Balancer について、詳しくは「Network Load Balancer ユーザーガイド」の「Network Load Balancer を作成する」および「Network Load Balancer の IP アドレスタイプを更新する」を参照してください。Application Load Balancer について、詳しくは「Application Load Balancer ユーザーガイド」の「Application Load Balancer を作成する」および「Application Load Balancer の IP アドレスタイプを更新する」を参照してください。

- IPv6-only 設定は Windows ではサポートされていません。
- IPv4-only エンドポイントと交信する IPv6-only 設定の Amazon ECS タスクにおいては、IPv6 から IPv4 へのネットワークアドレス変換に DNS64 と NAT64 を設定できます。詳細については、Amazon VPC ユーザーガイドの DNS64 および NAT64 を参照してください。
- IPv6-only 設定は、Fargate プラットフォームバージョン `1.4.0` 以降に対応しています。
- IPv6-only 設定の Amazon ECS ワークロードでは、Amazon ECR からイメージを取得する際に Amazon ECR デュアルスタックイメージ URI エンドポイントを使用する必要があります。詳しくは「Amazon Elastic Container Registry ユーザーガイド」の「Getting started with making requests over IPv6」を参照してください。

> **Note**
>
> Amazon ECR は、IPv6-only 設定のタスクが使用するデュアルスタックインターフェイス VPC エンドポイントには対応していません。詳しくは「Amazon Elastic Container Registry ユーザーガイド」の「Getting started with making requests over IPv6」を参照してください。

- Amazon ECS Exec は IPv6-only 設定には対応していません。
- Amazon CloudWatch は、FIPS-140 準拠の IPv6-only 設定で動作する Amazon ECS タスクに対する監視用途で使用するデュアルスタック FIPS エンドポイントには対応していません。FIPS-140 について、詳しくは「AWS Fargate 連邦情報処理標準 (FIPS-140)」を参照してください。

AWS リージョンAmazon ECS の IPv6-only モードをサポートしている

IPv6-only 設定でのタスクは、Amazon ECS が利用可能な以下の AWS リージョンで実行できます。

- 米国東部(オハイオ)
- 米国東部 (バージニア北部)

- 米国西部 (北カリフォルニア)
- 米国西部 (オレゴン)
- アフリカ (ケープタウン)
- アジアパシフィック (香港)
- アジアパシフィック (ハイデラバード)
- アジアパシフィック (ジャカルタ)
- アジアパシフィック (メルボルン)
- アジアパシフィック (ムンバイ)
- アジアパシフィック (大阪)
- アジアパシフィック (ソウル)
- アジアパシフィック (シンガポール)
- アジアパシフィック (シドニー)
- アジアパシフィック (東京)
- カナダ (中部)
- カナダ西部 (カルガリー)
- 中国 (北京)
- 中国 (寧夏)
- 欧州 (フランクフルト)
- 欧州 (ロンドン)
- 欧州 (ミラノ)
- 欧州 (パリ)
- 欧州 (スペイン)
- イスラエル (テルアビブ)
- 中東 (バーレーン)
- 中東 (アラブ首長国連邦)
- 南米 (サンパウロ)
- AWS GovCloud (米国東部)
- AWS GovCloud (米国西部)

Amazon ECS タスクのストレージオプション

Amazon ECS には、柔軟で使いやすく、コスト効率の良い各種データストレージオプションが用意されています。Amazon ECS は、以下のコンテナ向けデータボリュームオプションをサポートします。

データボリューム	対応容量	サポートされるオペレーティングシステム	ストレージの永続化	ユースケース
Amazon Elastic Block Store (Amazon EBS)	Fargate、Amazon EC2、Amazon ECS マネージドインスタンス	Linux、Windows (Amazon EC2 のみ)	スタンドアロンタスクにアタッチすることにより永続化できます。サービスが管理するタスクにアタッチすると一時的になります。	Amazon EBS ボリュームは、データ集約型のコンテナ化されたワークロード向けに、費用対効果と耐久性に優れた高性能なブロックストレージを提供します。一般的なユースケースには、データベース、仮想デスクトップ、ルートボリュームなどのトランザクションワークロード、またログ処理や ETL ワークロードなどのスループット集約型ワークロードがあります。詳細については、「Amazon ECS

データボリューム	対応容量	サポートされるオペレーティングシステム	ストレージの永続化	ユースケース
				での Amazon EBS ボリュームの使用」を参照してください。

データボリューム	対応容量	サポートされるオペレーティングシステム	ストレージの永続化	ユースケース
Amazon Elastic File System (Amazon EFS)	Fargate、Amazon EC2、Amazon ECS マネージドインスタンス	Linux	永続	Amazon EFS ボリュームは、Amazon ECS タスクで使用できるシンプルかつスケーラブルで永続的な共有ファイルストレージです。ファイルの追加および削除に合わせて自動的に拡大/縮小します。Amazon EFS ボリュームは同時実行をサポートするため、低レイテンシー、高スループット、書き込み後の読み取り整合性などのストレージ機能を必要とする水平スケーリングタイプのコンテナ化されたアプリケーションに適しています。一般的なユースケースには、データ分析、メディア

データボリューム	対応容量	サポートされるオペレーティングシステム	ストレージの永続化	ユースケース
				処理、コンテンツ管理、ウェブサービスなどのワークロードが含まれます。詳細については、「Amazon ECS での Amazon EFS ボリュームの使用」を参照してください。

データボリューム	対応容量	サポートされるオペレーティングシステム	ストレージの永続化	ユースケース
Amazon FSx for Windows File Server	Amazon EC2	Windows	永続	FSx for Windows File Server ボリュームは、永続的、分散型、共有型、および静的なファイルストレージを必要とする Windows タスクのプロビジョニングに使用できる、フルマネージド型 Windows ファイルサーバー を提供します。一般的なユースケースには、アプリケーションの出力を保存するためにローカルフォルダーを永続ストレージとして必要とする.NET アプリケーションなどがあります。Amazon FSx for Windows File Server では、コンテナ内にローカルフォルダが用意され

データボリューム	対応容量	サポートされるオペレーティングシステム	ストレージの永続化	ユースケース
				ており、SMB 共有によりバックアップされている同一のファイルシステム上で複数のコンテナが読み取り/書き込みを行うことができます。詳細については、「Amazon ECS での FSx for Windows File Server ボリュームの使用」を参照してください。

データボリューム	対応容量	サポートされるオペレーティングシステム	ストレージの永続化	ユースケース
Amazon FSx for NetApp ONTAP	Amazon EC2 (ネイティブマウント)、Fargate (FSx の S3 アクセスポイント経由)	Linux	永続	Amazon FSx for NetApp ONTAP ボリュームは、永続的で高性能、豊富な機能を備えた共有ファイルストレージを必要とする Linux タスク向けにプロビジョニング可能な、フルマネージド型の NetApp ONTAP ファイルシステムを提供します。Amazon FSx for NetApp ONTAP は NFS プロトコルと SMB プロトコルをサポートし、スナップショット、クローン作成、データ重複排除などのエンタープライズ向け機能を提供します。一般的なユースケースには、高性能コンピューティング

データボリューム	対応容量	サポートされるオペレーティングシステム	ストレージの永続化	ユースケース
				ワークロード、コンテンツリポジトリ、POSIX準拠の共有ストレージを必要とするアプリケーションなどがあります。詳細については、「Mounting Amazon FSx for NetApp ONTAP file systems from Amazon ECS containers」を参照してください。 FSx for NetApp ONTAP のネイティブボリュームマウントは Amazon EC2 起動タイプでのみ使用できますが、FSx の S3 アクセスポイントを使用して Fargate タスクから FSx for NetApp ONTAP データにアクセスできます。こ

データボリューム	対応容量	サポートされるオペレーティングシステム	ストレージの永続化	ユースケース
				のアプローチにより、コンテナ化されたアプリケーションは、NFS または SMB の直接マウントを必要とせずに、Amazon Simple Storage Service API を介して FSx for ONTAP データを読み書きできます。詳細については、「FSx for ONTAP 用の Amazon S3 アクセスポイントの作成」を参照してください。

データボリューム	対応容量	サポートされるオペレーティングシステム	ストレージの永続化	ユースケース
Amazon S3 Files	Fargate、Amazon ECS マネージドインスタンス	Linux	永続	Amazon S3 Files は、マウント可能なファイルシステムインターフェイスを介して Amazon S3 データへの高速でキャッシュされたアクセスを提供する高性能ファイルシステムです。S3 Files ボリュームは、S3 バケットに保存されているデータへの直接ファイルシステムアクセスをコンテナに提供します。一般的なユースケースには、データ分析、機械学習トレーニング、および S3 データへの高スループットアクセスを必要とするアプリケーションなどがあります。詳細については、「Amazon

データボリューム	対応容量	サポートされるオペレーティングシステム	ストレージの永続化	ユースケース
				ECS の S3 Files の設定」を参照してください。

データボリューム	対応容量	サポートされるオペレーティングシステム	ストレージの永続化	ユースケース
Docker ボリューム	Amazon EC2	Windows、Linux	永続	Docker ボリュームは Docker コンテナランタイムの機能の一つで、ホストのファイルシステムからディレクトリをマウントすることでコンテナにデータを保持することができます。Docker ボリュームドライバー (プラグインとも呼ばれる) は、コンテナボリュームを外部ストレージシステムと統合するのに使用されます。Docker ボリュームは、サードパーティードライバーまたは組み込みの `local` ドライバーで管理できます。Docker ボリュームの一般的なユースケースには、永

データボリューム	対応容量	サポートされるオペレーティングシステム	ストレージの永続化	ユースケース
				続的データボリュームの提供や、同じコンテナインスタンスの異なるコンテナ上の別々の場所でのボリュームの共有などが含まれます。詳細については、「Amazon ECS での Docker ボリュームの使用」を参照してください。

データボリューム	対応容量	サポートされるオペレーティングシステム	ストレージの永続化	ユースケース
バインドマウント	Fargate、Amazon EC2、Amazon ECS マネージドインスタンス	Windows、Linux	一時的	バインドマウントは、コンテナにマウントされた Amazon EC2 インスタンスや AWS Fargate などのホスト上にあるファイルまたはディレクトリで構成されます。バインドマウントの一般的なユースケースには、ソースコンテナのボリュームを同じタスク内の他のコンテナと共有したり、あるいはホストボリュームや空のボリュームを 1 つもしくは複数のコンテナにマウントすることが含まれます。詳細については、「Amazon ECS でのバインドマウントの使用」を参照してください。

Amazon ECS での Amazon EBS ボリュームの使用

Amazon Elastic Block Store (Amazon EBS) ボリュームは、可用性、費用対効果と耐久性に優れた、データ集約型ワークロード向けの高性能ブロックストレージです。Amazon EBS ボリュームは、高スループットなトランザクション集約型アプリケーションの Amazon ECS タスクに使用できます。Amazon EBS ボリュームについて、詳細は「Amazon EBS ユーザーガイド」の「Amazon EBS ボリューム」を参照してください。

Amazon ECS タスクにアタッチされた Amazon EBS ボリュームは、お客様に代わって Amazon ECS で管理します。スタンドアロンタスクの起動時に、1 つの EBS ボリュームをタスクにアタッチする設定を提供できます。サービスの作成または更新時に、Amazon ECS サービスが管理する各タスクについて、タスクごとに 1 つの EBS ボリュームをアタッチする設定を提供できます。新しい空のボリュームを設定してアタッチすることもできるし、あるいはスナップショットを使用して既存のボリュームからデータをロードすることもできます。

> **Note**
>
> スナップショットを使用してボリュームを設定する場合、MiB/秒単位で `volumeInitializationRate` を指定することで、予測された時間内にスナップショットからデータを取得して完全に初期化されたボリュームを作成できます。ボリュームの初期化の詳細については、「Amazon EBS ユーザーガイド」の「Amazon EBS ボリュームの初期化」を参照してください。Amazon EBS ボリュームの設定の詳細については、「ボリューム設定を Amazon ECS のタスク定義の起動時刻まで延期する」と「Amazon ECS のデプロイ時に Amazon EBS ボリューム設定を指定する」を参照してください。

ボリューム設定は、タスク定義の `configuredAtLaunch` パラメータを使用して起動時間まで延期できます。タスク定義ではなく起動時にボリューム設定を提供することで、特定のデータボリュームタイプや特定の EBS ボリューム設定に制限されないタスク定義を作成できます。その後、作成したタスク定義をさまざまなランタイム環境で再利用できます。たとえば、本番環境向けワークロードのデプロイ時には、本番前のテスト環境よりも高いスループットを提供できます。

タスクにアタッチされた Amazon EBS ボリュームは、AWS Key Management Service (AWS KMS) キーで暗号化してデータを保護することができます。詳細については、「Amazon ECS タスクにアタッチされた Amazon EBS ボリュームに保存されているデータの暗号化」を参照してください。

ボリュームのパフォーマンスのモニタリングには、Amazon CloudWatch メトリクスを使用できます。Amazon EBS ボリューム向け Amazon ECS メトリクスの詳細については、「Amazon ECS

CloudWatch メトリクス」および「Amazon ECS Container Insights メトリクス」を参照してください。

Amazon EBS ボリュームのタスクへのアタッチは、Amazon ECS をサポートするすべての商用および中国 AWS リージョンでサポートされています。

サポートされているオペレーティングシステムと容量

次の表は、サポートされているオペレーティングシステムと容量設定を示しています。

Capacity	Linux	Windows
Fargate	Amazon EBS ボリュームは、プラットフォームバージョン 1.4.0 以降 (Linux) でサポートされます。詳細については、「Amazon ECS 向け Fargate プラットフォームバージョン」を参照してください。	サポートされていません
EC2	Amazon EBS ボリュームは、Amazon ECS 最適化 Amazon マシンイメージ (AMI) を使用した Nitro ベースのインスタンス上でホストされるタスクに対応しています。インスタンスタイプについて、詳細は「Amazon EC2 ユーザーガイド」の「インスタンスタイプ」を参照してください。 Amazon EBS ボリュームは、ECS に最適化された AMI 20231219 以降でサポートされています。詳細については、「Amazon ECS に最適化	Amazon ECS に最適化された Amazon マシンイメージ (AMI) を使用した Nitro ベースのインスタンスでホストされるタスク。インスタンスタイプについて、詳細は「Amazon EC2 ユーザーガイド」の「インスタンスタイプ」を参照してください。 Amazon EBS ボリュームは、ECS に最適化された AMI 20241017 以降でサポートされています。詳細については、「Amazon ECS に最適化された Windows AMI メタデー

Capacity	Linux	Windows
	された AMI メタデータを取得する」を参照してください。	タを取得する」を参照してください。
Amazon ECS マネージドインスタンス	Amazon EBS ボリュームは、Linux の Amazon ECS マネージドインスタンス上でホストされるタスクに対応しています。	サポートされていません

考慮事項

Amazon EBS ボリュームを使用する場合には、以下の点を考慮してください。

- Amazon EBS ボリュームは、 `use1-az3` アベイラビリティーゾーン内の Fargate Amazon ECS タスクへのアタッチメント向けに設定することはできません。
- マグネティックタイプ (`standard`) の Amazon EBS ボリュームは、Fargate でホストされるタスクではサポートされていません。Amazon EBS ボリュームタイプについて、詳細は「Amazon EC2 ユーザーガイド」の「Amazon EBS ボリューム」を参照してください。
- Amazon ECS インフラストラクチャ IAM ロールは、デプロイ時にボリュームを設定するサービスまたはスタンドアロンタスクを作成するときに必要です。AWS マネージド型 `AmazonECSInfrastructureRolePolicyForVolumes` IAM ポリシーをロールにアタッチしてもよいし、あるいはマネージド型ポリシーをガイドとして使用して、所定のニーズを満たすアクセス許可を持つ独自のポリシーを作成してアタッチすることもできます。詳細については、「Amazon ECS インフラストラクチャ IAM ロール」を参照してください。
- 各 Amazon ECS タスクには最大 1 つの Amazon EBS ボリュームをアタッチできますが、新しいボリュームである必要があります。既存の Amazon EBS ボリュームをタスクにアタッチすることはできません。ただし、既存のボリュームのスナップショットを使用してデプロイ時に新しい Amazon EBS ボリュームを設定することはできます。
- Amazon ECS サービスで Amazon EBS ボリュームを使用するには、デプロイコントローラーが `ECS` である必要があります。このデプロイコントローラーを使用する場合、ローリングとブルー/グリーンデプロイ戦略の両方がサポートされます。
- マウント済みの Amazon EBS ボリュームへタスク内のコンテナが書き込みを行うには、適切なファイルシステムのアクセス許可を所持している必要があります。コンテナ定義で非ルートユーザーを指定すると、Amazon ECS は、指定されたユーザーがボリュームを読み書きできるように

するグループベースのアクセス許可でボリュームを自動的に設定します。ユーザーを指定しない場合、コンテナはルートとして実行され、ボリュームへのフルアクセスを持ちます。

- Amazon ECS は、アタッチされたボリュームにリザーブドタグ `AmazonECSCreated` および `AmazonECSManaged` を自動で追加します。これらのタグをボリュームから削除すると、Amazon ECS がユーザーに代わってボリュームを管理できなくなりますので注意してください。Amazon EBS ボリュームのタグ付けの詳細については、「Amazon EBS ボリュームのタグ付け」を参照してください。Amazon ECS リソースへのタグ付けの詳細については、「Amazon ECS リソースのタグ付け」を参照してください。
- パーティションを含む Amazon EBS ボリュームのスナップショットからのボリュームのプロビジョニングはサポートされていません。
- サービスによって管理されるタスクにアタッチされたボリュームは保存されず、タスクが終了すると必ず削除されます。
- AWS Outposts で実行される Amazon ECS タスクにアタッチするように Amazon EBS ボリュームを設定することはできません。

非ルートユーザー動作

コンテナ定義で非ルートユーザーを指定すると、Amazon ECS は、指定されたユーザーがボリュームを読み書きできるようにするグループベースのアクセス許可で Amazon EBS ボリュームを自動的に設定します。ボリュームは、次の特性でマウントされます。

- ボリュームは、ルートユーザーとルートグループによって所有されます。
- グループのアクセス許可は、読み取りおよび書き込みアクセスを許可するように設定されています。
- 非ルートユーザーは、ボリュームにアクセスするために適切なグループに追加されます。

非ルートコンテナで Amazon EBS ボリュームを使用する場合は、次のベストプラクティスに従ってください。

- 安全なアクセス許可を保証するために、コンテナイメージ全体で使用するユーザー ID (UID) とグループ ID (GID) を統一してください。
- コンテナイメージにマウントポイントディレクトリを事前作成することで、適切な所有権とアクセス許可を設定します。
- 開発環境で Amazon EBS ボリュームを使用してコンテナをテストし、ファイルシステムのアクセス許可が期待どおりに機能することを確認します。

- 同じタスク内の複数のコンテナがボリュームを共有する場合は、互換性のある UID/GID を使用するか、一貫したアクセス期待に従ってボリュームをマウントしてください。

ボリューム設定を Amazon ECS のタスク定義の起動時刻まで延期する

Amazon EBS ボリュームを設定してタスクにアタッチするには、タスク定義でマウントポイント設定を指定して、ボリュームに名前を付ける必要があります。また、Amazon EBS ボリュームはタスク定義内でアタッチするようには設定ができないため、`configuredAtLaunch` を `true` に設定する必要があります。そのため、Amazon EBS ボリュームはデプロイ時にアタッチするように設定します。

AWS Command Line Interface (AWS CLI) を使用してタスク定義を登録するには、テンプレートを JSON ファイルとして保存し、そのファイルを `register-task-definition` コマンドの入力として渡します。

AWS マネジメントコンソールを使用してタスク定義を作成および登録するには、「コンソールを使用した Amazon ECS タスク定義の作成」を参照してください。

以下のタスク定義に、タスク定義内の `mountPoints` および `volumes` オブジェクト用構文を示します。タスク定義パラメータの詳細については、「Fargate での Amazon ECS タスク定義パラメータ」を参照してください。この例を実行するには、*user input placeholders* をユーザー自身の情報に置き換えます。

Linux

```
{
    "family": "mytaskdef",
    "containerDefinitions": [
        {
            "name": "nginx",
            "image": "public.ecr.aws/nginx/nginx:latest",
            "networkMode": "awsvpc",
           "portMappings": [
                {
                    "name": "nginx-80-tcp",
                    "containerPort": 80,
                    "hostPort": 80,
                    "protocol": "tcp",
                    "appProtocol": "http"
                }
            ],
```

```
            "mountPoints": [
                {
                    "sourceVolume": "myEBSVolume",
                    "containerPath": "/mount/ebs",
                    "readOnly": true
                }
            ]
        }
    ],
    "volumes": [
        {
            "name": "myEBSVolume",
            "configuredAtLaunch": true
        }
    ],
    "requiresCompatibilities": [
        "FARGATE", "EC2"
    ],
    "cpu": "1024",
    "memory": "3072",
    "networkMode": "awsvpc"
}
```

Windows

```
{
    "family": "mytaskdef",
     "memory": "4096",
     "cpu": "2048",
    "family": "windows-simple-iis-2019-core",
    "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
    "runtimePlatform": {"operatingSystemFamily": "WINDOWS_SERVER_2019_CORE"},
    "requiresCompatibilities": ["EC2"]
    "containerDefinitions": [
        {
             "command": ["New-Item -Path C:\\inetpub\\wwwroot\\index.html -Type file
 -Value '<html> <head> <title>Amazon ECS Sample App</title> <style>body {margin-top:
 40px; background-color: #333;} </style> </head><body> <div style=color:white;text-
align:center> <h1>Amazon ECS Sample App</h1> <h2>Congratulations!</h2> <p>Your
 application is now running on a container in Amazon ECS.</p>'; C:\\ServiceMonitor.exe
 w3svc"],
            "entryPoint": [
                "powershell",
```

```
                "-Command"
            ],
            "essential": true,
            "cpu": 2048,
            "memory": 4096,
            "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-
ltsc2019",
            "name": "sample_windows_app",
            "portMappings": [
                {
                    "hostPort": 443,
                    "containerPort": 80,
                    "protocol": "tcp"
                }
            ],
            "mountPoints": [
                {
                    "sourceVolume": "myEBSVolume",
                    "containerPath": "drive:\ebs",
                    "readOnly": true
                }
            ]
        }
    ],
    "volumes": [
        {
            "name": "myEBSVolume",
            "configuredAtLaunch": true
        }
    ],
    "requiresCompatibilities": [
        "FARGATE", "EC2"
    ],
    "cpu": "1024",
    "memory": "3072",
    "networkMode": "awsvpc"
}
```

`mountPoints`

タイプ: オブジェクト配列

必須: いいえ

コンテナでのデータボリュームのマウントポイント。このパラメータは creat-container Docker API の `Volumes` にマッピングされ、docker run の `--volume` オプションにマッピングされます。

Windows コンテナは `$env:ProgramData` と同じドライブに全部のディレクトリをマウントできます。Windows コンテナは、別のドライブにディレクトリをマウントすることはできません。また、マウントポイントは複数のドライブにまたがることはできません。Amazon EBS ボリュームを Amazon ECS タスクに直接アタッチするには、マウントポイントを指定する必要があります。

`sourceVolume`

タイプ: 文字列

必須: はい (`mountPoints` を使用する場合)

マウントするボリュームの名前。

`containerPath`

型: 文字列

必須: はい (`mountPoints` を使用する場合)

ボリュームをマウントするコンテナ内のパス。

`readOnly`

タイプ: ブール値

必須: いいえ

この値が `true` の場合、コンテナはボリュームへの読み取り専用アクセスを許可されます。この値が `false` の場合、コンテナはボリュームに書き込むことができます。デフォルト値は `false` です。

Windows オペレーティングシステムを実行している EC2 インスタンスで実行されるタスクの場合、 値をデフォルト の `false` のままにします。

`name`

タイプ: 文字列

必須: いいえ

ボリュームの名前。最大 255 文字の英字 (大文字と小文字の区別あり)、数字、ハイフン (-)、アンダースコア (_) を使用できます。この名前は、コンテナ定義 `mountPoints` オブジェクトの `sourceVolume` パラメータで参照されます。

`configuredAtLaunch`

タイプ: ブール値

必須: はい。EBS ボリュームをタスクに直接アタッチしたい場合には必須です。

起動時にボリュームを設定可能にするかどうかを指定します。`true` に設定すると、スタンドアロンタスクを実行するとき、またはサービスを作成または更新するときにボリュームを設定できます。`false` に設定すると、タスク定義内で別のボリューム設定を提供することはできません。タスクにアタッチする Amazon EBS ボリュームを設定するには、このパラメータを `true` に設定する必要があります。

Amazon ECS タスクにアタッチされた Amazon EBS ボリュームに保存されているデータの暗号化

AWS Key Management Service (AWS KMS) を使用して、データを保護する暗号化キーを作成および管理できます。Amazon EBS ボリュームは、保管時には AWS KMS keys を使用して暗号化されます。以下の種類のデータが暗号化されます。

- ボリュームに保管されているデータ
- ディスク I/O
- ボリュームから作成されるスナップショット
- 暗号化されたスナップショットから作成された新しいボリューム

タスクにアタッチされている Amazon EBS ボリュームは、エイリアス `alias/aws/ebs` の付いたデフォルトのAWS マネージドキー か、もしくはボリューム設定に指定されているカスタマーマネージド対称キーを使用して暗号化できます。デフォルトの AWS マネージドキー はそれぞれの AWS リージョン ごとの AWS アカウント に対して固有で、自動的に作成されます。カスタマーマネージド型の対称キーを作成するには、「AWS KMS デベロッパーガイド」の「対称暗号化 KMS キーを作成する」に記載の手順に従ってください。

Amazon EBS 暗号化をデフォルトで設定できます。これにより、特定の AWS リージョン 内のタスクにアタッチされた新しいボリュームはすべて、アカウントに設定した KMS キーを使用して暗号化されます。Amazon EBS の暗号化およびデフォルトの暗号化の詳細については、「Amazon EBS ユーザーガイド」の「Amazon EBS の暗号化」を参照してください。

Amazon ECS マネージドインスタンスの動作

デフォルトで暗号化を有効にするか、暗号化したいボリュームの作成時に都度有効化することで、Amazon EBS ボリュームを暗号化することが可能です。(アカウントレベルで) デフォルトでの暗号化を有効にする方法について、詳細は「Amazon EBS ユーザーガイド」の「暗号化をデフォルトで有効にする」を参照してください。

これらのキーは任意の組み合わせで設定できます。KMS キーの優先順位は次のとおりです。

1. ボリューム設定で指定された KMS キー。ボリューム設定で KMS キーを指定すると、Amazon EBS のデフォルトとアカウントレベルで指定した KMS キーがオーバーライドされます。
2. アカウントレベルで指定された KMS キー。Amazon ECS マネージドストレージのクラスターレベルの暗号化に KMS キーを指定すると、Amazon EBS のデフォルトの暗号化はオーバーライドされますが、ボリューム設定に指定した KMS キーはオーバーライドされません。
3. Amazon EBS のデフォルトの暗号化。デフォルトの暗号化は、アカウントレベルの KMS キーまたはボリューム設定でキーを指定しない場合に適用されます。Amazon EBS 暗号化をデフォルトで有効にすると、デフォルトの暗号化用に指定した KMS キーがデフォルトになります。それ以外の場合、デフォルトはエイリアス `alias/aws/ebs` の AWS マネージドキー になります。

> **Note**
>
> `encrypted` をボリューム設定の `false` に設定し、アカウントレベルの KMS キーを指定せずにデフォトで Amazon EBS 暗号化を有効にすると、そのボリュームはデフォルトで Amazon EBS 暗号化に指定されたキーで暗号化されます。

Amazon ECS 以外のマネージドインスタンスの動作

クラスターを作成または更新する時に、Amazon ECS マネージドストレージの Amazon ECS クラスターレベルの暗号化を設定することもできます。クラスターレベルの暗号化はタスクレベルで適用され、指定された KMS キーを使用することで、特定のクラスター上で実行する各タスクにアタッチされた Amazon EBS ボリュームの暗号化を行うことができます。各タスクに対するクラスターレベルでの暗号化設定について、詳細は「Amazon ECS API リファレンス」の「マネージドストレージ設定」を参照してください。

これらのキーは任意の組み合わせで設定できます。KMS キーの優先順位は次のとおりです。

1. ボリューム設定で指定された KMS キー。ボリューム設定で KMS キーを指定すると、Amazon EBS のデフォルトとクラスターレベルで指定した KMS キーがオーバーライドされます。

2. クラスターレベルで指定された KMS キー。Amazon ECS マネージドストレージのクラスターレベルの暗号化に KMS キーを指定すると、Amazon EBS のデフォルトの暗号化はオーバーライドされますが、ボリューム設定に指定した KMS キーはオーバーライドされません。
3. Amazon EBS のデフォルトの暗号化。デフォルトの暗号化は、クラスターレベルの KMS キーまたはボリューム設定でキーを指定しない場合に適用されます。Amazon EBS 暗号化をデフォルトで有効にすると、デフォルトの暗号化用に指定した KMS キーがデフォルトになります。それ以外の場合、デフォルトはエイリアス `alias/aws/ebs` の AWS マネージドキー になります。

> **Note**
>
> `encrypted` を ボリューム設定 の `false` に設定し、クラスターレベルの KMS キーを指定せずにデフォトで Amazon EBS 暗号化を有効にすると、そのボリュームはデフォルトで Amazon EBS 暗号化に指定されたキーで暗号化されます。

カスタマーマネージド型 KMS キーのポリシー

カスタマーマネージド型キーを使用してタスクにアタッチされている EBS ボリュームを暗号化するには、KMS キーポリシーを設定して、ボリューム設定に使用する IAM ロールにキーの使用に必要な権限を付与する必要があります。キーポリシーには、`kms:CreateGrant` と `kms:GenerateDataKey*` 両方のアクセス許可を含める必要があります。スナップショットを使用して作成されたボリュームを暗号化するには、`kms:ReEncryptTo` および `kms:ReEncryptFrom` のアクセス許可が必要です。アタッチする新しい空のボリュームのみを設定して暗号化する場合は、`kms:ReEncryptTo` および `kms:ReEncryptFrom` のアクセス許可は除外できます。

以下の JSON スニペットは、KMS キーポリシーにアタッチするキーポリシーステートメントを示します。これらのステートメントにより、Amazon ECS がキーを使用して EBS ボリュームを暗号化できるようになります。ここに挙げたポリシーステートメントを実行するには、*`user input placeholders`* をユーザー自身の情報に置き換えます。他の場合と同様に、必要な権限のみを設定してください。

```
{
      "Effect": "Allow",
      "Principal": { "AWS": "arn:aws:iam::111122223333:role/ecsInfrastructureRole" },
      "Action": "kms:DescribeKey",
      "Resource":"*"
    },
    {
      "Effect": "Allow",
```

```
      "Principal": { "AWS": "arn:aws:iam::111122223333:role/ecsInfrastructureRole" },
      "Action": [
      "kms:GenerateDataKey*",
      "kms:ReEncryptTo",
      "kms:ReEncryptFrom"
      ],
      "Resource":"*",
      "Condition": {
        "StringEquals": {
          "kms:CallerAccount": "aws_account_id",
          "kms:ViaService": "ec2.region.amazonaws.com"
        },
        "ForAnyValue:StringEquals": {
          "kms:EncryptionContextKeys": "aws:ebs:id"
        }
      }
    },
    {
      "Effect": "Allow",
      "Principal": { "AWS": "arn:aws:iam::111122223333:role/ecsInfrastructureRole" },
      "Action": "kms:CreateGrant",
      "Resource":"*",
      "Condition": {
        "StringEquals": {
          "kms:CallerAccount": "aws_account_id",
          "kms:ViaService": "ec2.region.amazonaws.com"
        },
        "ForAnyValue:StringEquals": {
          "kms:EncryptionContextKeys": "aws:ebs:id"
        },
        "Bool": {
          "kms:GrantIsForAWSResource": true
        }
      }
    }
```

キーポリシーとアクセス許可について、詳しくは「AWS KMS デベロッパーガイド」の「AWS KMS のキーポリシー」および「AWS KMS のアクセス許可」を参照してください。キーのアクセス許可に関連する EBS ボリュームアタッチメントのトラブルシューティングについては、「Amazon ECS タスクへの Amazon EBS ボリュームのアタッチに関するトラブルシューティング」を参照してください。

Amazon ECS のデプロイ時に Amazon EBS ボリューム設定を指定する

`configuredAtLaunch` パラメータを `true` に設定してタスク定義を登録すると、デプロイ時にスタンドアロンタスクを実行するとき、またはサービスを作成または更新するときに Amazon EBS ボリュームを設定できます。`configuredAtLaunch` パラメータを使用してボリューム設定を起動時間まで延長する方法について、詳細は「ボリューム設定を Amazon ECS のタスク定義の起動時刻まで延期する」を参照してください。

ボリュームを設定するには、Amazon ECS API を使用するか、あるいは次の AWS CLI コマンドの入力として JSON ファイルを渡します。

- スタンドアロン ECS タスクを実行するための `run-task`。
- 特定のコンテナインスタンスでスタンドアロン ECS タスクを実行するための `start-task`。このコマンドは Fargate のタスクには適用されません。
- 新しい ECS サービスを作成するための `create-service`。
- 既存のサービスを更新するための `update-service`。

> **Note**
>
> マウント済みの Amazon EBS ボリュームへタスク内のコンテナが書き込みを行うには、適切なファイルシステムのアクセス許可を所持している必要があります。コンテナ定義で非ルートユーザーを指定すると、Amazon ECS は、指定されたユーザーがボリュームを読み書きできるようにするグループベースのアクセス許可でボリュームを自動的に設定します。ユーザーを指定しない場合、コンテナはルートとして実行され、ボリュームへのフルアクセスを持ちます。

あるいは、AWS マネジメントコンソール を使用して Amazon EBS ボリュームを設定することもできます。詳細については、Amazon ECS タスクとしてのアプリケーションの実行、Amazon ECS のローリング更新デプロイの作成、および Amazon ECS サービスを更新する を参照してください。

以下の JSON スニペットは、デプロイ時に設定可能なすべての Amazon EBS ボリュームパラメータを示します。ボリューム設定でこれらのパラメータを使用するには、*user input placeholders* をユーザー自身の情報に置き換えます。これらのパラメータの詳細については、「ボリュームの設定」を参照してください。

```
"volumeConfigurations": [
        {
```

```
            "name": "ebs-volume",
            "managedEBSVolume": {
                "encrypted": true,
                "kmsKeyId": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
                "volumeType": "gp3",
                "sizeInGiB": 10,
                "snapshotId": "snap-12345",
                "volumeInitializationRate":100,
                "iops": 3000,
                "throughput": 125,
                "tagSpecifications": [
                    {
                        "resourceType": "volume",
                        "tags": [
                            {
                                "key": "key1",
                                "value": "value1"
                            }
                        ],
                        "propagateTags": "NONE"
                    }
                ],
                "roleArn": "arn:aws:iam::1111222333:role/ecsInfrastructureRole",
                 "terminationPolicy": {
                    "deleteOnTermination": true//can't be configured for service-
managed tasks, always true
                },
                "filesystemType": "ext4"
            }
        }
    ]
```

> **Important**
>
> 設定内で指定する volumeName は、タスク定義で指定する volumeName と同じであること。

ボリュームアタッチメントのステータスの確認については、「Amazon ECS タスクへの Amazon EBS ボリュームのアタッチに関するトラブルシューティング」を参照してください。EBS ボリュームのアタッチメントに必要な Amazon ECS インフラストラクチャ AWS Identity and Access

Management (IAM) ロールの詳細については、「Amazon ECS インフラストラクチャ IAM ロール」を参照してください。

Amazon EBS ボリュームの設定内容を示す JSON スニペットの例を以下に挙げます。これらの例は、スニペットを JSON ファイルに保存し、そのファイルを AWS CLI コマンドのパラメータ (`--cli-input-json file://filename` パラメータを使用) として渡すことで使用できます。*user input placeholders* を、ユーザー自身の情報に置き換えます。

スタンドアロンタスク向けボリュームの設定

以下のスニペットは、Amazon EBS ボリュームをスタンドアロンタスクにアタッチする設定の構文を示します。以下の JSON スニペットは、volumeType、sizeInGiB、encrypted、kmsKeyId の各設定を構成するための構文を示します。JSON ファイルに指定された設定を使用して EBS ボリュームを作成し、スタンドアロンタスクにアタッチします。

```
{
   "cluster": "mycluster",
   "taskDefinition": "mytaskdef",
   "volumeConfigurations": [
        {
            "name": "datadir",
            "managedEBSVolume": {
                "volumeType": "gp3",
                "sizeInGiB": 100,
                "roleArn":"arn:aws:iam::1111222333:role/ecsInfrastructureRole",
                "encrypted": true,
                "kmsKeyId":
 "arn:aws:kms:region:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
            }
        }
   ]
}
```

サービス作成時のボリューム設定

以下のスニペットは、サービスで管理されるタスクにアタッチする Amazon EBS ボリュームを設定する構文を示します。ボリュームは、snapshotId パラメータを使用して指定されたスナップショットから 200 MiB/秒のレートで取得されます。JSON ファイルに指定された設定を使用して EBS ボリュームを作成し、サービスで管理される各タスクにアタッチします。

```
{
   "cluster": "mycluster",
```

```
    "taskDefinition": "mytaskdef",
    "serviceName": "mysvc",
    "desiredCount": 2,
    "volumeConfigurations": [
         {
             "name": "myEbsVolume",
             "managedEBSVolume": {
               "roleArn":"arn:aws:iam::1111222333:role/ecsInfrastructureRole",
               "snapshotId": "snap-12345",
               "volumeInitializationRate": 200
             }
         }
    ]
}
```

サービス更新時のボリューム設定

以下の JSON スニペットは、これまでタスクへのアタッチ用に Amazon EBS ボリュームが設定されていなかったサービスを更新するための構文を示します。configuredAtLaunch を true に設定したタスク定義リビジョンの ARN を提供する必要があります。以下の JSON スニペットは、volumeType、sizeInGiB、throughput、iops、および filesystemType 設定を構成する構文を示します。これらの設定を使用して EBS ボリュームを作成し、サービスで管理される各タスクにアタッチします。

```
{
    "cluster": "mycluster",
    "taskDefinition": "mytaskdef",
    "service": "mysvc",
    "desiredCount": 2,
    "volumeConfigurations": [
         {
             "name": "myEbsVolume",
             "managedEBSVolume": {
               "roleArn":"arn:aws:iam::1111222333:role/ecsInfrastructureRole",
                "volumeType": "gp3",
                 "sizeInGiB": 100,
                  "iops": 3000,
                "throughput": 125,
                "filesystemType": "ext4"
             }
         }
    ]
```

```
}
```

Amazon EBS ボリュームを使用しないようにサービスを設定する

以下の JSON スニペットは、Amazon EBS ボリュームをそれ以上使用しないようにサービスを更新するための構文を示します。configuredAtLaunch を false に設定したタスク定義の ARN 、または configuredAtLaunch パラメータのないタスク定義を提供する必要があります。空の volumeConfigurations オブジェクトも提供する必要があります。

```
{
   "cluster": "mycluster",
   "taskDefinition": "mytaskdef",
   "service": "mysvc",
   "desiredCount": 2,
   "volumeConfigurations": []
}
```

Amazon EBS ボリュームの終了ポリシー

Amazon ECS タスクが終了すると、Amazon ECS は deleteOnTermination の値に基づいて、終了したタスクに関連付けられている Amazon EBS ボリュームを削除する必要があるかどうかを判断します。デフォルトでは、タスクにアタッチされている EBS ボリュームは、タスクが終了すると削除されます。スタンドアロンタスクの場合、タスク終了時にボリュームを保存するように設定を変更することができます。

> **Note**
>
> サービスにより管理されるタスクにアタッチされたボリュームは保存されず、タスク終了時に必ず削除されます。

Amazon EBS ボリュームのタグ付け

tagSpecifications オブジェクトを使用して Amazon EBS ボリュームにタグを付けることができます。このオブジェクトを使用すると、ボリュームがスタンドアロンタスクにアタッチされているか、あるいはサービス内のタスクにアタッチされているかに応じて、独自のタグを提供したり、あるいはタスク定義またはサービスからタグを伝播するよう設定することができます。1 つのボリュームにアタッチできるタグの最大数は 50 個です。

> **Important**
>
> Amazon ECS は、AmazonECSCreated および AmazonECSManaged のリザーブドタグを Amazon EBS ボリュームに自動的にアタッチするので、1 つのボリュームにユーザーが追加できるタグの最大数は 48 個になります。追加できるタグは、ユーザー定義タグ、ECS マネージド型タグ、または伝播されたタグのいずれです。

Amazon ECS マネージド型タグをボリュームに追加する場合は、UpdateService、CreateService、RunTask、StartTask の呼び出しの enableECSManagedTags を true に設定する必要があります。Amazon ECS マネージド型タグをオンにすると、Amazon ECS はボリュームにクラスターとサービスの情報 (aws:ecs:*clusterName* と aws:ecs:*serviceName*) を自動的にタグ付けします。Amazon ECS リソースへのタグ付けの詳細については、「Amazon ECS リソースのタグ付け」を参照してください。

以下の JSON スニペットは、サービス内の各タスクにアタッチされた Amazon EBS ボリュームにユーザー定義タグを付ける構文を示します。この例を使用してサービスを作成するには、*user input placeholders* をユーザー自身の情報に置き換えます。

```
{
   "cluster": "mycluster",
   "taskDefinition": "mytaskdef",
   "serviceName": "mysvc",
   "desiredCount": 2,
   "enableECSManagedTags": true,
   "volumeConfigurations": [
        {
            "name": "datadir",
            "managedEBSVolume": {
                "volumeType": "gp3",
                "sizeInGiB": 100,
                 "tagSpecifications": [
                    {
                        "resourceType": "volume",
                        "tags": [
                            {
                                "key": "key1",
                                "value": "value1"
                            }
                        ],
```

```
                        "propagateTags": "NONE"
                    }
                ],
                "roleArn":"arn:aws:iam:1111222333:role/ecsInfrastructureRole",
                "encrypted": true,
                "kmsKeyId":
 "arn:aws:kms:region:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
            }
        }
    ]
}
```

> **Important**
>
> Amazon EBS ボリュームにタグを付けるには、volume リソースタイプを指定する必要があります。

Fargate オンデマンドタスク向け Amazon EBS ボリュームのパフォーマンス

Fargate オンデマンドタスクで使用できる Amazon EBS ボリュームのベースライン IOPS とスループットは、タスクにリクエストする CPU ユニット数の合計によって異なります。Fargate タスクに 0.25 ユニット、0.5 ユニット、または 1 ユニットの仮想 CPU (vCPU) をリクエストする場合は、汎用 SSD ボリューム (gp2 または gp3) もしくはハードディスクドライブ (HDD) ボリューム (st1 または sc1) を設定することをお勧めします。Fargate タスク向けに 1 ユニットを超える vCPU をリクエストする場合、タスクにアタッチされる Amazon EBS ボリュームには以下のベースラインパフォーマンス制限が適用されます。EBS のパフォーマンスは、一時的には以下の制限値よりも高くなる場合がありますが、これらの制限値に基づいてワークロードを計画することをお勧めします。

リクエストされる CPU (vCPU) ユニット数	Amazon EBS のベースライン IOPS (16 KiB I/O)	Amazon EBS のベースラインスループット (MiBps、128 KiB I/O)	ベースライン帯域幅 (Mbps)
2	3,000	75	360
4	5,000	120	1,150
8	10,000	250	2,300

リクエストされる CPU (vCPU) ユニット数	Amazon EBS のベースライン IOPS (16 KiB I/O)	Amazon EBS のベースラインスループット (MiBps、128 KiB I/O)	ベースライン帯域幅 (Mbps)
16	15,000	500	4,500

> **Note**
>
> Fargate タスクにアタッチする Amazon EBS ボリュームを設定すると、その Fargate タスク向け Amazon EBS パフォーマンス制限は、タスクの一時ストレージとアタッチされたボリューム間で共有されます。

EC2 タスクの Amazon EBS ボリュームのパフォーマンス

Amazon EBS の提供するボリュームタイプはそれぞれパフォーマンス特性や料金が異なるため、アプリケーションのニーズに応じてストレージのパフォーマンスやコストを調整してください。ボリュームごとの IOPS やスループットなどのパフォーマンスについて、詳細は「Amazon Elastic Block Store ユーザーガイド」の「Amazon EBS ボリュームタイプ」を参照してください。

Amazon ECS マネージドインスタンスタスクの Amazon EBS ボリュームのパフォーマンス

Amazon EBS の提供するボリュームタイプはそれぞれパフォーマンス特性や料金が異なるため、アプリケーションのニーズに応じてストレージのパフォーマンスやコストを調整してください。ボリュームごとの IOPS やスループットなどのパフォーマンスについて、詳細は「Amazon Elastic Block Store ユーザーガイド」の「Amazon EBS ボリュームタイプ」を参照してください。

Amazon ECS タスクへの Amazon EBS ボリュームのアタッチに関するトラブルシューティング

場合により、Amazon EBS ボリュームの Amazon ECS タスクへのアタッチをトラブルシューティングまたは検証しなければならない場合があります。

ボリュームのアタッチメントステータスを確認する

AWS マネジメントコンソールを使用して、Amazon EBS ボリュームの Amazon ECS タスクへのアタッチのステータスを確認できます。タスクが開始されたがアタッチメントが機能しない場合は、表示されるステータス理由を確認してトラブルシューティングに活用してください。作成されたボ

リュームは削除され、タスクは停止します。ステータス理由の詳細については、「Amazon ECS タスクへの Amazon EBS ボリュームアタッチメントのステータス理由」を参照してください。

コンソール上でボリュームのアタッチステータスとステータス理由を確認するには

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
2. [クラスター] ページで、タスクが実行されているクラスターを選択します。クラスターの詳細ページが表示されます。
3. クラスターの詳細ページで、[タスク] タブを選択します。
4. ボリュームアタッチメントステータスを確認したいタスクを選択します。確認したいタスクが停止している場合には、[表示するステータスを絞り込む] で [停止] を選択しなければならない場合があります。
5. タスクの詳細ページで、[ボリューム] タブを選択します。Amazon EBS ボリュームのアタッチメントステータスは、[アタッチメントステータス] で確認できます。ボリュームがタスクにアタッチされない場合は、[アタッチメントのステータス] でステータスを選択して、失敗の原因を表示できます。

または、DescribeTasks API を使用して、タスクのボリュームアタッチメントステータスおよび関連するステータス理由を確認することもできます。

サービスとタスクの失敗

Amazon EBS ボリュームに固有ではないサービスまたはタスクの失敗が発生して、ボリュームのアタッチメントに影響を及ぼす場合があります。詳細については、以下を参照してください。

- サービスイベントメッセージ
- 停止したタスクのエラーコード
- API 失敗の理由

コンテナが Amazon EBS ボリュームに書き込めない

適切なアクセス許可のない非ルートユーザー

コンテナ定義で非ルートユーザーを指定すると、Amazon ECS は書き込みアクセスを許可するグループベースのアクセス許可でボリュームを自動的に設定します。アクセス許可の問題がまだ解決しない場合:

- `uid:gid` フォーマット (例: `1001:1001`) を使用して、`user` パラメータがコンテナ定義で正しく指定されていることを確認してください。
- ボリュームのマウント後に、ユーザーのアクセス許可がコンテナイメージに上書きされないようにしてください。
- コンテナログを調べるか、Amazon ECS Exec を使用して実行中のコンテナを調べることで、アプリケーションが想定されたユーザー ID で実行されていることを確認します。

アクセス許可の問題があるルートユーザー

コンテナ定義でユーザーの指定がされていない場合、コンテナはルートとして実行され、ボリュームへのフルアクセスが必要になります。問題が発生した場合:

- コンテナ内のマウントポイントを確認して、ボリュームが正しくマウントされていることを確認します。
- マウントポイント設定で、ボリュームの設定が読み取り専用でないことを確認してください。

ユーザーが異なる複数コンテナタスク

複数のコンテナが異なるユーザーとして実行されるタスクでは、Amazon ECS はグループアクセス許可を自動的に管理し、指定されたすべてのユーザーのボリュームへの書き込みを許可します。コンテナが書き込めない場合:

- 書き込みアクセスを必要とするすべてのコンテナに `user` パラメータが正しく設定されていることを確認してください。
- ボリュームへのアクセスが必要なすべてのコンテナにボリュームがマウントされていることを確認してください。

コンテナ定義でユーザーを設定する方法について、詳細は「Fargate の Amazon ECS タスク定義パラメータ」を参照してください。

Amazon ECS タスクへの Amazon EBS ボリュームアタッチメントのステータス理由

Amazon ECS タスクにアタッチする Amazon EBS ボリュームを設定する際に AWS マネジメントコンソール上にステータス理由として表示されるエラーを解決するには、以下の情報を参照してください。コンソール上でこれらのステータス理由を見つける方法については、「ボリュームのアタッチメントステータスを確認する」を参照してください。

ECS は、設定された ECS インフラストラクチャロール「arn:aws:iam::*111122223333*:role/*ecsInfrastructureRole*」を引き受けることができませんでした。渡されるロールが Amazon ECS と適切な信頼関係にあることを確認してください

このステータス理由は以下のシナリオにおいて発生します。

- 必要な信頼ポリシーをアタッチせずに IAM ロールを提供した場合。ロールに必要な信頼ポリシーがない場合、Amazon ECS は提供された Amazon ECS インフラストラクチャ IAM ロールにアクセスできません。タスクが DEPROVISIONING 状態でスタックすることがあります。必要な信頼ポリシーの詳細については、「Amazon ECS インフラストラクチャ IAM ロール」を参照してください。
- IAM ユーザーが、Amazon ECS インフラストラクチャロールを Amazon ECS に渡すアクセス許可を持っていません。タスクが DEPROVISIONING 状態でスタックすることがあります。この問題を回避するには、ユーザーに PassRole 許可を付与します。詳細については、「Amazon ECS インフラストラクチャ IAM ロール」を参照してください。
- IAM ロールが、Amazon EBS ボリュームのアタッチメントに必要なアクセス許可を持っていません。タスクが DEPROVISIONING 状態でスタックすることがあります。Amazon EBS ボリュームをタスクにアタッチするために必要なアクセス許可の詳細については、「Amazon ECS インフラストラクチャ IAM ロール」を参照してください。

> **Note**
>
> このエラーメッセージは、ロールの伝播が遅れている場合にも表示されることがあります。数分待ってからロールを使用しても問題が解決しない場合は、ロールの信頼ポリシーが誤って設定されている可能性があります。

ECS が EBS ボリュームを設定できません。「IdempotentParameterMismatch が発生しました」。「指定したクライアントトークンは、すでに削除されているリソースに関連付けられています。別のクライアントトークンを使用してください。」

以下のような AWS KMS キーシナリオでは、IdempotentParameterMismatch メッセージが表示されることがあります。

- 無効な KMS キー ARN、ID、またはエイリアスが指定されました。このシナリオでは、一見タスクが正常に起動したように見えますが、AWS は KMS キーを非同期で認証するため、タスクは最終的に失敗します。詳細は「Amazon EC2 ユーザーガイド」の「Amazon EBS 暗号化」を参照してください。

- Amazon ECS インフラストラクチャ IAM ロールがそのキーを暗号化に使用することを許可する権限を持たないカスタマーマネージド型キーが提供されました。キーポリシー権限に関する問題を回避するには、「Amazon EBS ボリュームのデータ暗号化」の AWS KMS キーポリシー例を参照してください。

Amazon EBS ボリュームイベントと Amazon ECS タスク状態変更イベントを Amazon CloudWatch グループなどのターゲットに送信するように Amazon EventBridge を設定できます。その後、これらのイベントを使用して、ボリュームアタッチメントに影響を及ぼしている力スタマーマネージド型キーに関する問題を特定することができます。詳細については、以下を参照してください。

- AWS re: Post 上の「EventBridge ルールのターゲットとして使用する CloudWatch ロググループをどのように作成できますか?」。
- タスク状態変更イベント。
- Amazon EBS の Amazon EventBridge イベント (Amazon EBS ユーザーガイド)。

タスクへの EBS ボリュームアタッチメントの設定中に ECS がタイムアウトしました。

このメッセージは、ファイルシステム形式に関する以下のシナリオにおいて発生します。

- 設定時に指定されたファイルシステム形式は、タスクのオペレーティングシステムと互換性がありません。
- Amazon EBS ボリュームをスナップショットから作成するように設定しましたが、スナップショットのファイルシステム形式にはタスクのオペレーティングシステムとの互換性がありません。スナップショットから作成されたボリュームでは、スナップショットの作成時にボリュームが使用していたのと同じファイルシステムタイプを指定する必要があります。

Amazon ECS コンテナエージェントのログを利用して、EC2 のタスクに関するメッセージのトラブルシューティングを行うことができます。詳細については、「Amazon ECS ログファイルの場所」と「Amazon ECS ログコレクター」を参照してください。

Amazon ECS での Amazon EFS ボリュームの使用

Amazon Elastic File System (Amazon EFS) では、Amazon ECS タスクで使用するためのシンプルでスケーラブルなファイルストレージを提供します。Amazon EFSでは、ストレージ容量は伸縮性があります。この容量は、ファイルの追加や削除に伴い自動的に拡大および縮小されます。アプリケーションは、必要な時点で必要なストレージを確保できます。

Amazon EFS ファイルシステムを Amazon ECS で使用して、コンテナインスタンスのフリート全体のファイルシステムデータをエクスポートできます。これにより、タスクは、配置されているイン

スタンスにかかわらず、同じ永続的ストレージにアクセスできます。ファイルシステムを使用するには、タスク定義からコンテナインスタンスのボリュームマウントを参照する必要があります。

チュートリアルについては、「コンソールを使用した Amazon ECS での Amazon EFS ファイルシステムの設定」を参照してください。

考慮事項

Amazon EFS ボリュームを使用する場合には、以下の点を考慮してください。

- EC2 上で実行されるタスク向けに、Amazon EFS ファイルシステムサポートは、コンテナエージェントバージョン 1.35.0 を搭載した Amazon ECS 最適化 AMI バージョン 20191212 とともに、パブリックプレビューとして追加されました。ただし、Amazon EFS ファイルシステムのサポートは、コンテナエージェントバージョンが 1.38.0 の Amazon ECS 最適化 AMI バージョン 20200319 (Amazon EFS アクセスポイントと IAM 認可機能が含まれるもの) で一般提供されています。これらの機能を使用する際には、Amazon ECS 最適化 AMI バージョン 20200319 以降の使用をお勧めします。詳細については、「Amazon ECS に最適化された Linux AMI」を参照してください。

  > **Note**
  >
  > 独自の AMI を作成する場合、コンテナエージェント 1.38.0 以降、ecs-init バージョン 1.38.0-1 以降を使用し、Amazon EC2 インスタンスで以下のコマンドを実行して、Amazon ECS ボリュームプラグインを有効にする必要があります。コマンドは、ベースイメージとして Amazon Linux 2 と Amazon Linux のどちらを使用しているかによって異なります。
  >
  > Amazon Linux 2
  >
  > ```
  > yum install amazon-efs-utils
  > systemctl enable --now amazon-ecs-volume-plugin
  > ```
  >
  > Amazon Linux
  >
  > ```
  > yum install amazon-efs-utils
  > sudo shutdown -r now
  > ```

- Fargate でホストされるタスクの場合、プラットフォームバージョン 1.4.0 以降 (Linux) で、Amazon EFS ファイルシステムがサポートされます。詳細については、「Amazon ECS 向け Fargate プラットフォームバージョン」を参照してください。

- Fargate でホストされるタスクに Amazon EFS ボリュームを使用する場合、Fargate は Amazon EFS ボリュームの管理を担当するスーパーバイザーコンテナを作成します。スーパーバイザーコンテナは、タスクのメモリと CPU を少量使用します。スーパーバイザーコンテナは、タスクメタデータバージョン 4 エンドポイントにクエリを実行するときに表示されます。さらに、CloudWatch Container Insights では、コンテナ名 `aws-fargate-supervisor` として表示されます。EC2 の使用時について、詳細は「Amazon ECS タスクメタデータエンドポイントバージョン 4」を参照してください。Fargate の使用時について、詳細は「Fargate のタスク用の Amazon ECS タスクメタデータエンドポイントバージョン 4」を参照してください。
- Amazon EFS ボリュームの使用または `EFSVolumeConfiguration` の指定は、外部インスタンスではサポートされていません。
- Amazon EFS ボリュームの使用は、Amazon ECS マネージドインスタンスで実行されるタスクでサポートされています。
- エージェント設定ファイルの `ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION` パラメータは、デフォルトよりも小さい値 (約 1 時間) に設定することをお勧めします。この変更は、EFS マウント認証情報の有効期限切れを防ぎ、使用されていないマウントをクリーンアップするのに役立ちます。 詳細については、 を参照してください。Amazon ECS コンテナエージェントの設定

Amazon EFS アクセスポイントの使用

Amazon EFS アクセスポイントは、EFS ファイルシステムへのアプリケーション固有のエントリポイントです。これにより、共有データセットへのアプリケーションによるアクセスを管理します。Amazon EFS アクセスポイントの詳細およびアクセス制御方法については、Amazon Elastic File System ユーザーガイドの「Amazon EFS アクセスポイントの使用」を参照してください。

アクセスポイントを使用すると、アクセスポイントを介したすべてのファイルシステム要求に対してユーザーアイデンティティ (ユーザーの POSIX グループなど) を適用できます。また、アクセスポイントでは、ファイルシステムに対して別のルートディレクトリを適用することも可能です。クライアントは、指定したディレクトリまたはそのサブディレクトリ内のデータに対してのみアクセスが可能なためです。

> **Note**
>
> EFS アクセスポイントを作成する際は、ファイルシステム上でルートディレクトリとして機能するパスを指定します。Amazon ECS タスク定義内でアクセスポイント ID を持つ EFS ファイルシステムを参照する場合、ルートディレクトリは省略するか、EFS アクセスポイントに設定されたパスを適用するために `/` を設定する必要があります。

Amazon ECS タスクの IAM ロールを使用して、特定のアプリケーションで使用するアクセスポイントを限定できます。IAM ポリシーとアクセスポイントを組み合わせると、アプリケーションによる特定のデータセットへのアクセスを保護できます。タスクIAM ロールの使用方法については、「Amazon ECS タスクの IAM ロール」を参照してください。

Amazon ECS で Amazon EFS ボリュームを使用する場合のベストプラクティス

Amazon ECS で Amazon EFS を使用する場合は、以下のベストプラクティス推奨事項に留意すること。

Amazon EFS ボリュームのセキュリティとアクセスコントロール

Amazon EFS には、Amazon EFS ファイルシステムに保存されているデータのセキュリティを確保し、それを必要とするアプリケーションからのみアクセスを許可するためのアクセス制御機能があります。保管中と転送中両方の暗号化を有効にすることで、データのセキュリティを確保します。詳細については、Amazon Elastic File System ユーザーガイド の「Amazon EFS でのデータの暗号化」を参照してください。

データの暗号化に加えて、Amazon EFS を使用してファイルシステムへのアクセスを制限することもできます。EFS にアクセス制御を実装するには 3 つの方法があります。

- セキュリティグループ — Amazon EFS マウントターゲットでは、ネットワークトラフィックを許可または拒否するのに使用するセキュリティグループを設定できます。Amazon EFS にアタッチされるセキュリティグループを設定することにより、Amazon ECS インスタンス、もしくは `awsvpc` ネットワークモードを使用している場合は Amazon ECS タスクにアタッチされているセキュリティグループからの NFS トラフィック (ポート 2049) を許可できます。
- IAM — IAM を使用して Amazon EFS ファイルシステムへのアクセスを制限できます。Amazon ECS タスクを設定すると、EFS ファイルシステムをマウントするためにファイルシステムにアクセスする IAM ロールが必要になります。詳細は「Amazon Elastic File System ユーザーガイド」の「Using IAM to control file system data access」を参照してください。

  IAM ポリシーでは、Amazon EFS ファイルシステムへの接続時に TLS の使用をクライアントに要求するなど、あらかじめ定義された条件を執行することもできます。詳細は「Amazon Elastic File System ユーザーガイド」の「Amazon EFS condition keys for clients」を参照してください。
- Amazon EFS アクセスポイント — Amazon EFS アクセスポイントとは、アプリケーション固有の Amazon EFS ファイルシステムへのエントリポイントです。アクセスポイントを使用すると、そのアクセスポイントを介したすべてのファイルシステム要求に対してユーザーアイデンティティ (ユーザーの POSIX グループなど) を適用することができます。また、アクセスポイントでは、

ファイルシステムに対して別のルートディレクトリを適用することも可能です。これは、クライアントは指定したディレクトリまたはそのサブディレクトリ内のデータにしかアクセスできないためです。

IAM ポリシー

IAM ポリシーを使用すると、Amazon EFS ファイルシステムへのアクセスを制御できます。

ファイルシステムポリシーを使用してファイルシステムにアクセスするクライアントに対して、以下のアクションを指定できます。

[アクション]	説明
`elasticfilesystem:ClientMount`	ファイルシステムへの読み取り専用アクセス許可を付与します。
`elasticfilesystem:ClientWrite`	ファイルシステムへの書き込みアクセス許可を付与します。
`elasticfilesystem:ClientRootAccess`	ファイルシステムへのアクセス時にルートユーザーの使用を許可します。

ポリシーで各アクションを指定する必要があります。ポリシーは次の方法で定義できます。

- クライアントベース – ポリシーをタスクロールにアタッチします

  タスク定義の作成時に [IAM 認可] オプションを設定します。
- リソースベース – Amazon EFS ファイルシステムにポリシーをアタッチします

  リソースベースのポリシーが存在しない場合、ファイルシステム作成時にデフォルトですべてのプリンシパル (*) にアクセス許可が付与されます。

[IAM 認可] オプションを設定すると、タスクロールに関連付けられたポリシーと Amazon EFS リソースベースのポリシーがマージされます。[IAM 認可] オプションは、このポリシーを持つタスクアイデンティティ (タスクロール) を Amazon EFS に渡します。この結果、Amazon EFS リソースベースのポリシーに、ポリシーで指定された IAM ユーザーまたはロールのコンテキストを入れることができます。このオプションを設定しない場合、Amazon EFS リソースレベルのポリシーは IAM ユーザーを「匿名」として識別します。

セキュリティを最大化するため、Amazon EFS ファイルシステムに 3 つのアクセス制御すべてを実装することを検討してください。たとえば、Amazon EFS マウントポイントにアタッチされるセキュリティグループを設定することで、コンテナインスタンスまたは Amazon ECS タスクに関連付けられたセキュリティグループからの進入 NFS トラフィックのみを許可することができます。さらに、Amazon EFS の設定により、許可されたセキュリティグループからの接続であっても、ファイルシステムにアクセスするための IAM ロールを要求することができます。最後に、Amazon EFS アクセスポイントの使用により、POSIX ユーザー権限を執行してアプリケーション向けルートディレクトリを指定することができます。

以下のタスク定義スニペットは、アクセスポイントを使用して Amazon EFS ファイルシステムをマウントする方法を示します。

```
"volumes": [
    {
      "efsVolumeConfiguration": {
        "fileSystemId": "fs-1234",
        "authorizationConfig": {
          "accessPointId": "fsap-1234",
          "iam": "ENABLED"
        },
        "transitEncryption": "ENABLED",
        "rootDirectory": ""
      },
      "name": "my-filesystem"
    }
]
```

Amazon EFS ボリュームパフォーマンス

Amazon EFS には、汎用と Max I/O の 2 つのパフォーマンスモードがあり、汎用モードは、コンテンツ管理システムや CI/CD ツールなどのレイテンシーの影響を受けやすいアプリケーションに適しています。これとは対照的に、Max I/O ファイルシステムは、データ分析、メディア処理、機械学習などのワークロードに適しています。これらのワークロードは、数百または数千のコンテナからの並列演算を実行する必要があり、可能な限り高い総スループットと IOPS を必要とします。詳細は「Amazon Elastic File System ユーザーガイド」の「Amazon EFS performance modes」を参照してください。

レイテンシーの影響を受けやすいワークロードには、Max I/O パフォーマンスモードによる高い I/O レベルと、汎用パフォーマンスモードによる低レイテンシーの両方が必要です。このタイプのワークロードでは、複数の[General Purpose(汎用)]パフォーマンスモードファイルシステムを作成すること

をお勧めします。この場合、ワークロードとアプリケーションでサポート可能な限り、すべてのファイルシステム全体にアプリケーションのワークロードを分散することをお勧めします。

Amazon EFS ボリュームスループット

すべての Amazon EFS ファイルシステムには、Provisioned Throughput を使用したファイルシステムのプロビジョニングされたスループットの量、または Bursting Throughput を使用したファイルシステムの EFS 標準または One Zone ストレージクラスに保存されたデータの量のいずれかにより決定される、関連するメータリングスループットがあります。詳細は「Amazon Elastic File System ユーザーガイド」の「メータリングスループットについて理解する」を参照してください。

Amazon EFS ファイルシステムのデフォルトのスループットモードはバーストモードです。バーストモードでは、ファイルシステムが利用できるスループットは、ファイルシステムの拡大に応じてスケールインまたはスケールアウトされます。ファイルベースのワークロードは通常スパイクが発生する、つまり、特定の時間のみ高レベルのスループットを必要とし、残りの時間は低レベルのスループットになることが多いため、Amazon EFS は一定期間高いスループットレベルにバーストできるように設計されています。さらに、多くのワークロードは読み取り負荷が高いため、読み取り操作は他の NFS 操作 (書き込みなど) と 1:3 の比率でメータリングされます。

Amazon EFS ファイルシステムは、すべての Amazon EFS 標準ストレージまたは Amazon EFS One Zone ストレージについて 1 TB ごとに 50 MB/秒という安定したベースラインパフォーマンスを提供します。サイズに関係なく、すべてのファイルシステムは 100 MB/秒までバーストできます。1 TB を超える EFS 標準ストレージまたは EFS One Zone ストレージを持つファイルシステムでは、1 TB ごとに 100 MB/秒までバーストできます。読み取り操作は 1:3 の比率でメータリングされるため、読み取りスループットは 1 TiB ごとに最大 300 MiB/秒を駆動できます。ファイルシステムにデータを追加すると、そのファイルシステムで使用できる最大スループットは、Amazon EFS 標準ストレージクラスのストレージに比例して自動的にスケーリングされます。保存されているデータ量で達成できる以上に高いスループットが必要な場合は、ワークロードが必要とする量に合わせてプロビジョンドスループットを設定できます。

ファイルシステムのスループットは、ファイルシステムに接続されたすべての Amazon EC2 インスタンス間で共有されます。たとえば、100 MB/秒のスループットまでバーストできる 1 TB のファイルシステムでは、1 つの Amazon EC2 インスタンスから 10 MB/秒を駆動できます。詳細については、Amazon Elastic File System ユーザーガイドの「Amazon EFS のパフォーマンス」を参照してください。

Amazon EFS ボリュームのコストの最適化

Amazon EFS はストレージのスケーリングを簡素化します。Amazon EFS ファイルシステムは、データを追加すると自動的に拡張されます。特に Amazon EFS の Bursting Throughput モードで

は、Amazon EFS のスループットは、標準ストレージクラスのファイルシステムのサイズが大きくなるにつれて拡大します。EFS ファイルシステムのプロビジョンドスループットに追加費用を支払わずにスループットを向上させるには、Amazon EFS ファイルシステムを複数のアプリケーション間で共有することができます。Amazon EFS アクセスポイントを使用して、共有 Amazon EFS ファイルシステムにストレージ分離を実装できます。そうすることで、アプリケーション間で同じファイルシステムを共有していても、ユーザーが許可しない限りデータにアクセスできなくなります。

データが大きくなると、Amazon EFS ではアクセス頻度の低いファイルを自動的に下位のストレージクラスに移動できます。Amazon EFS 標準の低頻度アクセス (IA) ストレージクラスは、毎日アクセスされないファイルのストレージコストを削減します。これにより、Amazon EFS が提供する高可用性、高耐久性、伸縮性、および POSIX ファイルシステムへのアクセス性が損なわれることはありません。詳細については、「Amazon Elastic File System ユーザーガイド」の「Amazon EFS ストレージクラス」を参照してください。

Amazon EFS ライフサイクルポリシーを使用してアクセス頻度の低いファイルを Amazon EFS IA ストレージに移動することで、自動的にコストを節約することを検討してください。詳細は「Amazon Elastic File System ユーザーガイド」の「Amazon EFS のライフサイクル管理」を参照してください。

Amazon EFS ファイルシステムを作成する場合、Amazon EFS でデータを複数のアベイラビリティーゾーン (標準) に複製するか、あるいは単一のアベイラビリティーゾーン内でデータを冗長的に保存するかを選択できます。Amazon EFS One Zone ストレージクラスは、Amazon EFS 標準ストレージクラスと比較してストレージコストを大幅に削減できます。マルチ AZ の耐障害性を必要としないワークロードには Amazon EFS One Zone ストレージクラスの使用を検討してください。アクセス頻度の低いファイルを Amazon EFS One Zone の低頻度アクセスストレージに移動することで、Amazon EFS One Zone ストレージのコストをさらに削減できます。詳細については、「Amazon EFS 低頻度アクセス」を参照してください。

Amazon EFS ボリュームデータ保護

Amazon EFS は、標準ストレージクラスを使用するファイルシステムについては、複数のアベイラビリティーゾーン間でデータを冗長的に保存します。Amazon EFS One Zone ストレージクラスを選択した場合、単一のアベイラビリティーゾーン内でデータを冗長的に保存します。さらに、Amazon EFS は、1 年間に 99.999999999% (9 が 11 桁) の耐久性を保証するよう設計されています。

どの環境でもそうですが、データのバックアップを取り、誤って削除されないように保護することがベストプラクティスです。Amazon EFS データのベストプラクティスには、正常に機能し定期的にテストされている AWS Backup 型バックアップの使用が含まれます。Amazon EFS One Zone ストレージクラスを使用するファイルシステムは、意図的に無効にしない限り、ファイルシステムの作成

時にデフォルトでファイルを自動的にバックアップするように設定されています。詳細は「Amazon Elastic File System ユーザーガイド」の「EFS ファイルシステムのバックアップ」を参照してください。

タスク定義内での Amazon EFS ファイルシステムの指定

コンテナに Amazon EFS ファイルシステムボリュームを使用するには、タスク定義でボリュームとマウントポイントの設定を指定する必要があります。次のスニペットの JSON によるタスク定義では、コンテナの volumes と mountPoints オブジェクトの構文を示しています。

```
{
    "containerDefinitions": [
        {
            "name": "container-using-efs",
            "image": "public.ecr.aws/amazonlinux/amazonlinux:latest",
            "entryPoint": [
                "sh",
                "-c"
            ],
            "command": [
                "ls -la /mount/efs"
            ],
            "mountPoints": [
                {
                    "sourceVolume": "myEfsVolume",
                    "containerPath": "/mount/efs",
                    "readOnly": true
                }
            ]
        }
    ],
    "volumes": [
        {
            "name": "myEfsVolume",
            "efsVolumeConfiguration": {
                "fileSystemId": "fs-1234",
                "rootDirectory": "/path/to/my/data",
                "transitEncryption": "ENABLED",
                "transitEncryptionPort": integer,
                "authorizationConfig": {
                    "accessPointId": "fsap-1234",
                    "iam": "ENABLED"
                }
```

```
            }
        }
    ]
}
```

`efsVolumeConfiguration`

タイプ: オブジェクト

必須: いいえ

このパラメータは、Amazon EFS ボリュームを使用する場合に指定します。

`fileSystemId`

型: 文字列

必須: はい

使用する Amazon EFS ファイルシステムの ID。

`rootDirectory`

型: 文字列

必須: いいえ

ホスト内にルートディレクトリとしてマウントする Amazon EFS ファイルシステム内のディレクトリ。このパラメータを省略すると、Amazon EFS ボリュームのルートが使用されます。`/` を指定すると、このパラメータを省略した場合と同じ結果になります。

> **Important**
>
> `authorizationConfig` に EFS アクセスポイントを指定する場合は、ルートディレクトリパラメータを省略するか、または `/` に設定して EFS アクセスポイントにパスを設定する必要があります。

`transitEncryption`

型: 文字列

有効な値: `ENABLED` | `DISABLED`

必須: いいえ

Amazon ECS ホストと Amazon EFS サーバー間で、転送中の Amazon EFS データの暗号化を有効にするかどうかを指定します。Amazon EFS IAM 認可を使用する場合は、転送中の暗号化を有効にする必要があります。このパラメータを省略すると、DISABLED のデフォルト値が使用されます。詳細については、Amazon Elastic ファイルシステムユーザーガイドの「転送中データの暗号化」を参照してください。

`transitEncryptionPort`

タイプ: 整数

必須: いいえ

Amazon ECS ホストと Amazon EFS サーバーとの間で、暗号化されたデータを送信するときに使用するポート。転送中の暗号化ポートを指定しないと、Amazon EFS マウントヘルパーが使用するポート選択方式が使用されます。詳細については、[Amazon Elastic File System User Guide] (Amazon Elastic File System ユーザーガイド) の[EFS Mount Helper] (EFS マウントヘルパー) を参照してください。

`authorizationConfig`

タイプ: オブジェクト

必須: いいえ

Amazon EFS ファイルシステムに対する認可構成の詳細。

`accessPointId`

型: 文字列

必須: いいえ

使用するアクセスポイント ID。アクセスポイントを指定する場合は、efsVolumeConfiguration のルートディレクトリ値を省略するか、EFS アクセスポイントに設定されたパスを適用するために / を設定する必要があります。アクセスポイントを使用する場合は、EFSVolumeConfiguration で転送中の暗号化を有効にする必要があります。詳細については、Amazon Elastic ファイルシステムユーザーガイドの Amazon EFS アクセスポイントの使用を参照してください。

`iam`

型: 文字列

有効な値: ENABLED | DISABLED

必須: いいえ

タスク定義で定義した Amazon ECS タスクの IAM ロールを、Amazon EFS ファイルシステムのマウント時に使用するかどうかを指定します。使用する場合は、`EFSVolumeConfiguration` で転送中の暗号化を有効にする必要があります。このパラメータを省略すると、`DISABLED` のデフォルト値が使用されます。詳細については、「タスク用の IAM ロール」を参照してください。

コンソールを使用した Amazon ECS での Amazon EFS ファイルシステムの設定

Amazon ECS で Amazon Elastic File System (Amazon EFS) ファイルシステムを使用する方法について説明します。

ステップ 1: Amazon ECS クラスターを作成する

次の手順に従って Amazon ECS クラスターを作成します。

新しいクラスターを作成するには (Amazon ECS コンソール)

開始する前に、適切な IAM アクセス許可を割り当ててください。詳細については、「the section called “Amazon ECS クラスターの例”」を参照してください。

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションバーから、使用するリージョンを選択します。
3. ナビゲーションペインで [Clusters] (クラスター) を選択します。
4. [Clusters] (クラスター) ページで、[Create Cluster] (クラスターの作成) を選択します。
5. [クラスター設定] の [クラスター名] に、名前として「`EFS-tutorial`」を入力します。
6. (オプション) タスクとサービスが起動する VPC とサブネットを変更するには、[Networking] (ネットワーク) で、次のいずれかのオペレーションを実行します。
   - サブネットを削除するには、[Subnets] (サブネット) で、削除するサブネットごとに [X] を選択します。
   - [デフォルト] VPC 以外の VPC に変更するには、[VPC] で既存の [VPC] を選択し、[サブネット] で各サブネットを選択します。
7. クラスターに Amazon EC2 インスタンスを追加するには、[インフラストラクチャー] を展開して [Amazon EC2 インスタンス] を選択します。次に、キャパシティープロバイダーとして機能する Auto Scaling グループを設定します。

- Auto Scaling グループを作成するには、Auto Scaling group(ASG) (Auto Scaling グループ) から、[Create new group] (新しいグループの作成) を選択し、グループに関する以下の詳細情報を入力します。
  - [オペレーティングシステム/アーキテクチャ] で、[Amazon Linux 2] を選択します。
  - [EC2 instance type] で、[`t2.micro`] を選択します。

    [SSH key pair] (SSH キーペア) を使用する場合、インスタンスに接続する際に ID を証明するペアを選択してください。
  - [キャパシティ] に、「`1`」と入力します。

8. [作成] を選択します。

ステップ 2: Amazon EC2 インスタンスと Amazon EFS ファイルシステムのセキュリティグループを作成する

このステップでは、ポート 80 でのインバウンドネットワークトラフィックを許可する Amazon EC2 インスタンスと、コンテナインスタンスからのインバウンドアクセスを許可する Amazon EFS ファイルシステムのセキュリティグループを作成します。

以下のオプションを使用して Amazon EC2 インスタンスのセキュリティグループを作成します。

- [セキュリティグループ名] — セキュリティグループの名前を入力します。
- [VPC] - クラスター用に前に特定した VPC を選択します。
- インバウンドルール
  - タイプ - HTTP
  - ソース - 0.0.0.0/0

以下のオプションを使用して Amazon EFS ファイルシステムのセキュリティグループを作成します。

- [セキュリティグループ名] — セキュリティグループの名前を入力します。例えば、`EFS-access-for-sg-`*`dc025fa2`*。
- [VPC] - クラスター用に前に特定した VPC を選択します。
- インバウンドルール
  - [タイプ] – [NFS]

- ソース - [カスタム]で、インスタンス用に作成したセキュリティグループの ID を使用します。

セキュリティグループの作成方法について、詳細は「Amazon EC2 ユーザーガイド」の「Amazon EC2 インスタンス用のセキュリティグループの作成」を参照してください。

ステップ 3:Amazon EFS ファイルシステムを作成する

このステップでは、Amazon EFS ファイルシステムを作成します。

Amazon ECS タスク用の Amazon EFS ファイルシステムを作成するには

1. Amazon Elastic File System コンソール (https://console.aws.amazon.com/efs/) を開きます。
2. [Create file system] を選択します。
3. ファイルシステムの名前を入力し、コンテナインスタンスをホストする VPC を選択します。デフォルトでは、指定した VPC の各サブネットに、その VPC のデフォルトセキュリティグループを使用するマウントターゲットが割り当てられます。次に [カスタマイズ] を選択します。

   > **Note**
   >
   > このチュートリアルでは、Amazon EFS ファイルシステム、Amazon ECS クラスター、コンテナインスタンス、およびタスクが同じ VPC 内に存在することを前提としています。別の VPC からファイルシステムをマウントする方法について、詳細は「Amazon EFS ユーザーガイド」の「Walkthrough: Mount a file system from a different VPC」を参照してください。

4. [ファイルシステムの設定] ページでオプション設定を行い、[パフォーマンス設定] でファイルシステムの[バースト] スループットモードを選択します。設定が完了したら、[次へ] を選択します。

   a. (オプション) ファイルシステムのタグを追加します。例えば、ファイルシステムの一意の名前を指定するには、[名前] キーの横にある [値] 列にその名前を入力します。

   b. (オプション) ライフサイクル管理を有効にして、アクセス頻度の低いストレージのコストを節約します。詳細については、Amazon Elastic File System User Guide の「EFS のライフサイクル管理」を参照してください。

   c. (オプション) 暗号化を有効にします。保管時の Amazon EFS ファイルシステムの暗号化を有効にするチェックボックスを選択します。

5. [ネットワークアクセス]ページの [マウントターゲット] で、各アベイラビリティーゾーンの既存のセキュリティグループ設定を、ステップ 2: Amazon EC2 インスタンスと Amazon EFS ファイルシステムのセキュリティグループを作成する でファイルシステム用に作成したセキュリティグループに置き換え、[次へ] を選択します。
6. このチュートリアルでは[ファイルシステムポリシー]を設定する必要はないので、[次へ] を選択すればセクションをスキップできます。
7. ファイルシステムのオプションを確認し、[作成] を選択してプロセスを完了します。
8. [ファイルシステム] 画面から、ファイルシステム ID を書き留めます。次のステップで、Amazon ECS タスク定義で、この値をリファレンスします。

ステップ 4: Amazon EFS ファイルシステムにコンテンツを追加する

このステップでは、Amazon EFS ファイルシステムを Amazon EC2 インスタンスにマウントし、コンテンツを追加します。これは、このチュートリアルでデータの永続的な性質を示すことを目的としたテストです。この機能を使用する場合は、通常、Amazon EFS ファイルシステムにデータを書き込むためのアプリケーションや別の方法があります。

Amazon EC2 インスタンスを作成し、Amazon EFS ファイルシステムをマウントします。

1. Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
2. [インスタンスの作成] を選択してください。
3. [アプリケーションおよび OS イメージ (Amazon マシンイメージ)] で、[Amazon Linux 2 AMI (HVM)] を選択します。
4. [インスタンス タイプ] で、デフォルトのインスタンスタイプ `t2.micro` をそのまま使用します。
5. [キーペア (ログイン)] で、インスタンスへの SSH アクセス用のキーペアを選択します。
6. [ネットワーク設定] で、Amazon EFS ファイルシステムおよび Amazon ECS クラスター用に指定した VPC を選択します。サブネットと、ステップ 2: Amazon EC2 インスタンスと Amazon EFS ファイルシステムのセキュリティグループを作成する で作成したインスタンスセキュリティグループを選択します。インスタンスのセキュリティグループを設定します。[パブリック IP の自動割り当て] が有効になっていることを確認します。
7. [ストレージの設定] で、ファイルシステムの [編集] ボタンを選択し、[EFS] を選択します。ステップ 3:Amazon EFS ファイルシステムを作成する で作成したファイルシステムを選択します。必要に応じて、マウントポイントを変更したり、デフォルト値をそのまま使用したりできます。

> **Important**
>
> インスタンスにファイルシステムを追加する前に、サブネットを選択する必要があります。

8. [セキュリティグループを自動的に作成してアタッチする] をオフにします。もう 1 つのチェックボックスはオンのままにしておきます。[Add shared file system] (共有ファイルシステムの追加) を選択します。
9. [Advanced Details] で、Amazon EFS ファイルシステムのマウント ステップを使用してユーザー データスクリプトが自動的に入力されていることを確認します。
10. [概要] で、[インスタンス数] が 1 であることを確認します。[インスタンスを起動] を選択してください。
11. [インスタンスを起動] ページで、[すべてのインスタンスを表示] を選択して、インスタンスのステータスを表示します。最初、[インスタンスの状態] ステータスは PENDING です。状態が RUNNING に変わり、インスタンスがすべてのステータスチェックに合格すると、インスタンスは使用可能になります。

次に、Amazon EC2 インスタンスに接続し、コンテンツを Amazon EFS ファイルシステムに追加します。

Amazon EC2 インスタンスに接続し、コンテンツを Amazon EFS ファイルシステムに追加するには

1. 作成した Amazon EC2 インスタンスに SSH 接続します。詳細については、「Amazon EC2 ユーザーガイド」の「SSH を使用した Linux インスタンスへの接続」を参照してください。
2. ターミナルウィンドウから、df -T コマンドを実行して、Amazon EFS ファイルシステムがマウントされていることを確認します。次の出力では、Amazon EFS ファイルシステムのマウントを強調表示しています。

```
$ df -T
Filesystem     Type            1K-blocks    Used        Available Use% Mounted on
devtmpfs       devtmpfs           485468       0           485468   0% /dev
tmpfs          tmpfs              503480       0           503480   0% /dev/shm
tmpfs          tmpfs              503480     424           503056   1% /run
tmpfs          tmpfs              503480       0           503480   0% /sys/fs/
cgroup
/dev/xvda1     xfs               8376300 1310952          7065348  16% /
127.0.0.1:/    nfs4     9007199254739968       0 9007199254739968   0% /mnt/efs/fs1
```

```
tmpfs           tmpfs               100700       0           100700    0% /run/
user/1000
```

3. Amazon EFS ファイルシステムのマウント先のディレクトリに移動します。前述の例では、これは /mnt/efs/fs1 です。
4. 以下の内容で index.html という名前のファイルを作成します。

```
<html>
    <body>
        <h1>It Works!</h1>
        <p>You are using an Amazon EFS file system for persistent container
 storage.</p>
    </body>
</html>
```

ステップ 5: タスク定義を作成する

次のタスク定義は、efs-html というデータボリュームを作成します。nginx コンテナは、ホストデータボリュームを NGINX ルート /usr/share/nginx/html にマウントします。

Amazon ECS コンソールを使用して新しいタスク定義を作成するには

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで、タスク定義 を選択します。
3. [Create new task definition] (新しいタスク定義の作成)、[Create new task definition with JSON] (JSON で新しいタスク定義を作成) の順に選択します。
4. JSON エディターボックスには、次の JSON テキストをコピーして貼り付けます。fileSystemId は、実際の Amazon EFS ファイルシステムの ID に置き換えます。

```
{
    "containerDefinitions": [
        {
            "memory": 128,
            "portMappings": [
                {
                    "hostPort": 80,
                    "containerPort": 80,
                    "protocol": "tcp"
                }
            ],
```

```
            "essential": true,
            "mountPoints": [
                {
                    "containerPath": "/usr/share/nginx/html",
                    "sourceVolume": "efs-html"
                }
            ],
            "name": "nginx",
            "image": "public.ecr.aws/docker/library/nginx:latest"
        }
    ],
    "volumes": [
        {
            "name": "efs-html",
            "efsVolumeConfiguration": {
                "fileSystemId": "fs-1324abcd",
                "transitEncryption": "ENABLED"
            }
        }
    ],
    "family": "efs-tutorial",
    "executionRoleArn":"arn:aws:iam::111122223333:role/ecsTaskExecutionRole"
}
```

> **Note**
>
> Amazon ECS タスク実行 IAM ロールでは、Amazon EFS ファイルシステムをマウントするにあたり、特定の Amazon EFS 関連のアクセス許可は必要ありません。Amazon EFS のリソースベースのポリシーが存在しない場合、デフォルトで、ファイルシステム作成時にすべてのプリンシパル (*) にアクセス許可が付与されます。
> Amazon ECS タスクロールは、Amazon ECS タスク定義で「EFS IAM 認可」が有効になっている場合にのみ必要です。有効な場合、Amazon EFS のリソースベースのポリシーで タスクロール ID に対して Amazon EFS ファイルシステムへのアクセスを許可する必要があり、匿名アクセスは無効にする必要があります。

5. [作成] を選択します。

ステップ 6: タスクを実行して結果を表示する

さて、Amazon EFS ファイルシステムが作成され、NGINX コンテナのウェブコンテンツが用意されたので、作成したタスク定義を使用してタスクを実行できます。NGINX ウェブサーバーは、シンプルな HTML ページを提供します。Amazon EFS ファイルシステムのコンテンツを更新した場合、それらの変更は、そのファイルシステムがマウントされているすべてのコンテナにも伝達されます。

タスクは、クラスターのために定義したサブネットで実行されます。

コンソールを使用してタスクを実行し結果を表示するには

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. [Clusters] (クラスター) ページで、スタンドアロンタスクを実行するクラスターを選択します。

   サービスを起動するリソースを決定します。

サービスの起動元	Steps
クラスター	a. [Clusters] ページで、サービスを作成するクラスターを選択します。 b. Tasksタブで、Run new taskを選択します。
起動タイプ	a. [Task] (タスク) ページで、タスク定義を選択します。 b. リビジョンが複数ある場合は、リビジョンを選択します。 c. [Create] (作成)、[Run task] (タスクの実行) の順に選択します。

3. (オプション) スケジュールされたタスクをクラスターのインフラストラクチャ全体に分散する方法を選択します。[Compute configuration] (コンピュート設定) を展開し、以下の操作を実行します。

ディストリビューションの方法	Steps	
起動タイプ	a. [Compute options] (コンピューティングオプション) セクションで、[Launch type] (起動タイプ) を選択します。 b. [起動タイプ] で、[EC2] を選択します。	

4. Application type(アプリケーションの種類)で、Task(タスク)を選択します。
5. [タスク定義] で、先に作成した `efs-tutorial` タスク定義を選択します。
6. [必要なタスク] に、「1」と入力します。
7. [作成] を選択します。
8. [クラスター] ページで、[インフラストラクチャ] を選択します。
9. [コンテナインスタンス] で、接続先となるコンテナインスタンスを選択します。
10. [コンテナインスタンス] ページの インスタンスの [ネットワーキング] で、インスタンスの [パブリック IP] を書き留めます。
11. ブラウザを開き、パブリック IP アドレスを入力します。次のメッセージが表示されます。

```
It works!
You are using an Amazon EFS file system for persistent container storage.
```

> **Note**
>
> メッセージが表示されない場合は、コンテナインスタンスのセキュリティグループがポート 80 でインバウンドネットワークトラフィックを許可しており、ファイルシステムのセキュリティグループがコンテナインスタンスからのインバウンドアクセスを許可していることを確認してください。

Amazon ECS での FSx for Windows File Server ボリュームの使用

FSx for Windows File Server は、Windows ファイルシステムによってバックアップされるフルマネージド型 Windows ファイルサーバーを提供します。ECS と共に FSx for Windows File Server を使用する場合、永続的、分散、共有、静的ファイルストレージで Windows タスクをプロビジョニングできます。詳細については、FSx for Windows File Serverとは？を参照してください。

> **Note**
>
> FSx for Windows File Server ECS のタスクボリュームは、Amazon ECS に最適化された Windows Server 2016 Full AMI を使用する EC2 インスタンスではサポートされていません。Fargate 設定上の Windows コンテナで、FSx for Windows File Server ボリュームを使用することはできません。代わりに、起動時にコンテナをマウントするように変更できます。

FSx for Windows File Server を使用すると、共有外部ストレージ、リージョン別の高可用ストレージ、または高スループットストレージへのアクセスを必要とする Windows ワークロードのデプロイが行えます。Amazon ECS Windows インスタンスで実行されている Amazon ECS コンテナには、1 つ以上の FSx for Windows File Server ファイルシステムボリュームをマウントできます。1 つの Amazon ECS タスク内の複数の Amazon ECS コンテナ間で、FSx for Windows File Server ファイルシステムボリュームを共有できます。

ECS で FSx for Windows File Server を使用できるようにするには、次のタスク定義 JSON スニペット例に示すように、タスク定義に FSx for Windows File Server ファイルシステム ID と関連情報を含める必要があります。この例を、次のタスク定義 JSON スニペットで示します。タスク定義を作成して実行する際には、以下を用意しておきます。

- 有効なドメインに参加している ECS Windows EC2 インスタンス。これは AWS Directory Service for Microsoft Active Directory や、オンプレミスまたは Amazon EC2 上でセルフホストされたアクティブディレクトリのいずれかで、ホストされます。
- Active Directory ドメインの接続と FSx for Windows File Server ファイルシステムのアタッチに使用される、認証情報を含む AWS Secrets Manager シークレットまたは Systems Manager パラメータ。認証情報値は、アクティブディレクトリの作成時に入力した名前とパスワードの認証情報です。

関連チュートリアルについては、「Amazon ECS 用の FSx for Windows File Server ファイルシステムを設定する方法について説明します。」を参照してください。

考慮事項

FSx for Windows File Server ボリュームを使用する際は、以下の点を考慮します。

- Windows Amazon EC2 インスタンスでは、FSx for Windows File Server ボリュームが Amazon ECS でネイティブにサポートされるため、Amazon ECS がタスク定義設定を通じてマウントを自動的に管理します。

  Linux Amazon EC2 インスタンスでは、Amazon ECS がタスク定義を通じて FSx for Windows File Server ボリュームを自動的にマウントすることができません。ただし、ホストレベルで Linux EC2 インスタンスに FSx for Windows File Server ファイル共有を手動でマウントし、そのパスを Amazon ECS コンテナにバインドマウントすることは可能です。詳細については、「Linux からの Amazon FSx ファイル共有のマウント」を参照してください。

  > **Important**
  >
  > これは、セルフマネージド型の設定です。Linux での FSx for Windows File Server ファイル共有のマウントとメンテナンスに関するガイダンスについては、FSx for Windows File Server ドキュメントを参照してください。

  > **Important**
  >
  > Linux EC2 インスタンスに手動でマウントされた FSx for Windows File Server 共有を使用する場合は、Amazon ECS と FSx for Windows File Server がそれぞれ単独で動作するため、Amazon ECS は Amazon FSx マウントを監視せず、FSx for Windows File Server は Amazon ECS タスクの配置やライフサイクルイベントを追跡しません。Amazon ECS コンテナインスタンスと Amazon FSx ファイルシステム間におけるネットワーク到達可能性の確保、マウントヘルスチェックの実装、およびフェイルオーバーイベントを許容するための再接続ロジックの処理は、お客様の責任になります。

- Amazon ECS を使用した FSx for Windows File Server は AWS Fargate をサポートしません。
- Amazon ECS において、FSx for Windows File Server は Amazon ECS マネージドインスタンスではサポートされていません。
- `awsvpc` ネットワークモードで Amazon ECS を使用した FSx for Windows File Server はコンテナエージェントのバージョン `1.54.0` 以降が必要です。

- Amazon ECS タスクに使用できるドライブ文字の最大数は 23 です。FSx for Windows File Server ボリュームを持つ各タスクには、ドライブ文字が割り当てられます。
- デフォルトで、タスクのリソースのクリーンアップ時間は、タスク終了後の 3 時間後に設定されます。タスクによって作成されたファイルマッピングは、使用するタスクが存在しなくなっても 3 時間は保持されます。Amazon ECS 環境変数 `ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION` を使用して、デフォルトのクリーンアップ時間を設定できます。詳細については、「Amazon ECS コンテナエージェントの設定」を参照してください。
- 通常、タスクは FSx for Windows File Server ファイルシステムと同じ VPC でのみ実行されます。ただし、Amazon ECS クラスター VPC と FSx for Windows File Server ファイルシステムの間に VPC ピアリングを介してネットワーク接続が確立されている場合は、クロス VPC の実行がサポートされます。
- VPC セキュリティグループを設定することにより、ネットワークレベルで FSx for Windows File Server ファイルシステムへのアクセスを制御します。Active Directory セキュリティグループが正しく設定された Active Directory ドメインに参加している EC2 インスタンスでホストされているタスクのみが、FSx for Windows File Server ファイル共有にアクセスできます。セキュリティグループの設定が正確でない場合、Amazon ECS はタスクの起動に失敗し、次のエラーメッセージを表示します:「`unable to mount file system fs-id`」。
- FSx for Windows File Server は、AWS Identity and Access Management(IAM) を使用して、IAM ユーザーおよびグループが Windows ファイルサーバーリソースに対して特定の FSx で実行可能なアクションを制御します。クライアント認可を使用すると、ユーザーは、特定の FSx for Windows File Server ファイルシステムへのアクセスを許可または拒否する IAM ロールを定義できます。オプションとして、読み取り専用アクセスを必要とし、またクライアントからのファイルシステムへの root アクセスを許可または禁止することができます。詳細については、Amazon FSx Windows ユーザーガイドの「Security」を参照してください。

Amazon ECS での FSx for Windows File Server の使用に関するベストプラクティス

Amazon ECS で FSx for Windows File Server を使用する場合は以下のベストプラクティスに従うことをお勧めします。

FSx for Windows File Server のセキュリティとアクセス制御

FSx for Windows File Server には、FSx for Windows File Server ファイルシステムに保存されるデータのセキュリティを保証し、それを必要とするアプリケーションからのみアクセス可能とするために使用できる以下のアクセス制御機能があります。

FSx for Windows File Server ボリュームのデータ暗号化

FSx for Windows File Server は、2 つのファイルシステム向け暗号化形式をサポートしています。つまり、保管中および転送中のデータの暗号化です。転送中のデータの暗号化は、SMB プロトコル 3.0 以降をサポートするコンテナインスタンスにマップされたファイル共有でサポートされます。保管中のデータの暗号化は、Amazon FSx ファイルシステムの作成時に自動的に有効になります。Amazon FSx は、アプリケーションを変更することなくファイルシステムにアクセスする際に、SMB 暗号化を使用して転送中のデータを自動的に暗号化します。詳細は「Amazon FSx for Windows File Server ユーザーガイド」の「Amazon FSx でのデータ暗号化」を参照してください。

フォルダレベルのアクセス制御に Windows ACL を使用する

Windows Amazon EC2 インスタンスは、Active Directory 認証情報を使用して Amazon FSx ファイル共有にアクセスします。きめ細かいファイルレベルおよびフォルダレベルのアクセス制御には、標準の Windows アクセス制御リスト (ACL) を使用します。複数の認証情報を作成でき、それぞれの認証情報は、特定のタスクに対応する共有内の特定のフォルダ向けとなります。

以下の例では、タスクは Secrets Manager に保存されている認証情報を使用してフォルダ `App01` にアクセスできます。Amazon リソースネーム (ARN) は `1234` です。

```
"rootDirectory": "\\path\\to\\my\\data\App01",
"credentialsParameter": "arn-1234",
"domain": "corp.fullyqualified.com",
```

別の例では、タスクは Secrets Manager に保存されている認証情報を使用してフォルダ `App02` にアクセスできます。その ARN は 6789 です。

```
"rootDirectory": "\\path\\to\\my\\data\App02",
"credentialsParameter": "arn-6789",
"domain": "corp.fullyqualified.com",
```

Amazon ECS タスク定義で FSx for Windows File Server ファイルシステムを指定する

コンテナに FSx for Windows File Server ファイルシステムボリュームを使用するには、タスク定義でボリュームとマウントポイントの設定を指定します。次のスニペットの JSON によるタスク定義では、コンテナの `volumes` と `mountPoints` オブジェクトの構文を示しています。

```
{
    "containerDefinitions": [
        {
            "entryPoint": [
```

```
                "powershell",
                "-Command"
            ],
            "portMappings": [],
            "command": ["New-Item -Path C:\\fsx-windows-dir\\index.html -ItemType file
 -Value '<html> <head> <title>Amazon ECS Sample App</title> <style>body {margin-top:
 40px; background-color: #333;} </style> </head><body> <div style=color:white;text-
align:center> <h1>Amazon ECS Sample App</h1> <h2>It Works!</h2> <p>You are using Amazon
 FSx for Windows File Server file system for persistent container storage.</p>' -
Force"],
            "cpu": 512,
            "memory": 256,
            "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-
ltsc2019",
            "essential": false,
            "name": "container1",
            "mountPoints": [
                {
                    "sourceVolume": "fsx-windows-dir",
                    "containerPath": "C:\\fsx-windows-dir",
                    "readOnly": false
                }
            ]
        },
        {
            "entryPoint": [
                "powershell",
                "-Command"
            ],
            "portMappings": [
                {
                    "hostPort": 443,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ],
            "command": ["Remove-Item -Recurse C:\\inetpub\\wwwroot\\* -Force; Start-
Sleep -Seconds 120; Move-Item -Path C:\\fsx-windows-dir\\index.html -Destination C:\
\inetpub\\wwwroot\\index.html -Force; C:\\ServiceMonitor.exe w3svc"],
            "mountPoints": [
                {
                    "sourceVolume": "fsx-windows-dir",
                    "containerPath": "C:\\fsx-windows-dir",
                    "readOnly": false
```

```
                }
            ],
            "cpu": 512,
            "memory": 256,
            "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-
ltsc2019",
            "essential": true,
            "name": "container2"
        }
    ],
    "family": "fsx-windows",
    "executionRoleArn": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole",
    "volumes": [
        {
            "name": "fsx-windows-dir",
            "fsxWindowsFileServerVolumeConfiguration": {
                "fileSystemId": "fs-0eeb5730b2EXAMPLE",
                "authorizationConfig": {
                    "domain": "example.com",
                    "credentialsParameter": "arn:arn-1234"
                },
                "rootDirectory": "share"
            }
        }
    ]
}
```

`FSxWindowsFileServerVolumeConfiguration`

タイプ: オブジェクト

必須: いいえ

このパラメータは、タスクストレージに FSx for Windows File Server ファイルシステムを使用する場合に指定します。

`fileSystemId`

タイプ: 文字列

必須: はい

使用する FSx for Windows File Server ファイルシステムID。

`rootDirectory`

型: 文字列

必須: はい

ホスト内にルートディレクトリとしてマウントする FSx for Windows File Server ファイルシステム内のディレクトリ。

`authorizationConfig`

`credentialsParameter`

型: 文字列

必須: はい

認可の認証情報オプション。

- Secrets Manager シークレットの Amazon リソースネーム (ARN)。
- 用の Amazon リソースネーム (ARN)Systems Managerパラメータ。

`domain`

型: 文字列

必須: はい

AWS Directory Service for Microsoft Active Directory (AWS Managed Microsoft AD) ディレクトリ、またはセルフホスト型 EC2 Active Directory によってホストされる完全修飾ドメイン名。

FSx for Windows File Server ボリューム認証情報を保存する方法

credentials パラメータで使用するために認証情報を格納するには、2 つの異なる方法があります。

- AWS Secrets Manager シークレット

  この認証情報は、[その他のタイプのシークレット] カテゴリを使用して AWS Secrets Manager コンソールで作成できます。キー/値のペア、ユーザー名/管理者、パスワード/#####ごとに行を追加します。
- Systems Manager パラメータ

この認証情報は、次のコードスニペット例でのフォームにテキストを入力することで、Systems Manager パラメータコンソールで作成できます。

```
{
  "username": "admin",
  "password": "password"
}
```

タスク定義の FSxWindowsFileServerVolumeConfiguration パラメータにある credentialsParameter では、シークレット ARN または Systems Manager パラメータ ARN が保持されます 。詳細については、Secrets Manager ユーザーガイド の「AWSSecrets Manager とは？」およびSecrets Manager ユーザーガイド の「Systems Manager パラメータの保存」を参照してください。

Amazon ECS 用の FSx for Windows File Server ファイルシステムを設定する方法について説明します。

FSx for Windows File Server ファイルシステムと、ファイルシステムにアクセスできるコンテナをホストする Amazon ECS に最適化された Windows インスタンスを起動する方法について説明します。これを行うには、最初に Directory Service AWS Managed Microsoft Active Directory を作成します。次に、Amazon EC2 インスタンスとタスク定義を使用して、FSx for Windows File Server ファイルシステムとクラスターを作成します。FSx for Windows File Server ファイルシステムを使用するために、コンテナのタスク定義を設定します。最後に、ファイルシステムをテストします。

アクティブディレクトリ または FSx for Windows File Server ファイルシステムのいずれかを起動または削除するのに、毎回 20～45 分かかります。90 分以上予約して、チュートリアルを完了するか、いくつかのセッションでチュートリアルを完了する準備をしてください。

チュートリアルの前提条件

- 管理ユーザー。「Amazon ECS を使用するようにセットアップする」を参照してください。
- (オプション) RDP アクセスを介して EC2 Windows インスタンスに接続するための PEM キーペア。キーペアの作成方法については、「Amazon EC2 ユーザーガイド」の「Amazon EC2 のキーペアと Amazon EC2 インスタンス」を参照してください。
- 少なくとも 1 つのパブリックサブネットと 1 つのプライベートサブネット、および 1 つのセキュリティグループを持つ VPC。デフォルトの VPC を使用できます。NAT ゲートウェイやデバイスは必要ありません。Directory Service は、Active Directory でのネットワークアドレス変換 (NAT)

をサポートしていません。これを機能させるには、アクティブディレクトリ、FSx for Windows File Server ファイルシステム、ECS クラスター、および EC2 インスタンスが VPC 内に配置されている必要があります。VPC と Active Directory の詳細については、「VPC を作成する」および「AWS Managed Microsoft AD を作成するための前提条件」を参照してください。

- IAM ecsInstanceRole および ecsTaskExecutionRole アクセス許可は、お客様のアカウントに関連付けられています。このようなサービスリンクロールを使用すると、サービスが API 呼び出しを行い、ユーザーに代わってコンテナ、シークレット、ディレクトリ、ファイルサーバーにアクセスできます。

ステップ 1: IAM アクセスロールを作成する

AWS マネジメントコンソール を使用してクラスターを作成します。

1. Amazon ECS コンテナインスタンスの IAM ロールを参照して ecsInstanceRole があるかどうかを確認し、持っていない場合はどのように作成できるかを確認してください。
2. 実際の本番環境では、ロールポリシーを最小限のアクセス許可に合わせてカスタマイズすることをお勧めします。このチュートリアルを通じて取り組むために、次の AWS マネージド ポリシーが ecsInstanceRole に添付されていることを確認します。ポリシーがまだアタッチされていない場合は、ポリシーをアタッチします。

   - AmazonEC2ContainerServiceforEC2Role
   - AmazonSSMManagedInstanceCore
   - AmazonSSMDirectoryServiceAccess

   AWS マネージド ポリシーをアタッチするには

   a. IAM コンソールを開きます。
   b. ナビゲーションペインで [Roles (ロール)] を選択します。
   c. [AWS 管理ロール] を選択します。
   d. [アクセス許可]、[ポリシーのアタッチ] の順に選択します。
   e. アタッチする利用可能なポリシーを絞り込むには、[Filter] を使用します。
   f. 適切なポリシーを選択し、[Attach policy] を選択します。
3. Amazon ECS タスク実行IAM ロール を参照して ecsTaskExecutionRole があるかどうかを確認し、持っていない場合はどのように作成できるかを確認してください。

実際の本番環境では、ロールポリシーを最小限のアクセス許可に合わせてカスタマイズすることをお勧めします。このチュートリアルを通じて取り組むために、次の AWSマネージド ポリシーが ecsTaskExecutionRole に添付されていることを確認します。ポリシーがまだアタッチされていない場合は、ポリシーをアタッチします。AWSマネージド ポリシーをアタッチするには、前のセクションで説明した手順を使用します。

- SecretsManagerReadWrite
- AmazonFSxReadOnlyAccess
- AmazonSSMReadOnlyAccess
- AmazonECSTaskExecutionRolePolicy

手順 2: Windows Active Directory (AD) を作成する

1. 「AWS Directory Service 管理ガイド」の「AWS マネージド AD の作成]で説明されている手順に従います。このチュートリアル用に指定した VPC を使用します。「AWS マネージド AD の作成」のステップ 3 で、次のステップで使用するためにユーザー名と管理者パスワードを保存します。また、今後の手順の完全修飾ディレクトリ DNS 名をメモしておきます。Active Directory の作成中に、次の手順を実行できます。
2. 次のステップで使用する AWS Secrets Manager のシークレットを作成します。詳細については、「AWS Secrets Manager ユーザーガイド」の「Secrets Manager の使用を開始する」を参照してください。
   a. Secrets Managerコンソール を開きます。
   b. [新しいシークレットを保存する] を選択します。
   c. [その他のシークレット] を選択します。
   d. [シークレットキー/値] では、最初の行に値 **admin** を含むキー **username** を作成します。[+ 行を追加] をクリックします。
   e. 新しい行で、キー **password** を作成します。値には、AWSマネージド AD ディレクトリ の作成 の手順 3 で入力したパスワードを入力します。
   f. [Next] ボタンをクリックします。
   g. シークレットの名前と説明を入力します。[次へ] をクリックします。
   h. [次へ] をクリックします。[保存] をクリックします。
   i. [シークレット] ページのリストから、作成したシークレットをクリックします。
   j. 次の手順で使用するために、新しいシークレットの ARN を保存します。

k. Active Directory の作成中に、次の手順に進むことができます。

ステップ 3: セキュリティグループを確認および更新する

このステップでは、使用しているセキュリティグループのルールを確認および更新します。このためには、VPC 用に作成されたデフォルトのセキュリティグループを使用できます。

セキュリティグループを確認および更新します。

ポートとの間でデータを送信するには、セキュリティグループを作成または編集する必要があります。これについては、[FSx for Windows File Server ユーザーガイド]の[Amazon VPC セキュリティグループ]で説明されています。これを行うには、インバウンドルールの次の表の最初の行に示されているセキュリティグループインバウンドルールを作成します。このルールでは、同じセキュリティグループに割り当てられているネットワークインターフェイス (および関連付けられているインスタンス) からのインバウンドトラフィックを使用できます。作成したクラウドリソースはすべて同じ VPC 内にあり、同じセキュリティグループにアタッチされます。したがって、このルールでは、必要に応じて、FSx for Windows File Server ファイルシステム、アクティブディレクトリ、ECSインスタンスとの間でトラフィックを送信することができます。その他のインバウンドルールでは、ウェブサイトでトラフィックを利用し、ECS インスタンスに接続するための RDP アクセスが行えます。

次の表に、このチュートリアルに必要なセキュリティグループのインバウンドルールを示します。

タイプ	プロトコル	ポート範囲	ソース
すべてのトラフィック	すべて	すべて	*sg-securitygroup*
HTTPS	TCP	443	0.0.0.0/0
RDP	TCP	3389	ラップトップの IP アドレス

次の表に、このチュートリアルに必要なセキュリティグループのアウトバウンドルールを示します。

タイプ	プロトコル	ポート範囲	宛先

タイプ	プロトコル	ポート範囲	宛先
すべてのトラフィック	すべて	すべて	0.0.0.0/0

1. EC2 コンソールを開き、左側のメニューから [セキュリティグループ] を選択します。
2. 表示されるセキュリティグループのリストから、このチュートリアルで使用しているセキュリティグループの左側にあるチェックボックスをオンにします。

   セキュリティグループの詳細が表示されます。
3. [インバウンドルール] または [アウトバウンドルール] タブを選択して、[インバウンドルールの編集] または [アウトバウンドルールの編集] ボタンを選択して、インバウンドルールおよびアウトバウンドルールを編集します。前の表に表示されたルールと一致するようにルールを編集します。このチュートリアルの後半で EC2 インスタンスを作成した後、「Amazon EC2 ユーザーガイド」の「RDP を使用して Windows インスタンスに接続する」に記載の手順に従って、EC2 インスタンスのパブリック IP アドレスを使用してインバウンドルールの RDP ソースを編集します。

ステップ 4: FSx for Windows File Server ファイルシステムを作成する

セキュリティグループが検証および更新され、アクティブディレクトリ が作成され、アクティブステータスになったら、Active Directory と同じ VPC に FSx for Windows File Server ファイルシステムを作成します。次のステップを使用して、Windows タスク用の FSx for Windows File Server ファイルシステムを作成します。

最初のファイルシステムを作成します。

1. [Amazon FSx コンソール] を開きます。
2. ダッシュボードで [Create file system] (ファイルシステムの作成) を選択して、ファイルシステム作成ウィザードをスタートします。
3. [Select file system type] (ファイルシステムのタイプを選択) のページで、[FSx for Windows File Server] (FSx for Windows ファイルサーバー) を選択し、[Next] (次へ) を選択します。[Create file system] (ファイルシステムを作成) ページが表示されます。
4. [File system details] (ファイルシステム詳細) セクションで、ファイルシステムの名前を入力します。ファイルシステムに名前を付けると、ファイルの検索と管理が容易になります。最大 256 文字の Unicode 文字を使用できます。使用できる文字は、文字、数字、スペース、および特殊

文字のプラス記号 (+)、マイナス記号 (-)、等号 (=)、ピリオド (.)、アンダースコア (_)、コロン (:)、スラッシュ (/) です。

5. [Deployment type] で [Single-AZ] を選択して、1 つのアベイラビリティーゾーンにデプロイされたファイルシステムをデプロイします。[Single-AZ 2] (シングル AZ 2) は、最新世代の単一アベイラビリティーゾーンファイルシステムで、SSD および HDD ストレージをサポートします。
6. [Storage type] で、 [HDD] を選択します。
7. [Storage capacity] に、ストレージの最小容量を入力します。
8. [Throughput capacity] はデフォルト設定のままにします。
9. [ネットワーク & セキュリティ] セクションで、Directory Service ディレクトリに対して選択したものと同じ Amazon VPC を選択します。
10. [VPC Security Groups] で、ステップ 3: セキュリティグループを確認および更新するで検証したセキュリティグループを選択します。
11. [Windows 認証] では、[AWSマネージド Microsoft アクティブディレクトリ] を選択し、リストからお使いの Directory Service ディレクトリを選択します。
12. [Encryption] (暗号化) では、[aws / fsx (default)] (aws / fsx (デフォルト) の [Encryption key] (暗号化キー) 設定をデフォルトのままにします。
13. [Maintenance preferences] ではデフォルト設定のままにします。
14. [Next] ボタンをクリックします。
15. [Create file system] (ファイルシステムの作成) ページで表示されるファイルシステム設定を確認します。参照のために、ファイルシステム作成後に変更できるファイルシステム設定を書き留めます。[Create file system] (ファイルシステムの作成) を選択します。
16. ファイルシステム ID をメモします。ID は後の手順で使用する必要があります。

    FSx for Windows File Server ファイルシステムの作成中に、クラスターと EC2 インスタンスを作成するには、次のステップに進みます。

ステップ 5: Amazon ECS クラスターを作成する

Amazon ECS コンソールを使用してクラスターを作成する

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションバーから、使用するリージョンを選択します。
3. ナビゲーションペインで [Clusters] (クラスター) を選択します。

4. [Clusters] (クラスター) ページで、[Create Cluster] (クラスターの作成) を選択します。
5. [クラスター設定] の [クラスター名] に「windows-fsx-cluster」と入力します。
6. [インフラストラクチャ] を展開し、[AWS Fargate (サーバーレス)] をクリアして、[Amazon EC2 インスタンス] を選択します。

   - Auto Scaling グループを作成するには、Auto Scaling group(ASG) (Auto Scaling グループ) から、[Create new group] (新しいグループの作成) を選択し、グループに関する以下の詳細情報を入力します。

     - [オペレーティングシステム/アーキテクチャ] で [Windows Server 2019 Core] を選択します。
     - [EC2 インスタンスタイプ] で [t2.medium] または [t2.micro] を選択します。
7. [作成] を選択します。

ステップ 6: Amazon ECS に最適化された Amazon EC2 インスタンスを作成する

Amazon ECS Windows コンテナインスタンスを作成します。

Amazon ECS インスタンスを作成するには

1. `aws ssm get-parameters` コマンドを使用して、VPC をホストするリージョン用の AMI 名を取得します。詳細については、「Amazon ECS に最適化された AMI メタデータを取得する」を参照してください。
2. Amazon EC2 コンソールを使用して、インスタンスを起動します。

   a. Amazon EC2 コンソールの https://console.aws.amazon.com/ec2/ を開いてください。
   b. ナビゲーションバーから、使用するリージョンを選択します。
   c. EC2 ダッシュボードから、[Launch Instance] を選択します。
   d. [Name (名前)] に一意の名前を入力します。
   e. [アプリケーションと OS イメージ (Amazon マシンイメージ)] の [検索] フィールドに、取得した AMI 名を入力します。
   f. [インスタンスタイプ] で [t2.medium] または [t2.micro] を選択します。
   g. [Key pair (login)] (キーペア (ログイン)) には、キーペアを選択します。キーペアを指定しない場合、
   h. [ネットワーク設定] の [VPC] と [サブネット]で VPC とパブリックサブネットを選択します。

i. [Network settings] (ネットワーク設定) にある [Security group] (セキュリティグループ) には、既存のセキュリティグループを選択することも、新しいセキュリティグループを作成することもできます。選択したセキュリティグループに チュートリアルの前提条件 で定義されたインバウンドルールとアウトバウンドルールがあることを確認してください。

j. [Network settings] (ネットワーク設定) の [Auto-assign Public IP] (パブリック IP の自動割り当て) で、[Enable] (有効にする) を選択します。

k. [高度な詳細] を展開し、作成した Active Directory の ID を [ドメイン結合ディレクトリ] で選択します。このオプションドメインは、EC2 インスタンスの起動時に AD に参加します。

l. [Advanced details] (高度な詳細) で、[IAM instance profile] (IAM インスタンスプロファイル) として [ecsInstanceRole] を選択します。

m. 次のユーザーデータを使用して、Amazon ECS コンテナインスタンスを設定します。[Advanced Details] (詳細情報) にある [User data] (ユーザーデータ) フィールドに以下のスクリプトを貼り付け、*cluster_name* をクラスターの名前に置き換えます。

```
<powershell>
Initialize-ECSAgent -Cluster windows-fsx-cluster -EnableTaskIAMRole
</powershell>
```

n. 準備ができたら、確認フィールドを選択してから、[Launch Instances] を選択します。

o. 確認ページは、インスタンスが起動中であることを通知します。[View Instances] (インスタンスを表示) を選択して確認ページを閉じ、コンソールに戻ります。

3. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。

4. ナビゲーションペインで、[クラスター] を選択し、[windows-fsx-cluster] を選択します。

5. [インフラストラクチャ] タブを選択し、インスタンスが windows-fsx-cluster クラスターに登録されていることを確認します。

ステップ 7: Windows タスク定義を登録する

Amazon ECS クラスターで Windows コンテナを実行する前に、タスク定義を登録する必要があります。次のタスク定義の例では、シンプルなウェブページを表示します。このタスクは、FSx ファイルシステムにアクセスできる 2 つのコンテナを起動します。最初のコンテナは HTML ファイルをファイルシステムに書き込みます。2 番目のコンテナは、ファイルシステムから HTML ファイルをダウンロードし、ウェブページを提供します。

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。

2. ナビゲーションペインで、タスク定義 を選択します。
3. [Create new task definition] (新しいタスク定義の作成)、[Create new task definition with JSON] (JSON で新しいタスク定義を作成) の順に選択します。
4. JSON エディタボックスで、タスク実行ロールの値と FSx ファイルシステムの詳細を置き換えてから [保存] を選択します。

```
{
    "containerDefinitions": [
        {
            "entryPoint": [
                "powershell",
                "-Command"
            ],
            "portMappings": [],
            "command": ["New-Item -Path C:\\fsx-windows-dir\\index.html -ItemType
 file -Value '<html> <head> <title>Amazon ECS Sample App</title> <style>body
 {margin-top: 40px; background-color: #333;} </style> </head><body> <div
 style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1> <h2>It
 Works!</h2> <p>You are using Amazon FSx for Windows File Server file system for
 persistent container storage.</p>' -Force"],
            "cpu": 512,
            "memory": 256,
            "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-
ltsc2019",
            "essential": false,
            "name": "container1",
            "mountPoints": [
                {
                    "sourceVolume": "fsx-windows-dir",
                    "containerPath": "C:\\fsx-windows-dir",
                    "readOnly": false
                }
            ]
        },
        {
            "entryPoint": [
                "powershell",
                "-Command"
            ],
            "portMappings": [
                {
                    "hostPort": 443,
```

```
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ],
            "command": ["Remove-Item -Recurse C:\\inetpub\\wwwroot\\* -Force;
 Start-Sleep -Seconds 120; Move-Item -Path C:\\fsx-windows-dir\\index.html -
Destination C:\\inetpub\\wwwroot\\index.html -Force; C:\\ServiceMonitor.exe
 w3svc"],
            "mountPoints": [
                {
                    "sourceVolume": "fsx-windows-dir",
                    "containerPath": "C:\\fsx-windows-dir",
                    "readOnly": false
                }
            ],
            "cpu": 512,
            "memory": 256,
            "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-
ltsc2019",
            "essential": true,
            "name": "container2"
        }
    ],
    "family": "fsx-windows",
    "executionRoleArn": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole",
    "volumes": [
        {
            "name": "fsx-windows-dir",
            "fsxWindowsFileServerVolumeConfiguration": {
                "fileSystemId": "fs-0eeb5730b2EXAMPLE",
                "authorizationConfig": {
                    "domain": "example.com",
                    "credentialsParameter": "arn:arn-1234"
                },
                "rootDirectory": "share"
            }
        }
    ]
}
```

ステップ 8: タスクを実行して結果を表示する

タスクを実行する前に、FSx for Windows File Server ファイルシステムのステータスが [Available] であることを確認します。利用可能になったら、作成したタスク定義を使用してタスクを実行できます。タスクはコンテナを作成することから始まります。コンテナはファイルシステムを使用してコンテナ間で HTML ファイルをシャッフルします。シャッフル後、ウェブサーバーは単純な HTML ページを提供します。

> **Note**
>
> VPN 内からそのウェブサイトに接続できない場合があります。

Amazon ECS コンソールを使用して、タスクを実行して結果を表示します。

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
2. ナビゲーションペインで、[クラスター] を選択し、[windows-fsx-cluster] を選択します。
3. [タスク] タブを選択し、[新しいタスクを実行] を選択します。
4. [起動タイプ] で、[EC2] を選択します。
5. デプロイ設定の [タスク定義] で [fsx-windows] を選択し、[作成] を選択します。
6. タスクのステータスが [実行中] の場合は、タスク ID を選択します。
7. [コンテナ] で container1 のステータスが [停止] の場合、 container2 を選択してコンテナの詳細を表示します。
8. [container2 のコンテナの詳細] で [ネットワークバインディング] を選択し、コンテナに関連付けられている外部 IP アドレスをクリックします。ブラウザが開き、次のメッセージが表示されます。

```
Amazon ECS Sample App
It Works!
You are using Amazon FSx for Windows File Server file system for persistent
 container storage.
```

> **Note**
>
> メッセージが表示されるまでに数分かかることがあります。このメッセージが数分経っても表示されない場合は、VPN で実行していないことを確認し、コンテナインスタンス

のセキュリティグループがポート 443 でのインバウンドネットワーク HTTP トラフィックを許可していることを確認します。

ステップ 9: クリーンアップする。

> **Note**
>
> FSx for Windows File Server ファイルシステムまたは AD の削除には、20～45分かかります。AD の削除操作を開始する前に、FSx for Windows File Server ファイルシステムの削除操作が完了するまで待たなければなりません。

FSx for Windows File Server ファイルシステムを削除します。

1. [Amazon FSxコンソール] を開きます。
2. 作成した FSx for Windows File Server ファイルシステムの左側にあるラジオボタンを選択します。
3. [アクション] を選択します。
4. [Delete file system] をクリックします。

AD を削除します。

1. Directory Service コンソールを開きます。
2. 作成した AD の左側にあるラジオボタンを選択します。
3. [アクション] を選択します。
4. [Delete directory] を選択します。

クラスターを削除します。

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
2. ナビゲーションペインで、[クラスター] を選択し、[windows-fsx-cluster] を選択します。
3. [クラスターの削除] を選択します。
4. フレーズを入力し、[削除] を選択します。

EC2 インスタンスを終了します。

1. Amazon EC2 コンソールを開きます。
2. 左側のメニューから、[Instances] を選択します。
3. 作成した EC2 インスタンスの左側にあるボックスをチェックします。
4. [インスタンス状態]、[インスタンスを終了] の順にクリックします。

シークレットを削除します。

1. Secrets Managerコンソール を開きます。
2. このチュートリアル用に作成したシークレットを選択します。
3. [Actions] をクリックします。
4. [Delete secret] を選択します。

Amazon ECS の S3 Files の設定

S3 Files は、任意の AWS コンピューティングリソースを Amazon S3 のデータに直接接続する共有ファイルシステムです。これにより、データが S3 を離れることなく、完全なファイルシステムのセマンティクスと低レイテンシーのパフォーマンスを備えたファイルとして、すべての S3 データにすばやく直接アクセスできます。ファイルおよびディレクトリオペレーションを使用して、データの読み取り、書き込み、整理を行うことができ、S3 Files はファイルシステムと S3 バケットを自動的に同期させたままにします。Amazon ECS を使用すると、S3 ファイルシステムをタスク定義のボリュームとして定義し、S3 バケットに保存されているデータへの直接ファイルシステムアクセスをコンテナに提供できます。Amazon S3 Files とその機能の詳細については、「Amazon S3 ユーザーガイド」を参照してください。

可用性

Amazon ECS での S3 Files のサポートは、一般提供で次の起動タイプで利用できます。

- Fargate — 完全にサポートされています。
- Amazon ECS マネージドインスタンス — 完全にサポートされています。

> **Important**
>
> S3 Files は、現時点では Amazon EC2 起動タイプではサポートされていません。タスク定義で S3 ファイルシステムを設定し、Amazon EC2 起動タイプで実行しようとすると、起動時にタスクは失敗します。Amazon EC2 起動タイプのサポートは、今後のリリースが予定されています。

考慮事項

- S3 ファイルシステムは、タスク定義で専用 `s3filesVolumeConfiguration` パラメータを使用します。
- S3 ファイルシステムでは、ファイルシステムを識別するために完全な Amazon リソースネーム (ARN) が必要です。ARN 形式は以下です:

```
arn:{partition}:s3files:{region}:{account-id}:file-system/fs-xxxxx
```

- 転送暗号化が S3 ファイルシステムボリュームでは必須であり、自動的に適用されます。それを無効にするオプションはありません。
- タスク IAM ロールが S3 ファイルシステムボリュームでは必須であり、自動的に適用されます。それを無効にするオプションはありません。

前提条件

Amazon ECS タスク定義で S3 ファイルシステムボリュームを設定する前に、次の前提条件を満たしていることを確認してください:

- S3 ファイルシステムとマウントターゲット — S3 ファイルシステムが作成され、S3 バケットに関連付けられている必要があります。S3 ファイルシステムを作成する手順については、「Amazon S3 Files ユーザーガイド」を参照してください。
- タスク IAM ロール — タスク定義には、次のアクセス許可を使用するタスク IAM ロールを含める必要があります:
  - アプリケーションコードから S3 ファイルシステムに接続して操作するアクセス許可 (コンテナで実行)。
  - アプリケーションコードから S3 オブジェクトを読み取るアクセス許可 (コンテナで実行)。
- VPC とセキュリティグループの設定 — S3 ファイルシステムは、Amazon ECS タスクが実行される VPC とサブネットからアクセス可能である必要があります。

- (オプション) S3 Files アクセスポイント — アプリケーション固有のアクセスコントロールを適用する場合は、S3 Files アクセスポイントを作成し、タスク定義で ARN を指定します。

詳細については、「S3 Files の前提条件」を参照してください。

Amazon ECS タスク定義内で Amazon S3 Files ボリュームを指定する

Amazon ECS コンソール、AWS CLI、または AWS API を使用して、Amazon ECS タスク定義内で S3 Files ボリュームを設定できます。

Amazon ECS コンソールの使用

1. Amazon ECS コンソール (https://console.aws.amazon.com/ecs/) を開きます。
2. ナビゲーションペインで、タスク定義 を選択します。
3. [新しいタスク定義の作成] を選択するか、既存のタスク定義を選択して新しいリビジョンを作成します。
4. [インフラストラクチャ] セクションで、必要なアクセス許可を使用して設定されたタスク IAM ロールがあることを確認します。
5. [ストレージ] セクションで、[ボリュームを追加] を選択します。
6. [ボリュームタイプ] では、[S3 Files] を選択します。
7. [ファイルシステム ARN] では、S3 ファイルシステムの完全な ARN を入力します。ARN 形式は以下です:

```
arn:{partition}:s3files:{region}:{account-id}:file-system/fs-xxxxx
```

8. (オプション) [ルートディレクトリ]では、ルートとしてマウントするファイルシステム内のパスを入力します。指定しない場合、ファイルシステムのルート (/) が使用されます。
9. (オプション)[転送中の暗号化ポート] では、Amazon ECS ホストと S3 ファイルシステムの間で暗号化されたデータを送信するためのポート番号を入力します。転送中の暗号化ポートを指定しないと、Amazon EFS マウントヘルパーが使用するポート選択方式が使用されます。
10. (オプション) [アクセスポイント ARN] では、ドロップダウンリストから使用する S3 Files アクセスポイントを選択します。
11. [コンテナマウントポイント] セクションで、コンテナを選択し、コンテナ内のボリュームをマウントするコンテナ内のローカルマウントパスを指定します。
12. [作成] を選択してタスク定義を作成します。

の使用AWS CLI

AWS CLI を使用してタスク定義で S3 Files ボリュームを指定するには、ボリューム定義の `s3filesVolumeConfiguration`パラメータで `register-task-definition` コマンドを使用します。

以下は、S3 Files ボリュームを定義してコンテナにそれをマウントするタスク定義 JSON スニペットの例です。

```
{
  "family": "s3files-task-example",
  "taskRoleArn": "arn:aws:iam::123456789012:role/ecsTaskRole",
  "containerDefinitions": [
    {
      "name": "my-container",
      "image": "my-image:latest",
      "essential": true,
      "mountPoints": [
        {
          "containerPath": "/mnt/s3data",
          "sourceVolume": "my-s3files-volume"
        }
      ]
    }
  ],
  "volumes": [
    {
      "name": "my-s3files-volume",
      "s3filesVolumeConfiguration": {
        "fileSystemArn": "arn:aws:s3files:us-east-1:123456789012:file-
system/fs-0123456789abcdef0",
        "rootDirectory": "/",
        "transitEncryptionPort": 2999
      }
    }
  ]
}
```

タスク定義を登録します。

```
aws ecs register-task-definition --cli-input-json file://s3files-task-def.json
```

アクセスポイントを使用するには、accessPointArn パラメータを含めます。

```
{
  "name": "my-s3files-volume",
  "s3filesVolumeConfiguration": {
    "fileSystemArn": "arn:aws:s3files:us-east-1:123456789012:file-
system/fs-0123456789abcdef0",
    "rootDirectory": "/",
    "transitEncryptionPort": 2999,
    "accessPointArn": "arn:aws:s3files:us-east-1:123456789012:file-
system/fs-0123456789abcdef0/access-point/fsap-0123456789abcdef0"
  }
}
```

S3 Files ボリューム設定パラメータ

s3filesVolumeConfiguration オブジェクトで利用可能なパラメータは、次の表のとおりです。

fileSystemArn

タイプ: 文字列

必須: はい

マウントする S3 ファイルシステムの完全な ARN。形式: arn:*{partition}*:s3files:*{region}*:*{account-id}*:file-system/*fs-xxxxx*

rootDirectory

タイプ: 文字列

必須: いいえ

ボリュームのルートとしてマウントする S3 ファイルシステム内のディレクトリ。指定されない場合のデフォルト値は / です。

transitEncryptionPort

タイプ: 整数

必須: いいえ

Amazon ECS ホストと S3 ファイルシステムとの間で、暗号化されたデータを送信するために使用するポート。トランジット暗号化自体は、常に有効になっており、無効にすることはできません。

`accessPointArn`

タイプ: 文字列

必須: いいえ

使用する S3 Files アクセスポイントの完全な ARN。アクセスポイントは、適用されたユーザー ID とルートディレクトリ設定を使用して、ファイルシステムへのアプリケーション固有のエントリポイントを提供します。

Amazon ECS での Docker ボリュームの使用

Docker ボリュームを使用している場合は、組み込みの `local` ドライバーまたはサードパーティーのボリュームドライバーを使用できます。Docker ボリュームは Docker で管理され、ディレクトリはボリュームデータを含むコンテナインスタンスの `/var/lib/docker/volumes` に作成されます。

Docker ボリュームを使用するには、タスク定義で `dockerVolumeConfiguration` を指定します。詳細については、Docker ドキュメントの「Volumes」を参照してください。

Docker ボリュームの一般的ユースケースは以下のとおりです。

- コンテナで使用する永続データボリュームを提供する
- 定義したデータボリュームを同じコンテナインスタンス上の異なるコンテナにある別々の場所で共有する
- 空の非永続データボリュームを定義し、同じタスク内の複数のコンテナにマウントする
- サードパーティー製ドライバーによって管理されるタスクに、データボリュームを提供する

Docker ボリュームの使用に関する考慮事項

Docker ボリュームを使用する際は、以下の点を考慮します。

- Docker ボリュームは、EC2 起動タイプ、または外部インスタンスを使用する場合にのみサポートされます。
- Windows コンテナでは、`local` ドライバーの使用のみサポートされます。
- サードパーティー製ドライバーを使用する場合は、コンテナエージェントを起動する前に、必ずドライバーをコンテナインスタンスにインストールしアクティブ化しておきます。エージェントを開始する前にサードパーティー製ドライバーがアクティブ化されていない場合は、以下のいずれかのコマンドを使用してコンテナエージェントを再起動することが可能です。

- Amazon ECS に最適化された Amazon Linux 2 AMI の場合:

```
sudo systemctl restart ecs
```

- Amazon ECS に最適化された Amazon Linux AMI の場合:

```
sudo stop ecs && sudo start ecs
```

タスク定義で Docker ボリュームを指定する方法については、「Amazon ECS タスク定義で Docker ボリュームを指定する」を参照してください。

Amazon ECS タスク定義で Docker ボリュームを指定する

コンテナでデータボリュームを使用するには、ボリュームを指定し、タスク定義でポイント設定をマウントする必要があります。このセクションでは、コンテナのボリューム設定について説明します。Docker ボリュームを使用するdockerVolumeConfiguration を指定します。バインドマウントのホストボリュームを使用するタスクで、host とオプションの sourcePath を指定します。

次のタスク定義の JSON スニペットに、コンテナの volumes と mountPoints オブジェクト用の構文を示します。

```
{
    "containerDefinitions": [
        {
            "mountPoints": [
                {
                    "sourceVolume": "string",
                    "containerPath": "/path/to/mount_volume",
                    "readOnly": boolean
                }
            ]
        }
    ],
    "volumes": [
        {
            "name": "string",
            "dockerVolumeConfiguration": {
                "scope": "string",
                "autoprovision": boolean,
                "driver": "string",
                "driverOpts": {
```

```
                    "key": "value"
                },
                "labels": {
                    "key": "value"
                }
            }
        }
    ]
}
```

name

タイプ: 文字列

必須: いいえ

ボリュームの名前。最大 255 文字の英字 (大文字と小文字の区別あり)、数字、ハイフン (-)、アンダースコア (_) を使用できます。この名前は、コンテナ定義 mountPoints オブジェクトの sourceVolume パラメータで参照されます。

dockerVolumeConfiguration

タイプ: DockerVolumeConfiguration オブジェクト

必須: いいえ

このパラメータは、Docker ボリュームを使用する場合に指定します。Docker ボリュームは、EC2 インスタンスでタスクを実行する場合にのみサポートされます。Windows コンテナでは、local ドライバーの使用のみがサポートされます。バインドマウントを使用するには、代わりに host を指定します。

scope

型: 文字列

有効な値: task | shared

必須: いいえ

Docker ボリュームのスコープ。これにより、ボリュームのライフサイクルが決定されます。Docker ボリュームの範囲が task の場合は、タスクが開始すると自動的にプロビジョンされ、タスクが停止すると破棄されます。Docker ボリュームの範囲が shared の場合は、タスクの停止後も保持されます。

`autoprovision`

タイプ: ブール値

デフォルト値: `false`

必須: いいえ

この値が `true` の場合、既に存在していない場合は Docker ボリュームが作成されます。このフィールドは、`scope` が `shared` の場合にのみ使用されます。`scope` が `task` の場合、このパラメータは省略する必要があります。

`driver`

タイプ: 文字列

必須: いいえ

使用する Docker ボリュームドライバー。この名前はタスク配置に使用されるため、ドライバー値は Docker で提供されているドライバー名と一致する必要があります。ドライバーが Docker プラグイン CLI を使用してインストールされた場合は、`docker plugin ls` を使用してコンテナインスタンスからドライバー名を取得します。ドライバーが別の方法でインストール済みである場合は、Docker プラグイン検出を使用してドライバー名を取得します。

`driverOpts`

タイプ: 文字列

必須: いいえ

パススルーする Docker ドライバー固有のオプションのマップ。このパラメータは、Docker の「Create a volume」セクションの `DriverOpts` にマッピングされます。

`labels`

タイプ: 文字列

必須: いいえ

Docker ボリュームに追加するカスタムメタデータ。

`mountPoints`

タイプ: オブジェクト配列

必須: いいえ

コンテナでのデータボリュームのマウントポイント。このパラメータは creat-container Docker API の `Volumes` にマッピングされ、docker run の `--volume` オプションにマッピングされます。

Windows コンテナは `$env:ProgramData` と同じドライブに全部のディレクトリをマウントできます。Windows コンテナは、別のドライブにディレクトリをマウントすることはできません。また、マウントポイントは複数のドライブにまたがることはできません。Amazon EBS ボリュームを Amazon ECS タスクに直接アタッチするには、マウントポイントを指定する必要があります。

`sourceVolume`

タイプ: 文字列

必須: はい (`mountPoints` を使用する場合)

マウントするボリュームの名前。

`containerPath`

型: 文字列

必須: はい (`mountPoints` を使用する場合)

ボリュームをマウントするコンテナ内のパス。

`readOnly`

タイプ: ブール値

必須: いいえ

この値が `true` の場合、コンテナはボリュームへの読み取り専用アクセスを許可されます。この値が `false` の場合、コンテナはボリュームに書き込むことができます。デフォルト値は `false` です。

Windows オペレーティングシステムを実行している EC2 インスタンスで実行されるタスクの場合、 値をデフォルト の `false` のままにします。

Amazon ECS の Docker ボリュームの例

次の例は、コンテナにエフェメラルストレージを提供する方法、複数のコンテナに共有ボリュームを提供する方法、およびコンテナに NFS 永続ストレージを提供する方法を示しています。

Docker ボリュームを使用してコンテナの一時ストレージを提供する

この例では、コンテナは空のデータボリュームを使用しており、これはタスクが完了した後に破棄されます。ユースケースの一例としては、タスクの実行中に一部のスクラッチファイルストレージにアクセスする必要のあるコンテナが考えられます。このタスクは、Docker ボリュームを使用して行うことができます。

1. タスク定義の volumes セクションで、name および DockerVolumeConfiguration 値を使用して、データボリュームを定義します。この例では、タスクが停止した後にボリュームが削除されるように、範囲に task を指定して、組み込みの local ドライバーを使用します。

```
"volumes": [
    {
        "name": "scratch",
        "dockerVolumeConfiguration" : {
            "scope": "task",
            "driver": "local",
            "labels": {
                "scratch": "space"
            }
        }
    }
]
```

2. containerDefinitions セクションで、定義したボリュームの名前を参照する mountPoints 値と、コンテナにボリュームをマウントするための containerPath 値を使用して、コンテナを定義します。

```
"containerDefinitions": [
    {
        "name": "container-1",
        "mountPoints": [
            {
              "sourceVolume": "scratch",
              "containerPath": "/var/scratch"
            }
        ]
    }
]
```

Docker ボリュームを使用して複数のコンテナの永続的ストレージを提供するには

この例では、複数のコンテナで使用する共有ボリュームが必要で、また、使用するタスクの内の 1 つが停止した後も、このボリュームを維持する必要があります。組み込みの `local` ドライバーは使用中です。このためボリュームは、コンテナインスタンスのライフサイクルに紐付けられたままです。

1. タスク定義の `volumes` セクションで、`name` および `DockerVolumeConfiguration` 値を使用して、データボリュームを定義します。この例では、`shared` スコープをボリュームが維持されるように指定し、自動プロビジョニングを `true` に設定します。これは、使用するためのボリュームが作成されるようにするためです。また、組み込みの `local` ドライバーも使用します。

```
"volumes": [
    {
        "name": "database",
        "dockerVolumeConfiguration" : {
            "scope": "shared",
            "autoprovision": true,
            "driver": "local",
            "labels": {
                "database": "database_name"
            }
        }
    }
]
```

2. `containerDefinitions` セクションで、定義したボリュームの名前を参照する `mountPoints` 値と、コンテナにボリュームをマウントするための `containerPath` 値を使用して、コンテナを定義します。

```
"containerDefinitions": [
    {
        "name": "container-1",
        "mountPoints": [
        {
          "sourceVolume": "database",
          "containerPath": "/var/database"
        }
      ]
    },
    {
```

```
      "name": "container-2",
      "mountPoints": [
        {
          "sourceVolume": "database",
          "containerPath": "/var/database"
        }
      ]
    }
  ]
```

Docker ボリュームを使用してコンテナの NFS 永続的ストレージを提供する

この例では、コンテナはタスクの開始時に自動的にマウントされ、タスクの停止時にアンマウントされる NFS データボリュームを使用します。これは Docker ビルトイン `local` ドライバーを使用します。ユースケースの一例としては、ローカル NFS ストレージがあり、ECS Anywhere タスクからアクセスする必要がある場合があります。これは、NFS ドライバーオプションの Docker ボリュームを使用して実現できます。

1. タスク定義の `volumes` セクションで、`name` および `DockerVolumeConfiguration` 値を使用して、データボリュームを定義します。この例では、タスクが停止した後にボリュームがアンマウントされるように `task` のスコープを指定してください。`local` ドライバーを使用し、`type`、`device`、および `o` オプションを適宜使用して `driverOpts` を設定します。NFS_SERVER は NFS サーバーエンドポイントに置き換えます。

```
"volumes": [
       {
           "name": "NFS",
           "dockerVolumeConfiguration" : {
               "scope": "task",
               "driver": "local",
               "driverOpts": {
                   "type": "nfs",
                   "device": "$NFS_SERVER:/mnt/nfs",
                   "o": "addr=$NFS_SERVER"
               }
           }
       }
   ]
```

2. containerDefinitions セクションで、定義したボリュームの名前を参照する mountPoints 値と、コンテナにボリュームをマウントするための containerPath 値を使用して、コンテナを定義します。

```
"containerDefinitions": [
       {
           "name": "container-1",
           "mountPoints": [
               {
                 "sourceVolume": "NFS",
                 "containerPath": "/var/nfsmount"
               }
           ]
       }
   ]
```

Amazon ECS でのバインドマウントの使用

バインドマウントでは、ホスト (Amazon EC2 インスタンスなど) 上のファイルまたはディレクトリがコンテナにマウントされます。バインドマウントは、Fargate インスタンスと Amazon EC2 インスタンスの両方でホストされているタスクでサポートされています。バインドマウントは、それらを使用するコンテナのライフサイクルに紐付けられています。タスクが停止するなど、バインドマウントを使用するすべてのコンテナが停止すると、データが削除されます。Amazon EC2 インスタンスでホストされているタスクの場合、タスク定義で host とオプションの sourcePath 値を指定することにより、ホスト側 Amazon EC2 インスタンスのライフサイクルにデータを紐付けすることができます。詳細については、Docker ドキュメントの「Bind mounts」を参照してください。

バインドマウントの一般的なユースケースは以下のとおりです。

- 1 つ以上のコンテナにマウントするための空のデータボリュームを提供する。
- 1 つ以上のコンテナにホストデータボリュームをマウントする。
- ソースコンテナのデータボリュームを、同じタスク内の他のコンテナと共有する。
- Dockerfile から 1 つ以上のコンテナにパスとその内容を公開する。

バインドマウントを使用するときの考慮事項

バインドマウントを使用する際には、以下の点を考慮してください。

- プラットフォームバージョン 1.4.0 以降 (Linux) または 1.0.0 以降 (Windows) を使用して AWS Fargate でホストされているタスクの場合、デフォルトでは、バインドマウント用に最低 20 GiB の一時ストレージが割り当てられます。タスク定義で ephemeralStorage パラメータを指定することによって、一時ストレージの総量を最大 200 GiB まで増やすことができます。
- タスクの実行時に Dockerfile のファイルをデータボリュームに公開するために、Amazon ECS データプレーンが VOLUME ディレクティブを探します。VOLUME ディレクティブで指定された絶対パスが、タスク定義で指定された containerPath と同じである場合、VOLUME ディレクティブパス内のデータがデータボリュームにコピーされます。次の Dockerfile の例では、/var/log/exported ディレクトリにある examplefile という名前のファイルがホストに書き込まれ、その後でコンテナ内にマウントされます。

```
FROM public.ecr.aws/amazonlinux/amazonlinux:latest
RUN mkdir -p /var/log/exported
RUN touch /var/log/exported/examplefile
VOLUME ["/var/log/exported"]
```

  デフォルトでは、ボリューム許可は 0755 に設定され、所有者は root に設定されます。これらのアクセス許可は Dockerfile でカスタマイズできます。次の例では、ディレクトリの所有者を node として定義しています。

```
FROM public.ecr.aws/amazonlinux/amazonlinux:latest
RUN yum install -y shadow-utils && yum clean all
RUN useradd node
RUN mkdir -p /var/log/exported && chown node:node /var/log/exported
RUN touch /var/log/exported/examplefile
USER node
VOLUME ["/var/log/exported"]
```

- Amazon EC2 インスタンスでホストされているタスクで、host 値と sourcePath 値が指定されていない場合には、Docker デーモンがバインドマウントを自動的に管理します。このバインドマウントを参照するコンテナが存在しない場合、このマウントは、Amazon ECS コンテナエージェントのタスククリーンアップサービスによって最終的に削除されます。デフォルトでは、この処理はコンテナが終了してから 3 時間後に実行されま。また、この期間は ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION エージェント変数により設定することも可能です。詳細については、「Amazon ECS コンテナエージェントの設定」を参照してください。コンテナのライフサイクルを通じてこのデータを永続化する必要がある場合は、バインドマウントの sourcePath 値を指定します。

- Amazon ECS マネージドインスタンスでホストされているタスクの場合、ルートファイルシステムの一部が読み取り専用になっています。読み取り/書き込みバインドマウントは、永続データに /tmp、一時データに /var といった書き込み可能なディレクトリを使用する必要があります。他のディレクトリに読み取り/書き込みバインドマウントを作成しようとすると、以下のような エラーでタスクの起動が失敗します。

```
error creating empty volume: error while creating volume path '/path': mkdir /path:
 read-only file system
```

読み取り専用バインドマウント (mountPoints パラメータの "readOnly": true で設定) は、ホスト上にある任意のアクセス可能なディレクトリをポイントできます。

書き込み可能なパスの完全なリストを表示するには、Amazon ECS マネージドインスタンスでタスクを実行し、それを使用してインスタンスのマウントテーブルを検査できます。ホストファイルシステムにアクセスするには、以下の設定を使用してタスク定義を作成します。

```
{
    "pidMode": "host",
    "containerDefinitions": [{
        "privileged": true,
        ...
    }]
}
```

次に、コンテナ内から以下のコマンドを実行します。

```
# List writable mounts
cat /proc/1/root/proc/1/mounts | awk '$4 ~ /^rw,/ || $4 == "rw" {print $2}' | sort

# List read-only mounts
cat /proc/1/root/proc/1/mounts | awk '$4 ~ /^ro,/ || $4 == "ro" {print $2}' | sort
```

> **Important**
>
> privileged 設定は、ホスト上でルートアクセスに相当する拡張機能をコンテナに付与します。この例では、診断目的でホストのマウントテーブルを検査するために使用されます。詳細については、「コンテナを特権として実行することは避けてください (Amazon EC2)」を参照してください。

コンテナ内でのコマンドのインタラクティブな実行に関する詳細については、「ECS Exec を使用して Amazon ECS コンテナをモニタリングする」を参照してください。

Amazon ECS タスク定義でバインドマウントを指定する

Fargate または Amazon EC2 インスタンスでホストされている Amazon ECS タスクについて、以下のタスク定義 JSON スニペットにタスク定義向け volumes、mountPoints、ephemeralStorage オブジェクトの構文を示します。

```
{
   "family": "",
   ...
   "containerDefinitions" : [
      {
         "mountPoints" : [
            {
               "containerPath" : "/path/to/mount_volume",
               "sourceVolume" : "string"
            }
          ],
          "name" : "string"
       }
    ],
    ...
    "volumes" : [
       {
          "name" : "string"
       }
    ],
    "ephemeralStorage": {
    "sizeInGiB": integer
    }
}
```

Amazon EC2 インスタンスでホストされる Amazon ECS タスクで、タスクボリュームの詳細を指定する場合は、オプションの host パラメータおよび sourcePath を使用できます。これを指定すると、バインドマウントはコンテナではなくタスクのライフサイクルに紐付けされます。

```
"volumes" : [
    {
```

```
        "host" : {
            "sourcePath" : "string"
        },
        "name" : "string"
    }
]
```

以下では、各タスク定義パラメータについて詳しく説明します。

name

型: 文字列

必須: いいえ

ボリュームの名前。最大 255 文字の英字 (大文字と小文字の区別あり)、数字、ハイフン (-)、アンダースコア (_) を使用できます。この名前は、コンテナ定義 mountPoints オブジェクトの sourceVolume パラメータで参照されます。

host

必須: いいえ

host パラメーターは、バインドマウントのライフサイクルを、タスクではなくホスト Amazon EC2 インスタンスと、それが格納されている場所に関連付けるために使用されます。host パラメーターが空の場合、Docker デーモンはデータボリュームのホストパスを割り当てますが、関連付けられたコンテナの実行が停止した後にデータが保持されるとは限りません。

Windows コンテナは $env:ProgramData と同じドライブに全部のディレクトリをマウントできます。

> **Note**
>
> sourcePath パラメータは、Amazon EC2 インスタンスまたは Amazon ECS マネージドインスタンスでホストされているタスクを使用する場合にのみサポートされます。

sourcePath

タイプ: 文字列

必須: いいえ

host パラメータを使用する場合は、sourcePath を指定して、コンテナに表示されるホスト Amazon EC2 インスタンスのパスを宣言します。このパラメータが空の場合は、Docker デーモンによってホストパスが割り当てられます。host パラメータに sourcePath の場所が含まれている場合、データボリュームは手動で削除するまでホスト Amazon EC2 インスタンスの指定された場所に保持されます。sourcePath の値がホスト Amazon EC2 インスタンスに存在しない場合は、Docker デーモンによって作成されます。その場所が存在する場合は、ソースパスフォルダの内容がエクスポートされます。

mountPoints

タイプ: オブジェクト配列

必須: いいえ

コンテナでのデータボリュームのマウントポイント。このパラメータは creat-container Docker API の Volumes にマッピングされ、docker run の --volume オプションにマッピングされます。

Windows コンテナは $env:ProgramData と同じドライブに全部のディレクトリをマウントできます。Windows コンテナは、別のドライブにディレクトリをマウントすることはできません。また、マウントポイントは複数のドライブにまたがることはできません。Amazon EBS ボリュームを Amazon ECS タスクに直接アタッチするには、マウントポイントを指定する必要があります。

sourceVolume

タイプ: 文字列

必須: はい (mountPoints を使用する場合)

マウントするボリュームの名前。

containerPath

型: 文字列

必須: はい (mountPoints を使用する場合)

ボリュームをマウントするコンテナ内のパス。

readOnly

タイプ: ブール値

必須: いいえ

この値が `true` の場合、コンテナはボリュームへの読み取り専用アクセスを許可されます。この値が `false` の場合、コンテナはボリュームに書き込むことができます。デフォルト値は `false` です。

Windows オペレーティングシステムを実行している EC2 インスタンスで実行されるタスクの場合、 値をデフォルト の `false` のままにします。

`ephemeralStorage`

タイプ: オブジェクト

必須: いいえ

タスクに割り当てるエフェメラルストレージの容量(GB)。このパラメータは、AWS Fargate プラットフォームバージョン `1.4.0` 以降 (Linux) または `1.0.0` 以降 (Windows) を使用してホストされているタスクの場合、利用可能なエフェメラルストレージの総量をデフォルト容量を超えて拡張するために使用されます。

コパイロットCLI、CloudFormation、AWSSDK または CLI を使用して、バインドマウントのエフェメラルストレージを指定します。

Amazon ECS のバインドマウントの例

以下の例では、コンテナにバインドマウントを使用する場合の一般的なユースケースについて説明します。

Fargate タスク用の一時ストレージ容量の増加を割り当てるには

プラットフォームバージョン `1.4.0` 以降 (Linux) または `1.0.0` (Windows) を使用して Fargate でホストされる Amazon ECS タスクの場合、使用するタスク内のコンテナに対して、デフォルト容量を超えるエフェメラルストレージを割り当てることができます。この例は、他の例に組み込むことで、Fargate タスクに一時ストレージを割り当てることができます。

- タスク定義で、`ephemeralStorage`オブジェクトを定義します。`sizeInGiB` は `21`および`200` の値の間にある整数である必要があり、GiB に表されます。

```
"ephemeralStorage": {
    "sizeInGiB": integer
}
```

1 つまたは複数のコンテナに空のデータボリュームを提供する

場合によっては、タスク内のコンテナにスクラッチスペースを提供することがあります。例えば、タスクの実行中に同じスクラッチファイルの保存場所にアクセスする必要のある、2 つのデータベースコンテナがあるとします。これは、バインドマウントを使用して実現できます。

1. タスク定義の volumes セクションで、名前を database_scratch としてバインドマウントを定義します。

```
  "volumes": [
    {
      "name": "database_scratch"
    }
  ]
```

2. containerDefinitions セクションで、データベースのコンテナ定義を作成します。これにより、ボリュームがマウントされます。

```
"containerDefinitions": [
    {
      "name": "database1",
      "image": "my-repo/database",
      "cpu": 100,
      "memory": 100,
      "essential": true,
      "mountPoints": [
        {
          "sourceVolume": "database_scratch",
          "containerPath": "/var/scratch"
        }
      ]
    },
    {
      "name": "database2",
      "image": "my-repo/database",
      "cpu": 100,
      "memory": 100,
      "essential": true,
      "mountPoints": [
        {
          "sourceVolume": "database_scratch",
          "containerPath": "/var/scratch"
        }
```

```
      ]
    }
  ]
```

Dockerfile 内のパスとその内容をコンテナに公開する

この例には、コンテナ内にマウントするデータを書き込む Dockerfile があります。この例は、Fargate または Amazon EC2 インスタンスでホストされているタスクで機能します。

1. Dockerfile を作成します。次の例では、パブリックな Amazon Linux 2 コンテナイメージを使用して、コンテナ内にマウントする /var/log/exported ディレクトリに examplefile という名前のファイルを作成します。VOLUME ディレクティブは絶対パスを指定する必要があります。

   ```
   FROM public.ecr.aws/amazonlinux/amazonlinux:latest
   RUN mkdir -p /var/log/exported
   RUN touch /var/log/exported/examplefile
   VOLUME ["/var/log/exported"]
   ```

   デフォルトでは、ボリューム許可は 0755 に設定され、所有者は root に設定されます。これらのアクセス許可は Dockerfile で変更できます。以下の例では、/var/log/exported ディレクトリの所有者が node に設定されています。

   ```
   FROM public.ecr.aws/amazonlinux/amazonlinux:latest
   RUN yum install -y shadow-utils && yum clean all
   RUN useradd node
   RUN mkdir -p /var/log/exported && chown node:node /var/log/exported
   USER node
   RUN touch /var/log/exported/examplefile
   VOLUME ["/var/log/exported"]
   ```

2. タスク定義の volumes セクションで、名前を application_logs としてボリュームを定義します。

   ```
   "volumes": [
     {
       "name": "application_logs"
     }
   ]
   ```

3. containerDefinitions セクションで、アプリケーションのコンテナ定義を作成します。これにより、ストレージがマウントされます。containerPath 値は、Dockerfile の VOLUME ディレクティブで指定された絶対パスと一致する必要があります。

```
"containerDefinitions": [
  {
    "name": "application1",
    "image": "my-repo/application",
    "cpu": 100,
    "memory": 100,
    "essential": true,
    "mountPoints": [
      {
        "sourceVolume": "application_logs",
        "containerPath": "/var/log/exported"
      }
    ]
  },
  {
    "name": "application2",
    "image": "my-repo/application",
    "cpu": 100,
    "memory": 100,
    "essential": true,
    "mountPoints": [
      {
        "sourceVolume": "application_logs",
        "containerPath": "/var/log/exported"
      }
    ]
  }
]
```

ホスト Amazon EC2 インスタンスのライフサイクルに紐付けされているコンテナに、空のデータボリュームを提供するには

Amazon EC2 インスタンスでホストされているタスクの場合、バインドマウントを使用して、ホスト Amazon EC2 インスタンスのライフサイクルにデータを紐付けすることができます。これは、host パラメータを使用して sourcePath 値を指定することで設定できます。sourcePath に存在するファイルはすべて、containerPath の値でコンテナに表示されます。containerPath

に書き込まれたファイルは、ホストの Amazon EC2 インスタンス上の sourcePath 値に書き込まれます。

> **Important**
>
> Amazon ECS は、Amazon EC2 インスタンス間でストレージを同期しません。永続的ストレージを使用するタスクは、使用可能なキャパシティーのあるクラスター内の Amazon EC2 インスタンスに配置できます。タスクを停止して再び開始した後に永続的ストレージが必要な場合は、タスクの開始時に毎回 AWS CLI の start-task コマンドを使用して、同一の Amazon EC2 インスタンスを指定し直す必要があります。永続ストレージに Amazon EFS ボリュームを使用することもできます。詳細については、「Amazon ECS での Amazon EFS ボリュームの使用」を参照してください。

1. タスク定義の volumes セクションで、name および sourcePath 値を使用して、バインドマウントを定義します。次の例では、ホスト Amazon EC2 インスタンスはコンテナ内にマウントしたい /ecs/webdata のデータを含めます。

```
"volumes": [
  {
    "name": "webdata",
    "host": {
      "sourcePath": "/ecs/webdata"
    }
  }
]
```

2. containerDefinitions セクションで、定義したバインドマウントの名前を参照する mountPoints 値と、コンテナにバインドマウントをマウントするための containerPath 値を使用して、コンテナを定義します。

```
"containerDefinitions": [
  {
    "name": "web",
    "image": "public.ecr.aws/docker/library/nginx:latest",
    "cpu": 99,
    "memory": 100,
    "portMappings": [
      {
        "containerPort": 80,
```

```
          "hostPort": 80
        }
      ],
      "essential": true,
      "mountPoints": [
        {
          "sourceVolume": "webdata",
          "containerPath": "/usr/share/nginx/html"
        }
      ]
    }
  ]
```

定義したボリュームを異なる場所にある複数のコンテナにマウントする

タスク定義でデータボリュームを定義し、そのボリュームをさまざまなコンテナのさまざまな場所にマウントできます。例えば、ホストコンテナの /data/webroot に、ウェブサイトのデータフォルダがあるとします。そのデータボリュームを、異なるドキュメントルートを持つ 2 つの異なる Web サーバーに、読み取り専用としてマウントしたい場合があります。

1. タスク定義の volumes セクションで、名前を webroot、ソースパスを /data/webroot としてデータボリュームを定義します。

```
"volumes": [
  {
    "name": "webroot",
    "host": {
      "sourcePath": "/data/webroot"
    }
  }
]
```

2. containerDefinitions セクションで、各ウェブサーバーのコンテナを定義しています。各コンテナの mountPoints で、webroot ボリュームを、そのコンテナのドキュメントルートを参照する containerPath 値に関連付けます。

```
"containerDefinitions": [
  {
    "name": "web-server-1",
    "image": "my-repo/ubuntu-apache",
    "cpu": 100,
```

```
      "memory": 100,
      "portMappings": [
        {
          "containerPort": 80,
          "hostPort": 80
        }
      ],
      "essential": true,
      "mountPoints": [
        {
          "sourceVolume": "webroot",
          "containerPath": "/var/www/html",
          "readOnly": true
        }
      ]
    },
    {
      "name": "web-server-2",
      "image": "my-repo/sles11-apache",
      "cpu": 100,
      "memory": 100,
      "portMappings": [
        {
          "containerPort": 8080,
          "hostPort": 8080
        }
      ],
      "essential": true,
      "mountPoints": [
        {
          "sourceVolume": "webroot",
          "containerPath": "/srv/www/htdocs",
          "readOnly": true
        }
      ]
    }
  ]
```

volumesFrom を使用して別のコンテナからボリュームをマウントするには

Amazon EC2 インスタンスでホストされているタスクの場合、あるコンテナに 1 つ以上のボリュームを定義し、(同じタスクの) 異なるコンテナ定義で volumesFrom パラメータを使用し

て、sourceContainer のすべてのボリュームを、最初に定義されていたマウントポイントにマウントできます。volumesFrom パラメータは、タスク定義で定義されたボリューム、および Dockerfile でイメージに組み込まれたボリュームに適用されます。

1. (オプション) イメージに組み込まれているボリュームを共有するには、Dockerfile の VOLUME 命令を使用します。次の Dockerfile の例では、httpd イメージを使用しボリュームを追加して、それを Apache ドキュメントルートの dockerfile_volume にマウントしています。これは、httpd ウェブサーバーが使用するフォルダーです。

   ```
   FROM httpd
   VOLUME ["/usr/local/apache2/htdocs/dockerfile_volume"]
   ```

   この Dockerfile を使用してイメージを構築し、Docker ハブなどのレポジトリにプッシュして、タスク定義で使用できます。以下のステップで使用する my-repo/httpd_dockerfile_volume イメージ例は、上記の Dockerfile から構築したものです。

2. コンテナの他のボリュームとマウントポイントを定義するタスク定義を作成します。この例の volumes セクションでは、Docker デーモンによって管理される空のボリューム empty を作成します。また、host_etc と呼ばれるホストボリュームも定義されています。これは、ホストコンテナインスタンス上の /etc フォルダをエクスポートします。

   ```
   {
     "family": "test-volumes-from",
     "volumes": [
       {
         "name": "empty",
         "host": {}
       },
       {
         "name": "host_etc",
         "host": {
           "sourcePath": "/etc"
         }
       }
     ],
   ```

   コンテナ定義セクションで、先ほど定義したボリュームをマウントするコンテナを作成します。この例では、web コンテナが empty および host_etc ボリュームをマウントします。このコンテナは、Dockerfile 内のボリュームでビルドされたイメージを使用しています。

```
"containerDefinitions": [
    {
      "name": "web",
      "image": "my-repo/httpd_dockerfile_volume",
      "cpu": 100,
      "memory": 500,
      "portMappings": [
        {
          "containerPort": 80,
          "hostPort": 80
        }
      ],
      "mountPoints": [
        {
          "sourceVolume": "empty",
          "containerPath": "/usr/local/apache2/htdocs/empty_volume"
        },
        {
          "sourceVolume": "host_etc",
          "containerPath": "/usr/local/apache2/htdocs/host_etc"
        }
      ],
      "essential": true
    },
```

volumesFrom を使用して、web コンテナに関連付けられているすべてのボリュームをマウントする、別のコンテナを作成します。web コンテナ上にあるすべてのボリュームは、同様に busybox コンテナにもマウントされます。これには、my-repo/httpd_dockerfile_volume イメージのビルドに使用された Dockerfile で指定されたボリュームも含まれます。

```
    {
      "name": "busybox",
      "image": "busybox",
      "volumesFrom": [
        {
          "sourceContainer": "web"
        }
      ],
      "cpu": 100,
      "memory": 500,
```

```
      "entryPoint": [
        "sh",
        "-c"
      ],
      "command": [
        "echo $(date) > /usr/local/apache2/htdocs/empty_volume/date && echo $(date)
 > /usr/local/apache2/htdocs/host_etc/date && echo $(date) > /usr/local/apache2/
htdocs/dockerfile_volume/date"
      ],
      "essential": false
    }
  ]
}
```

このタスクが実行されると、2 つのコンテナでボリュームがマウントされ、busybox コンテナの command がファイルに日付と時刻を書き込みます。このファイルは、各ボリュームフォルダに date の名前で保存されています。その後、フォルダは web コンテナによって表示されるウェブサイトで見ることができます。

> **Note**
>
> busybox コンテナはクイックコマンドを実行して終了するため、コンテナ定義で "essential": false として設定する必要があります。そうしなければ、終了時にタスク全体が停止します。

Amazon ECS のコンテナスワップメモリ空間の管理

Amazon ECS を使用すると、Linux ベースの Amazon EC2 インスタンスで、スワップメモリ空間の使用状況をコンテナレベルで制御できるようになります。コンテナ単位のスワップ設定を使用すると、タスク定義内で各コンテナについて、スワップを有効または無効にできます。これが有効化されたコンテナでは、使用されるスワップ領域の最大量を制限できます。例えば、レイテンシーが厳しいコンテナでは、スワップを無効にすることができます。一方で、メモリ需要が一時的に高くなるコンテナでは、スワップを有効にすることで、コンテナのロード時にメモリ不足エラーが発生する可能性を減らすことができます。

コンテナのスワップ設定は、以下のコンテナ定義パラメータによって管理されます。

maxSwap

コンテナが使用できるスワップメモリの合計 (MiB 単位)。このパラメータは、docker run の --memory-swap オプションに変換されます。値はコンテナメモリの合計に maxSwap の値を加えた値です。

maxSwap の 0 値を指定した場合、コンテナはスワップを使用しません。許容値は、0 または任意の正の整数です。maxSwap パラメータを省略すると、コンテナは実行中のコンテナインスタンスのスワップ設定を使用します。maxSwap パラメータを使用するには、swappiness 値を設定する必要があります。

swappiness

これにより、コンテナのメモリスワップ動作を調整できます。swappiness の値が 0 であると、必要な場合を除きスワップは発生しません。swappiness の値が 100 の場合は、ページが積極的にスワップされます。使用できる値は、0 と 100 の間の整数です。swappiness パラメータを指定しない場合、デフォルト値の 60 が使用されます。maxSwap の値が指定されていない場合、このパラメータは無視されます。このパラメータは、docker run の --memory-swappiness オプションにマッピングされます。

次の例に、JSON での構文を示します。

```
"containerDefinitions": [{
        ...
        "linuxParameters": {
            "maxSwap": integer,
            "swappiness": integer
        },
        ...
}]
```

考慮事項

コンテナごとのスワップ構成を使用する場合は、次の点を考慮してください。

- スワップ領域は有効化され、使用するコンテナ向けのタスクをホストする、Amazon EC2 インスタンスに割り当てられる必要があります。Amazon ECS 最適化 AMI のデフォルトでは、スワップは有効になっていません。この機能を使用するには、インスタンスでスワップを有効にする必要があります。詳細は「Amazon EC2 ユーザーガイド」の「インスタンスストアスワップボリュー

ム」または「How do I allocate memory to work as swap space in an Amazon EC2 instance?」を参照してください。

- スワップスペースのコンテナ定義パラメータは、EC2 を指定したタスク定義でのみサポートされます。これらのパラメータは、Fargate の Amazon ECS での使用のみを目的としたタスク定義ではサポートされていません。
- この機能は Linux コンテナでのみサポートされています。現在、Windows コンテナはサポートされていません。
- コンテナ定義の `maxSwap` および `swappiness` パラメータがタスク定義から省略されている場合、各コンテナの `swappiness` には、デフォルト値の `60` が設定されます。さらに、スワップの合計使用量は、コンテナのメモリ量の 2 倍に制限されます。
- Amazon Linux 2023 でタスクを使用している場合、`swappiness` パラメータはサポートされていません。

Amazon ECS マネージドインスタンスの Amazon ECS タスク定義の違い

Amazon ECS マネージドインスタンスを使用するには、Amazon ECS マネージドインスタンスの起動タイプを使用するようにタスク定義を設定する必要があります。Amazon ECS マネージドインスタンスを使用する際には、追加の考慮事項があります。

タスク定義パラメータ

Amazon ECS マネージドインスタンスを使用するタスクは、利用可能な Amazon ECS のタスク定義パラメータの大部分に対応しています。ただし一部のパラメータには、Amazon ECS マネージドインスタンスタスクでの使用には特定の動作制限が伴います。

次のタスク定義パラメータは Amazon ECS マネージドインスタンスタスクでは無効となっています。

- `disableNetworking`
- `dnsSearchDomains`
- `dnsServers`
- `dockerSecurityOptions`
- `dockerVolumeConfiguration`
- `ephemeralStorage`
- `extraHosts`

- fsxWindowsFileServerVolumeConfiguration
- hostname
- inferenceAccelerator
- ipcMode
- links
- maxSwap
- proxyConfiguration
- sharedMemorySize
- sourcepath ボリューム
- swappiness
- tmpfs

以下のタスク定義パラメータは Amazon ECS マネージドインスタンスタスクで有効ですが、注意すべき制限があります。

- networkConfiguration – Amazon ECS マネージドインスタンスタスクは、awsvpc または host ネットワークモードを使用します。
- placementConstraints – 以下の制約属性がサポートされています。
  - ecs.subnet-id
  - ecs.availability-zone
  - ecs.instance-type
  - ecs.cpu-architecture
- requiresCompatibilities – タスク定義が Amazon ECS マネージドインスタンスと互換性があることを確認するには、MANAGED_INSTANCES を含める必要があります。
- resourceRequirement – InferenceAccelerator はサポートされていません。
- operatingSystemFamily – Amazon ECS マネージドインスタンスは LINUX を使用します。
- volumes - sourcePath が設定されたバインドマウントを使用するときは、パスがホスト上の書き込み可能なディレクトリをポイントしている必要があります。Amazon ECS マネージドインスタンスファイルシステムの一部は読み取り専用です。書き込み可能なディレクトリには、/var および /tmp が含まれます。詳細については、「Amazon ECS でのバインドマウントの使用」を参照してください。

タスク定義が Amazon ECS マネージドインスタンスで使用できることを検証するために、タスク定義を登録する際に以下を指定できます。

- AWS マネジメントコンソール の [Requires Compatibilities (互換性が必要)] フィールドで、MANAGED_INSTANCES を指定します。
- AWS CLI で、--requires-compatibilities オプションを指定します。
- Amazon ECS API で、requiresCompatibilities フラグを指定します。

Fargate における Amazon ECS タスク定義の違い

Fargate を使用するには、Fargate 起動タイプを使用するようにタスク定義を設定する必要があります。Fargate の使用については他にもいくつかの考慮事項があります。

タスク定義パラメータ

Fargate を使用するタスクは利用可能な Amazon ECS タスク定義パラメータのすべてをサポートするわけではありません。一部のパラメータはサポートされていません。また、Fargate タスクでは動作が異なるパラメータがあります。

次のタスク定義パラメータは Fargateタスクでは無効です:

- disableNetworking
- dnsSearchDomains
- dnsServers
- dockerSecurityOptions
- extraHosts
- gpu
- ipcMode
- links
- placementConstraints
- privileged
- maxSwap
- swappiness

以下のタスク定義パラメータはFargateタスクで有効ですが、注意すべき制限があります:

- `linuxParameters` – コンテナに適用される Linux 固有のオプションを指定する場合、`capabilities` に追加できる機能は `CAP_SYS_PTRACE` のみです。`devices`、`sharedMemorySize`、および `tmpfs` パラメータはサポートされません。詳細については、「Linux パラメータ」を参照してください。
- `volumes` – Fargateタスクはバインドマウントのホストボリュームのみをサポートするため、`dockerVolumeConfiguration` パラメータはサポートされません。詳細については、「ボリューム」を参照してください。
- cpu - AWS Fargate での Windows コンテナの場合、値は 1 vCPU 未満にすることはできません。
- `networkConfiguration` – Fargate タスクは、常に `awsvpc` ネットワークモードを使用します。

タスク定義が Fargateでの使用が有効であることを確認するために、タスク定義を登録する際に以下を指定できます:

- AWS マネジメントコンソール の [Requires Compatibilities (互換性が必要)] フィールドで、`FARGATE` を指定します。
- AWS CLI で、`--requires-compatibilities` オプションを指定します。
- Amazon ECS API で、`requiresCompatibilities` フラグを指定します。

オペレーティングシステムとアーキテクチャ

AWS Fargate のタスク定義とコンテナの定義を構成する場合、コンテナが実行するオペレーティングシステムを指定する必要があります。以下のオペレーティングシステムが AWS Fargate でサポートされています。

- Amazon Linux 2

> **Note**
>
> Linux コンテナは、ホストオペレーティングシステムのカーネルおよびカーネル設定のみを使用することに留意してください。例えば、カーネル構成には `sysctl` システムコントロールが含まれます。Linux コンテナイメージは、任意の Linux ディストリビューションのファイルとプログラムを含むベースイメージから作成できます。CPU アーキテクチャが一致すると、どの OS 上で実行しているどの Linux コンテナイメージからでもコンテナを実行できます。

- Windows Server 2019 Full

- Windows Server 2019 Core
- Windows Server 2022 Full
- Windows Server 2022 Core

AWS Fargate で Windows コンテナを実行する場合、X86_64 CPU アーキテクチャを備えている必要があります。

AWS Fargate で Linux コンテナを実行する場合、ARM ベースのアプリケーションに X86_64 CPU アーキテクチャ、または ARM64 アーキテクチャを使用できます。詳細については、「the section called “64 ビット ARM ワークロードでのタスク定義”」を参照してください。

タスク CPU とメモリ

AWS Fargate の Amazon ECS タスク定義では、CPU とメモリをタスクレベルで指定する必要があります。ほとんどのユースケースでは、タスクレベルでこれらのリソースを指定するだけで十分です。以下の表に、タスクレベルの CPU とメモリの有効な組み合わせを示します。タスク定義では、メモリの値を MiB または GB の文字列として指定できます。例えば、メモリの値を `3072` (MiB) または `3 GB` (GB) のいずれかで指定できます。JSON ファイルでは、CPU 値を CPU ユニットまたは仮想 CPU (vCPU) の文字列として指定できます。例えば、CPU 値を `1 vCPU` (vCPU) または `1024` (CPU ユニット) として指定できます。

CPU の値	メモリの値	AWS Fargate でサポートされるオペレーティングシステム
256 (.25 vCPU)	512 MiB、1 GB、2 GB	Linux
512 (.5 vCPU)	1 GB、2 GB、3 GB、4 GB	Linux
1,024 (1 vCPU)	2 GB、3 GB、4 GB、5 GB、6 GB、7 GB、8 GB	Linux、Windows
2,048 (2 vCPU)	4 GB ~ 16 GB (1 GB のインクリメント)	Linux、Windows
4,096 (4 vCPU)	8 GB ~ 30 GB (1 GB のインクリメント)	Linux、Windows
8192 (8 vCPU)	16 GB ~ 60 GB (4 GB のインクリメント)	Linux

CPU の値	メモリの値	AWS Fargate でサポートされるオペレーティングシステム
Note このオプションには Linux プラットフォーム 1.4.0 以降が必要です。		
16384 (16 vCPU) Note このオプションには Linux プラットフォーム 1.4.0 以降が必要です。	32 GB ~ 120 GB (8 GB のインクリメント)	Linux
32768 (32 vCPU) Note このオプションには Linux プラットフォーム 1.4.0 以降が必要です。	60 GB、120 GB、244 GB	Linux

タスクネットワーク

AWS Fargate の Amazon ECS を使用するタスクでは、awsvpc ネットワークモードが必要です。これは各タスクに Elastic Network Interface を提供します。このネットワークモードを使用したタスクの実行またはサービスの作成時に、ネットワークインターフェイスにアタッチするサブネットを 1 つ以上、またはネットワークインターフェイスに適用するセキュリティグループを 1 つ以上、指定する必要があります。

パブリックサブネットを使用している場合は、ネットワークインターフェイスにパブリック IP アドレスを指定するかどうかを決定します。パブリックサブネットの Fargate タスクを使用してコンテナイメージをプルするには、タスクの Elastic Network Interface に、インターネットへのルートまたはリクエストをインターネットにルーティングできる NAT ゲートウェイを持つパブリック IP アドレスが割り当てられている必要があります。　　プライベートサブネットのFargateタスクでコンテナイメージをプルするには、リクエストをインターネットにルーティングするためのNATゲートウェイがサブネットに必要です。Amazon ECR でコンテナイメージをホストする場合、 Amazon ECR を設定してインターフェイス VPC エンドポイントを使用するようにできます。この場合、タスクのプライベート IPv4 アドレスがイメージのプルに使用されます。Amazon ECRインターフェイスエンドポイントの詳細については、Amazon Elastic Container Registry ユーザーガイドの Amazon ECR Interface VPC エンドポイント (AWS PrivateLink)を参照してください。

Fargate サービス向け `networkConfiguration` セクションの例を以下に示します。

```
"networkConfiguration": {
   "awsvpcConfiguration": {
      "assignPublicIp": "ENABLED",
      "securityGroups": [ "sg-12345678" ],
      "subnets": [ "subnet-12345678" ]
   }
}
```

タスクリソースの制限

AWS Fargate での Linux コンテナのAmazon ECS タスク定義では、コンテナに設定するリソース制限を定義するための `ulimits` パラメータがサポートされます。

AWS Fargate での Windows の Amazon ECS タスク定義では、コンテナに設定するリソース制限を定義するための `ulimits` パラメータがサポートされます。

Fargate でホストされる Amazon ECS タスクは、オペレーションシステムで設定されたデフォルトのリソース制限値を使用します。ただし、`nofile` リソース制限パラメータを除きます。`nofile` リソース制限は、コンテナが使用できるオープンファイルの数の制限を設定します。Fargate では、デフォルトの `nofile` ソフト制限は　65535 で、ハード制限は 65535 です。両方の制限の値を設定できます (最大 1048576)。

以下は、2 倍になったカスタム `nofile` 制限を定義する方法を示すタスク定義スニペットの例です。

```
"ulimits": [
    {
```

```
        "name": "nofile",
        "softLimit": 2048,
        "hardLimit": 8192
    }
]
```

調整可能なその他のリソース制限の詳細については、リソースの制限 を参照してください。

ログ記録

イベントログ

Amazon ECS は、実行したアクションを EventBridge にログ記録します。Amazon ECS の Eventbridge イベントを使用して、Amazon ECS クラスター、サービスおよびタスクの現在の状態に関するほぼリアルタイムの通知を受け取れます。さらに、これらのイベントに対応するアクションを自動化できます。詳細については、「EventBridge を使用して Amazon ECS エラーへの対応を自動化する」を参照してください。

タスクライフサイクルログ

Fargate で実行されるタスクは、タイムスタンプを公開してタスクライフサイクルの状態を追跡します。AWS CLI および SDK でタスクを説明することにより、AWS マネジメントコンソール のタスク詳細でタイムスタンプを確認できます。例えば、タイムスタンプを使用して、タスクがコンテナイメージのダウンロードに費やした時間を評価し、コンテナイメージのサイズを最適化するか、Seekable OCI インデックスを使用するかを判断できます。コンテナイメージのプラクティスに関する詳細については、「Amazon ECS コンテナイメージのベストプラクティス」を参照してください。

アプリケーションログ

AWS Fargate の Amazon ECS タスク定義はログ設定の `awslogs`、`splunk`、および `awsfirelens` ログドライバーをサポートします。

この`awslogs`ログドライバーは、Amazon CloudWatch Logs にログ情報を送信するように Fargate タスクを設定します。以下に、`awslogs` ログドライバーが設定されているタスク定義のスニペットを示します。

```
"logConfiguration": {
   "logDriver": "awslogs",
   "options": {
      "awslogs-group" : "/ecs/fargate-task-definition",
```

```
        "awslogs-region": "us-east-1",
        "awslogs-stream-prefix": "ecs"
    }
}
```

タスク定義で awslogs ログドライバーを使用してコンテナログをCloudWatch Logs に送信する方法の詳細については、Amazon ECS ログを CloudWatch に送信するを参照してください。

タスク定義での awsfirelens ログドライバーの詳細については、「Amazon ECS ログを AWS サービスまたは AWS Partner に送信する」を参照してください。

タスク定義での splunk ログドライバーの使用の詳細については、「splunk ログドライバー」を参照してください。

タスクストレージ

Fargate でホストされた Amazon ECS タスクでは、次のストレージタイプがサポートされています:

- Amazon EBS ボリュームは、データ集約型のコンテナ化されたワークロード向けに、費用対効果と耐久性に優れた高性能なブロックストレージを提供します。詳細については、「Amazon ECS での Amazon EBS ボリュームの使用」を参照してください。
- 永続的ストレージ用のAmazon EFS ボリューム。詳細については、「Amazon ECS での Amazon EFS ボリュームの使用」を参照してください。
- エフェメラルストレージ用のバインドマウント。詳細については、「Amazon ECS でのバインドマウントの使用」を参照してください。

Seekable OCI (SOCI) を使ったコンテナイメージの遅延読み込み

Linux プラットフォーム バージョン 1.4.0 を使用する Fargate 上の Amazon ECS タスクは、Seekable OCI (SOCI) を使用してタスクをより速く開始できます。SOCI では、コンテナは起動前にイメージプルに数秒しかかからないため、イメージがバックグラウンドでダウンロードされている間、環境のセットアップとアプリケーションのインスタンス化に時間を割けます。これは遅延読み込みと呼ばれています。Fargate が Amazon ECS タスクを開始すると、Fargate はタスク内のイメージに SOCI インデックスが存在するかどうかを自動的に検出し、イメージ全体がダウンロードされるのを待たずにコンテナを起動します。

SOCI インデックスなしで実行されるコンテナの場合、コンテナイメージはコンテナを起動する前に完全にダウンロードされます。この動作は、Fargate の他のすべてのプラットフォームバージョンと Amazon EC2 インスタンス上の Amazon ECS 最適化された AMI でも同じです。

Seekable OCI (SOCI) は AWS によって開発されたオープンソースのテクノロジーであり、コンテナイメージを遅延読み込みすることでコンテナをより速く起動できます。SOCI は、既存のコンテナイメージ内のファイルのインデックス (SOCI インデックス) を作成することで機能します。このインデックスは、イメージ全体をダウンロードする前にコンテナイメージから個々のファイルを抽出できるため、コンテナをより速く起動するのに役立ちます。SOCI インデックスは、コンテナレジストリ内のイメージと同じリポジトリにアーティファクトとして保存する必要があります。インデックスはイメージのコンテンツの信頼できるソースなので、信頼できるソースからの SOCI インデックスのみを使用してください。詳細については、「コンテナイメージを遅延ロードするためのSeekable OCIの導入」を参照してください。

SOCI の使用を検討しているお客様は、SOCI Index Manifest v2 のみを使用できます。Fargate で SOCI を以前使用したことのある既存のお客様は、引き続き SOCI Index Manifest v1 を使用できますが、SOCI Index Manifest v2 に移行することを強くお勧めします。SOCI Index Manifest v2 は、一貫したデプロイを確保するために、コンテナイメージとその SOCI インデックスの間に明示的な関係を作成します。

考慮事項

Fargate に SOCI インデックスを使用してコンテナイメージをタスクに遅延読み込ませる場合は、次の点を考慮してください。

- Linux プラットフォームバージョン `1.4.0` で実行されるタスクのみ SOCI インデックスを使用できます。Fargate で Windows コンテナを実行するタスクはサポートされていません。
- X86_64 または ARM64 CPU アーキテクチャ上で実行されるタスクがサポートされています。
- タスク定義内のコンテナイメージは、互換性のあるイメージレジストリに保存する必要があります。以下に互換性のあるレジストリを示します。
  - Amazon ECR プライベートレジストリ
- gzip 圧縮を使用する、または圧縮されていないコンテナイメージのみがサポートされます。zstd 圧縮を使用するコンテナイメージはサポートされていません。
- SOCI Index Manifest v2 の場合、SOCI インデックスマニフェストを生成すると、SOCI インデックスの注釈を追加するときにコンテナイメージマニフェストが変更されます。これにより、新しいコンテナイメージダイジェストが作成されます。コンテナイメージのファイルシステムレイヤーの内容は変更されません。
- SOCI Index Manifest v2 の場合、コンテナイメージがすでにコンテナイメージリポジトリに保存されている場合、SOCI インデックスが生成されたら、コンテナイメージを再プッシュする必要があります。コンテナイメージを再プッシュしても、ファイルシステムレイヤーが重複することでストレージコストは増加せず、新しいマニフェストファイルのみをアップロードします。

- 圧縮サイズが 250 MiB より大きいコンテナイメージを使用して遅延読み込みを試すことをお勧めします。小さいイメージを読み込む時間が短くなる可能性は低くなります。
- 遅延読み込みによってタスクの開始にかかる時間が変わる可能性があるため、Elastic Load Balancing のヘルスチェック猶予期間など、さまざまなタイムアウトを変更する必要がある場合があります。
- コンテナイメージが遅延ロードされないようにするには、SOCI インデックスをアタッチせずにコンテナイメージを再プッシュする必要があります。

Seekable OCI インデックスの作成

コンテナイメージを遅延ロードするには、SOCI インデックス (メタデータファイル) を作成し、コンテナイメージと共にコンテナイメージリポジトリに保存する必要があります。SOCI インデックスを作成しプッシュするには、GitHub にあるオープンソースの soci-snapshotter CLI ツールを使用できます。または、CloudFormation AWS SOCI Index Builder をデプロイすることもできます。これは、コンテナイメージが Amazon ECR にプッシュされるときに SOCI インデックスを自動的に作成してプッシュするサーバーレスソリューションです。ソリューションとインストール手順の詳細については、GitHub の「CloudFormation AWS SOCI Index Builder」を参照してください。CloudFormation AWS SOCI Index Builder は SOCI の導入を自動化できる手段ですが、オープンソースの SOCI ツールの方がインデックス生成に関してはより柔軟性があり、継続的インテグレーションと継続的デリバリー (CI/CD) パイプラインにインデックス生成を統合できます。

> **Note**
>
> イメージの SOCI インデックスを作成するには、そのイメージが `soci-snapshotter` を実行しているコンピュータ上の containerd イメージストアに存在する必要があります。イメージが Docker イメージストアにある場合、イメージは見つかりません。

タスクが遅延読み込みを使用したことの検証

SOCI を使用してタスクが遅延読み込みされたことを確認するには、タスク内部からタスクメタデータエンドポイントを確認します。タスクメタデータエンドポイントバージョン 4 をクエリすると、クエリ元のコンテナのデフォルトパスに `Snapshotter` フィールドが存在します。さらに、`/task` パス内のコンテナごとに `Snapshotter` フィールドがあります。このフィールドのデフォルト値は `overlayfs` であり、SOCI が使用されている場合、このフィールドは `soci` に設定されます。コンテナイメージに SOCI インデックスマニフェスト v2 がアタッチされていることを確認するには、AWS CLI を使用して Amazon ECR からイメージインデックスを取得できます。

```
IMAGE_REPOSITORY=r
IMAGE_TAG=latest

aws ecr batch-get-image \
    --repository-name=$IMAGE_REPOSITORY \
    --image-ids imageTag=$IMAGE_TAG \
    --query 'images[0].imageManifest' --output text | jq -r '.manifests[] |
 select(.artifactType=="application/vnd.amazon.soci.index.v2+json")'
```

コンテナイメージに SOCI インデックスマニフェスト v1 がアタッチされていることを確認するには、OCI Referrers API を使用できます。

```
ACCOUNT_ID=111222333444
AWS_REGION=us-east-1
IMAGE_REPOSITORY=nginx-demo
IMAGE_TAG=latest
IMAGE_DIGEST=$(aws ecr describe-images --repository-name $IMAGE_REPOSITORY --image-ids
 imageTag=$IMAGE_TAG --query 'imageDetails[0].imageDigest' --output text)
ECR_PASSWORD=$(aws ecr get-login-password)

curl \
    --silent \
    --user AWS:$ECR_PASSWORD \
    https://$ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/v2/$IMAGE_REPOSITORY/
referrers/$IMAGE_DIGEST?artifactType=application%2Fvnd.amazon.soci.index.v1%2Bjson | jq
 -r '.'
```

Windows を実行している EC2 インスタンスでの Amazon ECS タスク定義の違い

EC2 Windows インスタンスで実行されるタスクは、使用可能なすべての Amazon ECS タスク定義パラメータをサポートしているわけではありません。サポートされていないパラメータもあり、また一部のパラメータでは動作が異なることがあります。

以下のタスク定義パラメータは、Amazon EC2 Windows タスク定義ではサポートされていません。

- `containerDefinitions`
  - `disableNetworking`

  - dnsServers
  - dnsSearchDomains
  - extraHosts
  - links
  - linuxParameters
  - privileged
  - readonlyRootFilesystem
  - user
  - ulimits
- volumes
  - dockerVolumeConfiguration
- cpu

  Windows コンテナではコンテナレベル CPU を指定することをお勧めします。
- memory

  Windows コンテナではコンテナレベルメモリを指定することをお勧めします。
- proxyConfiguration
- ipcMode
- pidMode
- taskRoleArn

  EC2 Windows インスタンス上のタスク向け IAM ロールには追加の設定が必要ですが、それらの設定の多くは Linux コンテナインスタンス上のタスクの IAM ロールの設定と類似しています。詳細については、「the section called “Amazon EC2 Windows インスタンスの追加設定”」を参照してください。

コンソールを使用した Amazon ECS タスク定義の作成

タスクまたはサービスとして実行するアプリケーションを定義できるように、タスク定義を作成します。

外部起動タイプのタスク定義を作成するときは、JSON エディタを使用してタスク定義を作成し、requireCapabilities パラメータを EXTERNAL に設定する必要があります。

タスク定義は、コンソールエクスペリエンスを使用するか、JSON ファイルを指定して作成できます。JSON エディタを使用すると、Amazon Q に推奨値を提供させることができます。詳細については、Amazon Q Developer を使用して Amazon ECS コンソールでタスク定義のレコメンデーションを生成するを参照してください。

JSON 検証

Amazon ECS コンソールの JSON エディタは、JSON ファイル内で以下を検証します。

- ファイルが有効な JSON ファイルであること。
- ファイルに外部キーが含まれていないこと。
- ファイルに `familyName` パラメータが含まれていること。
- `containerDefinitions` の下に少なくとも 1 つのエントリがあること。

CloudFormation スタック

次の動作は、2023 年 1 月 12 日以前に新しい Amazon ECS コンソールで作成されたタスク定義に適用されます。

タスク定義の作成時、Amazon ECS コンソールは、名前が `ECS-Console-V2-TaskDefinition-` で始まる CloudFormation スタックを自動的に作成します。AWS CLI または AWS SDK を使用してタスク定義を登録解除した場合は、手動でタスク定義スタックを削除する必要があります。詳細については、「CloudFormation ユーザーガイド」の「スタックの削除」を参照してください。

2023 年 1 月 12 日以降に作成されたタスク定義では、CloudFormation スタックの自動的な作成は行われません。

手順

Amazon ECS console

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで、タスク定義 を選択します。
3. [新しいタスク定義の作成] メニューで、[新しいタスク定義を作成する] を選択します。
4. [Task definition family] (タスク定義ファミリー) を使用する場合、タスク定義ファミリーには、タスク定義に一意の名前を指定します。

5. [起動タイプ] で、アプリケーション環境を選択します。コンソールのデフォルトは、AWS Fargate (サーバーレス) です。Amazon ECS は、この値を使用して検証を実行し、タスク定義パラメータがインフラストラクチャタイプに対して有効であることを確認します。
6. [Operating system/Architecture] (オペレーティングシステム/アーキテクチャ) を使用する場合、タスクのオペレーティングシステムと CPU アーキテクチャを選択します。

   タスクを 64 ビット ARM アーキテクチャで実行するには、[Linux/ARM64] を選択します。詳細については、「the section called “ランタイムプラットフォーム”」を参照してください。

   Windows コンテナで AWS Fargate タスクを実行するには、サポートされている Windows オペレーティングシステムを選択します。詳細については、「the section called “オペレーティングシステムとアーキテクチャ”」を参照してください。
7. [Task size] (タスクサイズ) では、タスク用に予約する CPU とメモリの値を選択します。CPU 値は vCPUs として指定され、メモリは GB として指定されます。

   Fargate でホストされるタスクの場合、次の表に有効な CPU とメモリの組み合わせを示します。

CPU の値	メモリの値	AWS Fargate でサポートされるオペレーティングシステム
256 (.25 vCPU)	512 MiB、1 GB、2 GB	Linux
512 (.5 vCPU)	1 GB、2 GB、3 GB、4 GB	Linux
1,024 (1 vCPU)	2 GB、3 GB、4 GB、5 GB、6 GB、7 GB、8 GB	Linux、Windows
2,048 (2 vCPU)	4 GB ~ 16 GB (1 GB のインクリメント)	Linux、Windows
4,096 (4 vCPU)	8 GB ~ 30 GB (1 GB のインクリメント)	Linux、Windows
8192 (8 vCPU)	16 GB ~ 60 GB (4 GB のインクリメント)	Linux

CPU の値	メモリの値	AWS Fargate でサポートされるオペレーティングシステム
Note このオプションには Linux プラットフォーム 1.4.0 以降が必要です。		
16384 (16 vCPU) Note このオプションには Linux プラットフォーム 1.4.0 以降が必要です。	32 GB ~ 120 GB (8 GB のインクリメント)	Linux
32768 (32 vCPU) Note このオプションには Linux プラットフォーム 1.4.0 以降が必要です。	60 GB、120 GB、244 GB	Linux

EC2 インスタンスまたは外部インスタンスを使用するタスクの場合、サポートされるタスク CPU 値は 128 CPU ユニット (0.125 vCPU) から 196608 CPU ユニット (192 vCPU) の範囲となります。

メモリ値を GB 単位で指定するには、数字に続けて [GB] と入力します。たとえば、[メモリ値] を 3 GB に設定するには、[3GB] と入力します。

> **Note**
>
> タスクレベル CPU およびメモリのパラメータは Windows コンテナでは無視されます。

8. [Network mode] (ネットワークモード) の場合、使用するネットワークモードを選択します。デフォルトは、awsvpc モードです。詳細については、Amazon ECS タスクネットワーキングを参照してください。

   [ポートマッピング] で、[ホストポート] に [ブリッジ] を選択した場合は、コンテナ用に予約するコンテナインスタンスのポート番号を入力します。

9. (オプション) [タスクロール] セクションを展開してタスク向け AWS Identity and Access Management (IAM) ロールを設定できます。

   a. [Task role] (タスクロール) では、タスクに割り当てる IAM ロールを選択します。タスク IAM ロールは、タスク内のコンテナが AWS API 操作を呼び出すアクセス許可を提供します。

   b. [タスク実行ロール] で、ロールを選択します。

      タスク実行ロールをいつ使用するかの詳細については、「the section called “タスク実行 IAM ロール”」を参照してください。ロールが不要な場合は、[なし] を選択します。

10. (オプション) タスク配置 セクションを展開して、配置の制約を追加します。タスク配置の制約により、組み込み属性またはカスタム属性を使用して、タスクの配置に使用されるコンテナインスタンスをフィルタリングできます。

11. (オプション) フォールトインジェクション セクションを展開して、フォールトインジェクションを有効にします。フォールトインジェクションを使用すると、特定の障害シナリオに対するアプリケーションの反応をテストできます。

12. タスク定義で定義するコンテナごとに、以下のステップを実行します。

    a. [Name] (名前) に、コンテナの名前を入力します。

    b. [Image URI] (イメージ URI) には、コンテナの開始に使用するイメージを指定します。Amazon ECR Public Gallery レジストリ内のイメージは、Amazon ECR Public レジストリ名の使用でのみ指定することができます。例えば、`public.ecr.aws/ecs/amazon-ecs-agent:latest` が指定されている場合は、Amazon ECR Public Gallery でホストされている Amazon Linux コンテナが使用されます。他のすべてのリポジトリ

の場合は、`repository-url/image:tag` または `repository-url/image@digest` の形式を使用してリポジトリを指定します。

c. イメージが Amazon ECR の外部のプライベートレジストリにある場合は、[プライベートレジストリ] で [プライベートレジストリ認証] をオンにします。その後、[Secrets Manager ARN または名前] で、シークレットの Amazon リソースネーム (ARN) を入力します。

d. [基本コンテナ] の場合、もしタスク定義に 2 つ以上のコンテナが定義されていれば、そのコンテナを基本と見なすかどうかを指定できます。コンテナが [基本] とマークされている場合は、そのコンテナが停止するとタスクは停止します。各タスク定義には、少なくとも 1 つの基本コンテナを含む必要があります。

e. ポートマッピングにより、コンテナはホスト上のポートにアクセスしてトラフィックを送受信できるようになります。[Port mappings] (ポートマッピング) で、次のいずれかを実行]ます。

- [awsvpc] ネットワークモードを使用する場合、[Container port] (コンテナポート) と [Protocol] (プロトコル) で、コンテナに使用するポートマッピングを選択します。
- [bridge] ネットワークモードを使用する場合、[Container port] (コンテナポート) と [Protocol] (プロトコル) で、コンテナに使用するポートマッピングを選択します。

追加のコンテナポートマッピングを指定するには、[Add more port mappings] (ポートマッピングを追加) を選択します。

f. コンテナにルート ファイル システムへの読み取り専用アクセスを付与するには、[読み取り専用ルートファイルシステム] で [読み取り専用] を選択します。

g. (オプション) [リソース割り当て制限] のタスクレベルの値とは異なるコンテナ レベルの CPU、GPU、およびメモリの制限を定義するには、次の手順を実行します。

- [CPU] には、Amazon ECS コンテナエージェントがそのコンテナ用に予約している CPU ユニット数を入力します。
- [GPU] では、コンテナインスタンス用の GPU ユニット数を入力します。

  GPU サポートを使用する Amazon EC2 インスタンスには、各 GPU について 1 つの GPU ユニットが装備されます。詳細については、「the section called “GPU ワークロード向けのタスク定義”」を参照してください。
- [メモリのハード制限] には、コンテナに提供するメモリの量を GB 単位で入力します。コンテナがハード制限を超えようとすると、コンテナは停止します。

- Docker 20.10.0 以降のデーモンではコンテナ用に最低 6 MiB のメモリを予約するため、コンテナ用に 6 MiB 未満のメモリを指定しないでください。

  Docker 19.03.13-ce 以前のデーモンではコンテナ用に最低 4 MiB のメモリを予約するため、コンテナ用に 4 MiB 未満のメモリを指定しないでください。

- [メモリのソフト制限] に、コンテナに予約するメモリのソフト制限 (GB 単位) を入力します。

  システムメモリに競合がある場合、Docker はコンテナのメモリ量をこのソフト制限値内に維持しようとします。タスクレベルのメモリを指定しない場合は、[メモリのハード制限] と [メモリのソフト制限] の一方または両方にゼロ以外の整数を指定する必要があります。両方を指定する場合は、[メモリのハード制限] がメモリの [ソフト制限] より大きい必要があります。

  この機能は Windows コンテナではサポートされません。

h. (オプション) [環境変数] セクションを展開し、コンテナに挿入する環境変数を指定します。環境変数は、キー値ペアを使用して個別に指定するか、Amazon S3 バケットでホストされている環境変数ファイルを指定して一括で指定できます。環境変数ファイルのフォーマット方法については、「個々の環境変数を Amazon ECS コンテナに渡す」を参照してください。

   シークレットストレージの環境変数を指定するときは、[キー] にシークレット名を入力します。次に、[ValueFrom] に、Systems Manager パラメータストアシークレットまたは Secrets Manager シークレットの完全な ARN を入力します。

i. (オプション) [Use log collection] (ログコレクションを使用) を選択し、ログ構成を指定します。使用可能なログドライバーごとに、指定するログドライバーオプションがあります。デフォルトのオプションでは、コンテナログを Amazon CloudWatch Logs に送信します。その他のログドライバーオプションは、AWS FireLens を使用して設定されます。詳細については、「Amazon ECS ログを AWS サービスまたは AWS Partner に送信する」を参照してください。

   以下では、各コンテナログの送信先について詳しく説明します。

   - Amazon CloudWatch — コンテナログを CloudWatch Logs に送信するようにタスクを設定します。デフォルトのログドライバーオプションが提供され、ユーザーに代わり CloudWatch ロググループを作成します。別のロググループ名を指定するには、ドライバーオプションの値を変更します。

- Splunk へのログのエクスポート — リモートサービスにログを送信する Splunk ドライバーにコンテナログを送信するタスクを設定します。Splunk Web サービスの URL を入力する必要があります。Splunk トークンは機密データとして扱われる可能性があるため、シークレットのオプションとして指定します。
- [Amazon Data Firehose へのログのエクスポート] — Firehose にコンテナログを送信するようタスクを設定します。Firehose 配信ストリームにログを送信するデフォルトのログドライバーオプションが提供されています。別の配信ストリーム名を指定するには、ドライバーオプションの値を変更します。
- Amazon Kinesis Data Streams へのログのエクスポート — Kinesis Data Streams にコンテナログを送信するようタスクを設定します。Kinesis Data Streams のストリームにログを送信するデフォルトのログドライバーオプションが提供されています。別のストリーム名を指定するには、ドライバーオプションの値を変更します。
- Amazon OpenSearch Service へのログのエクスポート — コンテナログを OpenSearch Service ドメインに送信するようタスクを設定します。ログドライバーオプションを提供する必要があります。
- Amazon S3 へのログのエクスポート — Amazon S3 バケットにコンテナログを送信するようタスクを設定します。デフォルトのログドライバーオプションが提供されていますが、有効な Amazon S3 バケット名を指定する必要があります。

j. (オプション) コンテナの追加パラメータを設定します。

このオプションを設定するには	この操作を行います	
再起動ポリシー これらのオプションは、コンテナの終了時にコンテナを再起動する再起動ポリシーを定義します。	[再起動ポリシー] を展開して以下の項目を設定します。 • コンテナの再起動ポリシーを有効にするには、[再起動ポリシーを有効にする] をオンにします。 • [無視された終了コード] には、整数コンテナの終了コードのカンマ区切りリストを指定します。指定された終了コードのいずれかでコンテナが終了した場合、Amazon ECS はコンテナの再起動を試みません。何も指定しない場合、Amazon ECS はいずれの終了コードも無視しません。 • [試行リセット期間] には、終了した際に再起動を試みる前にコンテナが実行する必要がある期間を整数 (秒) で指定します。Amazon ECS は、[試行リセット期間] で指定した秒ごと	

このオプションを設定するには	この操作を行います	
	にコンテナの再起動を 1 回だけ試みることができます。何も指定しない場合、再起動を試みる前にコンテナは 300 秒間実行する必要があります。	

このオプションを設定するには	この操作を行います	
HealthCheck これらはコンテナが正常かどうかを判断するコマンドです。詳細については、「コンテナのヘルスチェックを使用して Amazon ECS タスク状態を判定する」を参照してください。	HealthCheck を展開し、次の項目を設定します。 • [Command] (コマンド) には、カンマで区切られたコマンドのリストを入力します。コマンド引数を直接実行するための CMD、またはコンテナのデフォルトシェルでコマンドを実行するための CMD-SHELL を使用してコマンドを実行できます。これらのいずれも指定しない場合は CMD が使用されます。 • [Interval] (間隔) に、各ヘルスチェックの間隔を秒数で入力します。有効な値は 5～30 です。 • [Timeout] (タイムアウト) には、成功まで待機しているヘルスチェックが、失敗したと見なされるまでの期間 (秒単位) を入力します。有効な値は 2～60 です。 •	

このオプションを設定するには	この操作を行います	
	[Start period] (開始時期) には、ヘルスチェックコマンドを実行する前にコンテナがブートストラップするのを待つ期間 (秒単位) を入力します。有効な値は 0～300 です。 • [Retries] (再試行) には、障害発生時にヘルスチェックコマンドを再試行する回数を入力します。有効な値の範囲は 1～10 です。	
起動の依存関係の順序 このオプションは、コンテナの起動と停止の依存関係を定義します。コンテナには複数の依存関係を含めることができます。	[起動時の依存関係の順序] を展開し、次の項目を設定します。 a. [コンテナ依存関係を追加] を選択します。 b. [コンテナ] で、対象のコンテナを選択します。 c. [条件]で、起動時の依存条件を選択します。 依存関係を追加するには、[コンテナ依存関係を追加]を選択します。	

このオプションを設定するには	この操作を行います	
コンテナのタイムアウト これらのオプションは、コンテナをいつ起動および停止するかを決定します。	[コンテナタイムアウト] を展開し、次の項目を設定します。 • コンテナの依存関係を解決するための再試行を止めるまでの待機時間を設定するには、[タイムアウト開始時間] (秒) を入力します。 • コンテナが正常に終了しなかった場合にコンテナが停止されるまでの [タイムアウト停止時間] (秒) を指定します。	

このオプションを設定するには	この操作を行います
コンテナネットワーク設定 これらのオプションはコンテナ内でネットワークを使用するかどうかを決定します。	[コンテナネットワーク設定] を展開し、以下を設定します。 • コンテナネットワーキングを無効にするには、[ネットワーキングをオフにする] を選択します。 • コンテナに提供する DNS サーバー IP アドレスを設定するには、[DNS サーバー] に各サーバーの IP アドレスを個々の行に入力します。 • コンテナに提示された完全修飾されていないホスト名を検索するように DNS ドメインを設定するには、[DNS 検索ドメイン] で各ドメインを別々の行に入力します。 パターンは **^[a-zA-Z0-9-.]{0,253}[a-zA-Z0-9]$** です。 • コンテナのホスト名を設定するには、[ホスト

このオプションを設定するには	この操作を行います	
	名] にコンテナのホスト名を入力します。 • コンテナの /etc/hosts ファイルに追加するホスト名と IP アドレスマッピングを追加するには、[追加のホストを追加] を選択し、[ホスト名] と [IP アドレス] にホスト名と IP アドレスを入力します。	

このオプションを設定するには	この操作を行います
Docker の設定 これらは Dockerfile 内の値を上書きします。	[Docker 設定] を展開して以下の項目を設定します。 • [コマンド] には、コンテナに対する実行可能なコマンドを入力します。 このパラメータは、Docker Remote API の [コンテナの作成] セクションにある `Cmd` にマッピングされ、`COMMAND` オプションは `docker run` にマッピングされます。このパラメータは、Dockerfile 内の `CMD` 命令よりも優先されます。 • [エントリポイント] で、コンテナに渡される Docker エントリポイントを入力します。 このパラメータは、Docker Remote API の [コンテナの作成] セクションにある `Entrypoint` にマッピングされ、`--`

このオプションを設定するには	この操作を行います	
	`entrypoint` オプションは `docker run` にマッピングされます。このパラメータは、Dockerfile 内の `ENTRYPOINT` 命令よりも優先されます。 • [作業ディレクトリ] には、エントリポイントと提供されたコマンド命令をコンテナが実行するための、ディレクトリを入力します。 このパラメータは、Docker Remote API の [コンテナの作成] セクションにある `WorkingDir` にマッピングされ、`--workdir` オプションは `docker run` にマッピングされます。このパラメータは、Dockerfile 内の `WORKDIR` 命令よりも優先されます。	

このオプションを設定するには	この操作を行います	
[リソースの制限 (Ulimits)] この値は、オペレーティングシステムのデフォルトのリソースクォータ設定を上書きします。 このパラメータは、Docker Remote API の コンテナの作成 セクションにある `Ulimits` にマッピングされ、`--ulimit` オプションは docker run にマッピングされます。	[リソース制限 (ulimits)] を展開し、[ulimit の追加] を選択します。[制限名] で制限を選択します。次に、[ソフトリミット] と [ハードリミット] に値を入力します。 さらに ulimits を追加するには、[ulimit の追加] を選択します。	
Docker ラベル このオプションは、コンテナにメタデータを追加します。 このパラメータは、Docker Remote API の コンテナの作成 セクションにある `Labels` にマッピングされ、`--label` オプションは docker run にマッピングされます。	[Docker ラベル]を展開し、[キー値ペアを追加] を選択して、[キー] と [値] を入力します。 Docker ラベルをさらに追加するには、[キー値ペアを追加] を選択します。	

k. (オプション) [コンテナを追加する]をクリックして、タスク定義にコンテナを追加します。

13. (オプション) [ストレージ] セクションを使用して、Fargate でホストされるタスクの一時ストレージ量を拡張することができます。またこのセクションを使用して、タスクのデータボリューム設定を追加することもできます。

- Fargate タスクに使用可能なエフェメラルストレージをデフォルト値の 20 gibibytes (GiB) を超えて拡張するには、[Amount] (量) に最大で 200 GiB までの値を入力します。

14. (オプション) タスク定義のデータ ボリューム構成を追加するには、[ボリュームの追加] を選択し、以下のステップを実行します。

    a. [Volume name] (ボリューム名) には、データボリュームの名前を入力します。データボリューム名は、コンテナマウントポイントを作成するときに使用されます。

    b. [ボリューム設定] では、ボリュームをタスク定義の作成時に設定するのか、デプロイ時に設定するのかを選択します。

    > **Note**
    >
    > タスク定義の作成時に設定できるボリュームには、バインドマウント、Docker、Amazon EFS、Amazon FSx for Windows File Server などがあります。デプロイ時にタスクを実行するとき、またはサービスを作成または更新するときに設定できるボリュームには Amazon EBS が含まれます。

    c. [ボリュームタイプ]では、選択した設定タイプと互換性のあるボリュームタイプを選択し、ボリュームタイプを設定します。

ボリュームタイプ	Steps	
バインドマウント	a. [Add mount point] (マウントポイントの追加) を選択し、次の設定を行います。 • [Container] (コンテナ) には、マウントポイントのコンテナを選択します。 • [Source volume] (ソースボリューム) を使用	

ボリュームタイプ	Steps	
	する場合、コンテナにマウントするデータボリュームを選択します。 • [Container path] (コンテナパス) に、ボリュームをマウントするコンテナのパスを指定します。 • [読み取り専用] では、コンテナがボリュームに対して読み取り専用アクセスを持つかどうかを選択します。 b. マウントポイントを追加するには、[Add mount point] (マウントポイントの追加)。	

ボリュームタイプ	Steps	
EFS	a. [File system ID] (ファイルシステム ID) で、Amazon EFS のファイルシステム ID を選択します。 b. (オプション) [Root directory] (ルートディレクトリ) には、ホスト内にルートディレクトリとしてマウントする Amazon EFS ファイルシステム内のディレクトリを入力します。このパラメータを省略すると、Amazon EFS ボリュームのルートが使用されます。 EFS アクセスポイントを使用する予定の場合は、このフィールドを空白のままにします。 c. (オプション) [Access point] (アクセスポイント) で、使用するアクセスポイント ID を選択します。 d. (オプション) Amazon EFS ファイルシステムと Amazon ECS ホスト間のデータを暗号化する	

ボリュームタイプ	Steps	
	場合、または、ボリュームをマウントするときにタスク実行ロールを使用する場合は、[Advanced configurations] (詳細設定) を選択し、以下を設定します。 • Amazon EFS ファイルシステムと Amazon ECS ホスト間のデータを暗号化するには、[Transit encryption] (トランジットの暗号化) を選択し、[Port] (ポート) に、Amazon ECS ホストと Amazon EFS サーバーの間で暗号化されたデータを送信する際に使用するポートを入力します。転送中の暗号化ポートを指定しないと、Amazon EFS マウントヘルパーが使用するポート選択方式が使用されます。詳細については、[Amazon Elastic File System User Guide] (Amazon Elastic File System ユーザーガイド) の[EFS Mount Helper] (EFS マウント	

ボリュームタイプ	Steps	
	ヘルパー) を参照してください。 • Amazon EFS ファイルシステムをマウントするときにタスク定義で定義された Amazon ECS タスク IAM ロールを使用するには、[IAM 認可] を選択します。 e. [Add mount point] (マウントポイントの追加) を選択し、次の設定を行います。 • [Container] (コンテナ) には、マウントポイントのコンテナを選択します。 • [Source volume] (ソースボリューム) を使用する場合、コンテナにマウントするデータボリュームを選択します。 • [Container path] (コンテナパス) に、ボリュームをマウントするコンテナのパスを指定します。 •	

ボリュームタイプ	Steps	
	[読み取り専用] では、コンテナがボリュームに対して読み取り専用アクセスを持つかどうかを選択します。 f. マウントポイントを追加するには、[Add mount point] (マウントポイントの追加)。	

ボリュームタイプ	Steps	
Docker	a. [ドライバー] に、Docker ボリュームの設定を入力します。Windows コンテナでは、[ローカル] ドライバーの使用のみがサポートされます。バインドマウントを使用するには、host を指定します。 b. [Scope] (スコープ) で、ボリュームのライフサイクルを選択します。 • タスクの開始時と停止時にライフサイクルが続くようにするには、[Task] (タスク) を選択します。 • タスクが停止した後もボリュームを維持するには、[Shared] (共有) を選択します。 c. [Add mount point] (マウントポイントの追加) を選択し、次の設定を行います。 • [Container] (コンテナ) には、マウントポイントのコンテナを選択します。 •	

ボリュームタイプ	Steps	
	[Source volume] (ソースボリューム) を使用する場合、コンテナにマウントするデータボリュームを選択します。 • [Container path] (コンテナパス) に、ボリュームをマウントするコンテナのパスを指定します。 • [読み取り専用] では、コンテナがボリュームに対して読み取り専用アクセスを持つかどうかを選択します。 d. マウントポイントを追加するには、[Add mount point] (マウントポイントの追加)。	

ボリュームタイプ	Steps	
FSx for Windows File Server	a. [ファイルシステム ID] で、FSx for Windows File Server システムの ID を選択します。 b. [ルートディレクトリ] で、ホスト内にルートディレクトリとしてマウントするための、FSx for Windows File Server ファイルシステム内のディレクトリを入力します。 c. [認証情報パラメータ] で、認証情報の保存方法を選択します。 • AWS Secrets Manager を使用するには、Secrets Manager シークレットの Amazon リソースネーム (ARN) を入力します。 • AWS Systems Manager を使用するには、Systems Manager パラメータの Amazon リソースネーム (ARN) を入力します。 d. [ドメイン] に、AWS Directory Service	

ボリュームタイプ	Steps	
	for Microsoft Active Directory (AWS Managed Microsoft AD) ディレクトリまたはセルフホスト型 EC2 Active Directory によってホストされる完全修飾ドメイン名を入力します。 e. [Add mount point] (マウントポイントの追加) を選択し、次の設定を行います。 • [Container] (コンテナ) には、マウントポイントのコンテナを選択します。 • [Source volume] (ソースボリューム) を使用する場合、コンテナにマウントするデータボリュームを選択します。 • [Container path] (コンテナパス) に、ボリュームをマウントするコンテナのパスを指定します。 • [読み取り専用] では、コンテナがボリュームに対して読み取り専用	

ボリュームタイプ	Steps	
	アクセスを持つかどうかを選択します。 f. マウントポイントを追加するには、[Add mount point] (マウントポイントの追加)。	

ボリュームタイプ	Steps	
Amazon EBS	a. [Add mount point] (マウントポイントの追加) を選択し、次の設定を行います。 • [Container] (コンテナ) には、マウントポイントのコンテナを選択します。 • [Source volume] (ソースボリューム) を使用する場合、コンテナにマウントするデータボリュームを選択します。 • [Container path] (コンテナパス) に、ボリュームをマウントするコンテナのパスを指定します。 • [読み取り専用] では、コンテナがボリュームに対して読み取り専用アクセスを持つかどうかを選択します。 b. マウントポイントを追加するには、[Add mount point] (マウントポイントの追加)。	

ボリュームタイプ	Steps	

15. 別のコンテナからボリュームを追加するには、[以下からボリュームを追加:] を選択し、次のように構成します。

    - [コンテナ] で、対象のコンテナを選択します。
    - [ソース] で、マウントするボリュームが含まれるコンテナを選択します。
    - [読み取り専用] では、コンテナがボリュームに対して読み取り専用アクセスを持つかどうかを選択します。

16. (オプション) AWS Distro for OpenTelemetry 統合を使用してアプリケーショントレースとメトリクス収集設定を構成するには、[モニタリング] を展開し、[メトリクス収集を使用] を選択してタスクのメトリクスを収集し、Amazon CloudWatch または Amazon Managed Service for Prometheus に送信します。このオプションを選択すると、Amazon ECS はアプリケーションメトリクスの送信向けに事前設定された AWS Distro for OpenTelemetry コンテナサイドカーを作成します。詳細については、「アプリケーションメトリクスを使用して Amazon ECS アプリケーションのパフォーマンスを相関させる」を参照してください。

    a. [Amazon CloudWatch] を選択した場合、カスタムアプリケーションメトリクスはカスタムメトリクスとして CloudWatch にルーティングされます。詳細については、「アプリケーションメトリクスを Amazon CloudWatch にエクスポートする」を参照してください。

    > **Important**
    >
    > Amazon CloudWatch にアプリケーションメトリクスをエクスポートする場合、タスク定義には、必要なアクセス権限を持つタスク IAM ロールが必要です。詳しくは、「OpenTelemetry 用 AWS Distro と Amazon CloudWatch の統合に必要な IAM 許可」を参照してください。

    b. [Amazon Managed Service for Prometheus (Prometheus libraries instrumentation)] (Prometheus 向け Amazon マネージドサービス (Prometheus ライブラリ計測)) を選択した場合、タスクレベルの CPU、メモリ、ネットワーク、ストレージのメトリクスとカスタムアプリケーションメトリクスが Amazon Managed Service for Prometheus にルーティングされます。[Workspace のリモート書き込み用エンドポイント] には、Prometheus ワークスペースのリモート書き込みエンドポイント URL を入力し

ます。[スクレイピングターゲット] には、AWS Distro for OpenTelemetry コレクターがメトリクスデータをスクレイプするために使用できるホストとポートを入力します。詳細については、「アプリケーションメトリクスを Amazon Managed Service for Prometheus にエクスポートする」を参照してください。

> **Important**
>
> Amazon Managed Service for Prometheus にアプリケーションメトリクスをエクスポートする場合、タスク定義には、必要なアクセス権限を持つタスク IAM ロールが必要です。詳細については、「OpenTelemetry 用 AWS Distro と Amazon Managed Service for Prometheus の統合に必要な IAM 許可」を参照してください。

c. [Amazon Managed Service for Prometheus (OpenTelemetry 計測)] を選択した場合、タスクレベルの CPU、メモリ、ネットワーク、ストレージのメトリクスとカスタムアプリケーションメトリクスが Amazon Managed Service for Prometheus にルーティングされます。[Workspace のリモート書き込み用エンドポイント] には、Prometheus ワークスペースのリモート書き込みエンドポイント URL を入力します。詳細については、「アプリケーションメトリクスを Amazon Managed Service for Prometheus にエクスポートする」を参照してください。

> **Important**
>
> Amazon Managed Service for Prometheus にアプリケーションメトリクスをエクスポートする場合、タスク定義には、必要なアクセス権限を持つタスク IAM ロールが必要です。詳細については、「OpenTelemetry 用 AWS Distro と Amazon Managed Service for Prometheus の統合に必要な IAM 許可」を参照してください。

17. (オプション) [Tags] (タグ) を展開します。タグをキーバリューペアとしてタスク定義に追加します。

    - [タグを追加] [Add tag] (タグを追加) を選択し、以下を実行します。
      - [キー] にはキー名を入力します。
      - [値] にキー値を入力します。
    - [タグの削除] タグの横にある [タグの削除] を選択します。

18. [作成] を選択してタスク定義を登録します。

Amazon ECS console JSON editor

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで、タスク定義 を選択します。
3. [新しいタスク定義の作成] メニューで、[JSON で新しいタスク定義を作成する] を選択します。
4. JSON エディタボックスで、JSON ファイルを編集し、

   JSON は、the section called “JSON 検証” で指定された検証チェックに合格する必要があります。
5. [作成] を選択します。

Amazon Q Developer を使用して Amazon ECS コンソールでタスク定義のレコメンデーションを生成する

Amazon ECS コンソールで JSON エディタを使用してタスク定義を作成する場合、Amazon Q Developer を使用することで、タスク定義に AI 生成のコード提案を取得できます。

インラインチャット機能を使用して、会話インターフェイスを使用してタスク定義 JSON を生成、説明、またはリファクタリングするように Amazon Q Developer に依頼できます。生成された提案は、タスク定義の任意の時点で挿入し、提案された変更を承認または拒否できます。Amazon ECS では、Amazon Q Developer を利用するために既存のインライン提案機能も強化されました。

JSON エディタを使用してタスク定義を作成する場合、Amazon Q Developer にタスク定義をより迅速に作成するための推奨事項を提供してもらうことができます。プロパティベースのインライン提案を作成することも、Amazon Q Developer の提案を使用してサンプルコードのブロック全体をオートコンプリートすることもできます。

この機能は、Amazon Q Developer がサポートされているリージョンで使用できます。詳細は「AWS Services by Regions」を参照してください。

前提条件

前提条件は次のとおりです。

- コンソールでタスク定義を作成するユーザーには、コンソールのアクセス許可に加えて、レコメンデーションのための `codewhisperer:GenerateRecommendations` のアクセス許可とインライ

ンチャットを使用するための q:SendMessage が必要になります。詳細については、「Amazon Q Developer を使用して、コンソールで推奨事項を提供するために必要なアクセス許可」を参照してください。

手順

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで、タスク定義 を選択します。
3. [新しいタスク定義の作成] メニューで、[JSON で新しいタスク定義を作成する] を選択します。

   [タスク定義の作成] ページを開きます。

   コンソールには、次のデフォルトテンプレートが用意されています。

   ```
   {
       "requiresCompatibilities": [
           "FARGATE"
       ],
       "family": "",
       "containerDefinitions": [
           {
               "name": "",
               "image": "",
               "essential": true
           }
       ],
       "volumes": [],
       "networkMode": "awsvpc",
       "memory": "3 GB",
       "cpu": "1 vCPU",
       "executionRoleArn": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole"
   }
   ```

4. Amazon Q インライン提案ポップアップで、[許可] を選択します。

   ポップアップを閉じると、歯車アイコンで Amazon Q を有効にできます。

5. JSON エディタボックスで、JSON ドキュメントを編集します。

   Amazon Q でパラメータを作成して入力するには、追加する内容を含むコメントを入力します。次の例では、コメントにより Amazon Q が太字の行を生成します。

```
{
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "family": "",
    "containerDefinitions": [
        {
            "name": "",
            "image": "",
            "essential": true
        },
        // add an nginx container using an image from Public ECR, with port 80
 open, and send logs to CloudWatch log group "myproxy"
        {
            "name": "nginx",
            "image": "public.ecr.aws/nginx/nginx:latest",
            "essential": true,
            "portMappings": [
                {
                    "containerPort": 80,
                    "hostPort": 80,
                    "protocol": "tcp"
                }
            ],
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-group": "myproxy",
                    "awslogs-region": "us-east-1",
                    "awslogs-stream-prefix": "nginx"
                }
            }
        }

    ],
    "volumes": [],
    "networkMode": "awsvpc",
    "memory": "3 GB",
    "cpu": "1 vCPU",
    "executionRoleArn": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole"
}
```

6. インラインチャット機能を使用するには、線を強調表示し、星のアイコンを選択します。

Amazon Q Developer チャットボックスが表示されます。

リクエストを入力します。

Amazon Q Developer は JSON を生成して更新します。

変更を受け入れるには、[すべて受け入れる] を選択します。

7. [作成] を選択します。

コンソールを使用した Amazon ECS タスク定義の更新

あるタスク定義リビジョンは、現在のタスク定義のコピーであり、新しいパラメータ値が既存のパラメータ値に置き換えられます。変更しないパラメータはすべて新しいリビジョンにあります。

タスク定義を更新するには、タスク定義リビジョンを作成します。サービスでタスク定義を使用する場合、更新されたタスク定義を使用するようサービスを更新する必要があります。

リビジョンを作成するときに、次のコンテナプロパティと環境プロパティを変更できます。

- コンテナイメージ URI
- ポートマッピング
- 環境変数
- インフラストラクチャ要件
- タスクサイズ
- コンテナサイズ
- タスクロール
- タスク実行ロール
- ボリュームとコンテナマウントポイント
- プライベートレジストリ

JSON エディタを使用すると、Amazon Q に推奨値を提供させることができます。詳細については、Amazon Q Developer を使用して Amazon ECS コンソールでタスク定義のレコメンデーションを生成するを参照してください。

JSON 検証

Amazon ECS コンソールの JSON エディタは、JSON ファイル内で以下を検証します。

- このJSON ファイルは有効です
- ファイルに外部キーは含まれていません
- ファイルに `familyName` パラメータが含まれています
- `containerDefinitions` の下に少なくとも 1 つのエントリがあります

手順

Amazon ECS console

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションバーから、タスク定義を含むリージョンを選択します。
3. ナビゲーションペインで、[Task Definitions] を選択します。
4. タスク定義を選択します。
5. タスク定義リビジョンを選択し、[新しいリビジョンを作成]、[新しいリビジョンを作成] を選択します。
6. [Create new task definition revision (タスク定義の新しいリビジョンの作成) ページで変更を加えます。例えば、既存のコンテナの定義 (コンテナイメージ、メモリ制限、ポートマッピングなど) を変更する場合は、コンテナを選択して変更を加えます。AWS Fargate、マネージドインスタンス、Amazon EC2 インスタンス のいずれかにタスク定義の互換性を更新できます。
7. 情報を確認し、[更新] を選択します。
8. サービスでタスク定義を使用する場合、更新されたタスク定義でサービスを更新します。詳細については、「Amazon ECS サービスを更新する」を参照してください。

Amazon ECS console JSON editor

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで、タスク定義 を選択します。
3. [Create new revision] (新しいリビジョンの作成)、[Create new revision with JSON] (JSON で新しいリビジョンを作成) の順に選択します。

4. JSON エディタボックスで、JSON ファイルを編集し、

   JSON は、the section called “JSON 検証” で指定された検証チェックに合格する必要があります。

5. [作成] を選択します。

新しいコンソールを使用した Amazon ECS タスク定義リビジョンの登録解除

そのタスク定義リビジョンを登録解除することで、タスクの実行やサービスの更新時、そのタスク定義リビジョンが ListTaskDefinition API コールにもコンソールにも表示されなくなります。

タスク定義リビジョンは登録解除されると、すぐに INACTIVE とマークされます。INACTIVE タスク定義リビジョンを参照する既存のタスクおよびサービスは、中断することなく引き続き実行されます。INACTIVE タスク定義のリビジョンを参照する既存のサービスは、希望するサービスの数を変更することでスケールアップまたはスケールダウンできます。

INACTIVE タスク定義のリビジョンを使用して、新しいタスクを実行したり、新しいサービスを作成したりすることはできません。また、INACTIVE タスク定義のリビジョンを参照するように既存のサービスを更新することはできません (登録解除後 10 分経過するまでは、これらの制限が有効にならないことがあります)。

> **Note**
>
> タスクファミリーのすべてのリビジョンを登録解除すると、タスク定義ファミリーが INACTIVE リストに移動されます。INACTIVE タスク定義の新しいリビジョンを追加すると、タスク定義ファミリーが ACTIVE リストに戻ります。
> 現時点では、INACTIVE タスク定義のリビジョンはアカウントで無期限に検出可能です。ただし、この動作は、今後変更される可能性があります。したがって、関連するタスクおよびサービスのライフサイクルを超えて持続する INACTIVE タスク定義のリビジョンに依存しないでください。

CloudFormation スタック

次の動作は、2023 年 1 月 12 日以前に新しい Amazon ECS コンソールで作成されたタスク定義に適用されます。

タスク定義の作成時、Amazon ECS コンソールは、名前が ECS-Console-V2-TaskDefinition- で始まる CloudFormation スタックを自動的に作成します。AWS CLI または AWS SDK を使用してタスク定義を登録解除した場合は、手動でタスク定義スタックを削除する必要があります。詳細については、「CloudFormation ユーザーガイド」の「スタックの削除」を参照してください。

2023 年 1 月 12 日以降に作成されたタスク定義では、CloudFormation スタックの自動的な作成は行われません。

手順

新しいタスク定義を登録解除するには (Amazon ECS コンソール)

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションバーから、タスク定義を含むリージョンを選択します。
3. ナビゲーションペインで、[Task Definitions] を選択します。
4. [Task Definitions] (タスク定義) ページで、登録解除する 1 つ以上のタスク定義リビジョンを含むタスク定義ファミリーを選択します。
5. [タスク定義名] ページで、削除するリビジョンを選択し、次に [アクション]、[登録解除] の順に選択します。
6. [Deregister] (登録解除) ウィンドウの情報を確認し、[Deregister] (登録解除) を選択して終了します。

コンソールを使用した Amazon ECS タスク定義リビジョンの削除

Amazon ECS で特定のタスク定義リビジョンが不要になったら、そのタスク定義リビジョンを削除することができます。

削除されたタスク定義リビジョンは、その後すぐに INACTIVE から DELETE_IN_PROGRESS に遷移します。状態が DELETE_IN_PROGRESS のタスク定義リビジョンを参照している既存のタスクおよびサービスは、中断することなく引き続き実行されます。

状態が DELETE_IN_PROGRESS のタスク定義リビジョンは、新しいタスクの実行や、新しいサービスの作成のために使用することはできません。また、DELETE_IN_PROGRESS 状態にあるタスク定義リビジョンを参照するように、既存のサービスを更新することもできません。

すべての INACTIVE タスク定義リビジョンを削除すると、タスク定義名はコンソールで表示されず、API でも返されません。タスク定義リビジョンが DELETE_IN_PROGRESS 状態にある場合、タ

スク定義名はコンソールに表示され、API で返されます。タスク定義名は Amazon ECS によって保持され、次回その名前を使用してタスク定義を作成するときにリビジョンがインクリメントされます。

削除をブロックできる Amazon ECS リソース

タスク定義リビジョンに依存する Amazon ECS リソースがある場合、タスク定義の削除リクエストは完了しません。次のリソースが原因で、タスク定義が削除されない場合があります。

- Amazon ECS スタンドアロンタスク – タスクを正常に動作させるには、タスク定義が必要です。
- Amazon ECS サービスタスク - タスクを正常に動作させるには、タスク定義が必要です。
- Amazon ECS サービスのデプロイとタスクセット - Amazon ECS のデプロイまたはタスクセットのスケーリングイベントが開始される場合は、タスク定義が必要です。

タスク定義が `DELETE_IN_PROGRESS` の状態のままである場合は、コンソールまたは AWS CLI を使用して、タスク定義の削除をブロックしているリソースを特定し、停止できます。

ブロックされたリソースが削除された後のタスク定義の削除

タスク定義の削除をブロックするリソースを削除すると、次のルールが適用されます。

- Amazon ECS タスク - タスク定義の削除は、タスクが停止されてから完了するまでに最大 1 時間かかる場合があります。
- Amazon ECS サービスのデプロイとタスクセット - タスク定義の削除は、デプロイまたはタスクセットが削除されてから完了するまでに最大 24 時間かかる場合があります。

手順

タスク定義を登録解除するには (Amazon ECS コンソール)

タスク定義リビジョンは、削除する前に登録解除する必要があります。詳細については、「the section called “新しいコンソールを使用したタスク定義リビジョンの登録解除”」を参照してください。

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションバーから、タスク定義を含むリージョンを選択します。
3. ナビゲーションペインで、[Task Definitions] を選択します。

4. [タスク定義] ページで、登録解除する 1 つ以上のタスク定義リビジョンを含むタスク定義ファミリーを選択します。
5. [タスク定義名] ページで、削除するリビジョンを選択し、次に [アクション]、[削除] の順に選択します。

   [削除] が使用できない場合は、タスク定義の登録を解除する必要があります。
6. [削除] 確認ボックス内の情報を確認し、[削除する] を選択して終了します。

Amazon ECS タスク定義のユースケース

さまざまな AWS のサービスや機能のタスク定義を作成する方法を説明します。

ワークロードによっては、設定を要するいくつかのタスク定義パラメータがあります。また、EC2 では、そのワークロード用に設計された特定のインスタンスを選択する必要があります。

トピック

- GPU ワークロード向けの Amazon ECS タスク定義
- 動画トランスコーディングワークロードでの Amazon ECS タスク定義
- AWS Neuron 機械学習ワークロードでの Amazon ECS タスク定義
- 深層学習インスタンスでの Amazon ECS タスク定義
- 64 ビット ARM ワークロードでの Amazon ECS タスク定義
- Amazon ECS ログを CloudWatch に送信する
- Amazon ECS ログを AWS サービスまたは AWS Partner に送信する
- Amazon ECS での AWS 以外のコンテナイメージの使用
- コンテナ再起動ポリシーを使用して Amazon ECS タスク内の個々のコンテナを再起動する
- Amazon ECS コンテナに機密データを渡す

GPU ワークロード向けの Amazon ECS タスク定義

GPU 対応コンテナインスタンスを使用してクラスターを作成することで、GPU を使用するワークロードが Amazon ECS でサポートされます。p2、p3、p4d、p5、g3、g4、g5、g6、g6e、および g6f インスタンスタイプを使用する Amazon EC2 GPU ベースのコンテナインスタンスにより、NVIDIA GPU にアクセスできます。詳細については、「Amazon EC2 インスタンスタイプガイド」の「Linux Accelerated Computing Instances」を参照してください。

Amazon ECS には、事前設定された NVIDIA カーネルドライバーと Docker GPU ランタイムが付属する、GPU 最適化 AMI が用意されています。詳細については、「Amazon ECS に最適化された Linux AMI」を参照してください。

コンテナレベルのタスク配置のために、タスク定義でいくつかの GPU を指定できます。Amazon ECS は、使用が可能で GPU をサポートしているコンテナインスタンスをスケジュールし、物理的 GPU を適切なコンテナに固定して最適なパフォーマンスを実現します。

以下の Amazon EC2 GPU ベースのインスタンスタイプがサポートされています。詳細については、「Amazon EC2 P2 インスタンス」、「Amazon EC2 P3 インスタンス」、「Amazon EC2 P4d インスタンス」、「Amazon EC2 P5 インスタンス」、「Amazon EC2 G3 インスタンス」、「Amazon EC2 G4 インスタンス」、「Amazon EC2 G5 インスタンス」、「Amazon EC2 G6 インスタンス」、「Amazon EC2 G6e インスタンス」、および「Amazon EC2 G6f インスタンス」を参照してください。

インスタンスタイプ	GPU	GPU メモリ (GiB)	vCPU	メモリ (GiB)
p3.2xlarge	1	16	8	61
p3.8xlarge	4	64	32	244
p3.16xlarge	8	128	64	488
p3dn.24xlarge	8	256	96	768
p4d.24xlarge	8	320	96	1152
p5.48xlarge	8	640	192	2048
g3s.xlarge	1	8	4	30.5
g3.4xlarge	1	8	16	122
g3.8xlarge	2	16	32	244
g3.16xlarge	4	32	64	488
g4dn.xlarge	1	16	4	16
g4dn.2xlarge	1	16	8	32

インスタンスタイプ	GPU	GPU メモリ (GiB)	vCPU	メモリ (GiB)
g4dn.4xlarge	1	16	16	64
g4dn.8xlarge	1	16	32	128
g4dn.12xlarge	4	64	48	192
g4dn.16xlarge	1	16	64	256
g5.xlarge	1	24	4	16
g5.2xlarge	1	24	8	32
g5.4xlarge	1	24	16	64
g5.8xlarge	1	24	32	128
g5.16xlarge	1	24	64	256
g5.12xlarge	4	96	48	192
g5.24xlarge	4	96	96	384
g5.48xlarge	8	192	192	768
g6.xlarge	1	24	4	16
g6.2xlarge	1	24	8	32
g6.4xlarge	1	24	16	64
g6.8xlarge	1	24	32	128
g6.16.xlarge	1	24	64	256
g6.12xlarge	4	96	48	192
g6.24xlarge	4	96	96	384
g6.48xlarge	8	192	192	768

インスタンスタイプ	GPU	GPU メモリ (GiB)	vCPU	メモリ (GiB)
g6.metal	8	192	192	768
gr6.4xlarge	1	24	16	128
g6e.xlarge	1	48	4	32
g6e.2xlarge	1	48	8	64
g6e.4xlarge	1	48	16	128
g6e.8xlarge	1	48	32	256
g6e16.xlarge	1	48	64	512
g6e12.xlarge	4	192	48	384
g6e24.xlarge	4	192	96	768
g6e48.xlarge	8	384	192	1536
gr6.8xlarge	1	24	32	256
g6f.large	1/8 (0.125)	3	2	8
g6f.xlarge	1/8 (0.125)	3	4	16
g6f.2xlarge	1/4 (0.25)	6	8	32
g6f.4xlarge	1/2 (0.5)	12	16	64
gr6f.4xlarge	1/2 (0.5)	12	16	128

AWS Systems Manager パラメータストア API をクエリすることで、Amazon ECS に最適化された AMI の Amazon マシンイメージ (AMI) ID を取得できます。このパラメータを使用することで、Amazon ECS 最適化 AMI ID を手動で検索する必要がなくなります。Systems Manager Parameter Store API の詳細については、GetParameter を参照してください。使用するユーザーに

は、Amazon ECS 最適化 AMI メタデータを取得するための `ssm:GetParameter` IAM 許可が必要です。

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/gpu/
recommended --region us-east-1
```

Amazon ECS マネージドインスタンスで GPU を使用する

Amazon ECS マネージドインスタンスは、次の Amazon EC2 インスタンスタイプを通じて、機械学習、高性能コンピューティング、ビデオ処理などのワークロードに対して GPU アクセラレーションコンピューティングをサポートしています。Amazon ECS マネージドインスタンスが対応するインスタンスタイプについて、詳しくは「Amazon ECS マネージドインスタンスのインスタンスタイプ」を参照してください。

以下は、Amazon ECS マネージドインスタンスでサポートされている GPU ベースのインスタンスタイプのサブセットです。

- g4dn: NVIDIA T4 GPUs を利用し、機械学習の推論、コンピュータビジョン、グラフィックを多用するアプリケーションに適しています。
- g5: NVIDIA A10G GPUs を利用し、グラフィックスを多用するアプリケーションや機械学習ワークロードでより高いパフォーマンスを提供します。
- p3: NVIDIA V100 GPUs を利用し、高性能コンピューティングと深層学習トレーニングに適しています。
- p4d: NVIDIA A100 GPUs を利用し、機械学習トレーニングとハイパフォーマンスコンピューティングを支援します。
- g6f: NVIDIA L4 GPUs (フラクショナル) を搭載し、ハードウェアでパーティション分割された GPU スライス (NVIDIA L4 GPU の 1/8、1/4、または 1/2) に専用の GPU メモリとコンピューティングを提供します。

Amazon ECS マネージドインスタンスで GPU 対応インスタンスタイプを使用すると、NVIDIA ドライバーと CUDA ツールキットがインスタンスに事前にインストールされるため、GPU 対応ワークロードの実行が容易になります。

GPU 対応インスタンスの選択

Amazon ECS マネージドインスタンスワークロードの GPU 対応インスタンスタイプを選択するには、キャパシティプロバイダーの起動テンプレートで `instanceRequirements` オブジェクトを使用します。次のスニペットは、GPU 対応インスタンスの選択に使用できる属性を示しています。

```
{
  "instanceRequirements": {
    "acceleratorTypes": "gpu",
    "acceleratorCount": 1,
    "acceleratorManufacturers": ["nvidia"]
  }
}
```

次のスニペットは、起動テンプレートで GPU 対応インスタンスタイプを指定するために使用できる属性を示しています。

```
{
  "instanceRequirements": {
    "allowedInstanceTypes": ["g4dn.xlarge", "p4de.24xlarge"]
  }
}
```

GPU 対応コンテナイメージ

コンテナで GPUs を使用するには、必要な GPU ライブラリとツールを含むコンテナイメージを使用する必要があります。NVIDIA には、GPU ワークロードのベースとして使用できるいくつかの構築済みコンテナイメージが用意されています。

- `nvidia:cuda`: GPU コンピューティング用の CUDA ツールキットを使用したベースイメージ。
- `tensorflow/tensorflow:latest-gpu`: GPU 対応の TensorFlow。
- `pytorch/pytorch:latest-cuda`: GPU 対応の PyTorch。

GPU を使用した Amazon ECS マネージドインスタンスのタスク定義サンプルは「Amazon ECS タスク定義での GPU の指定」を参照してください。

考慮事項

> **Note**
>
> g2 インスタンスファミリータイプのサポートは廃止されました。
> p2 インスタンスファミリータイプは、バージョン 20230912 より前の Amazon ECS GPU 最適化 AMI でのみサポートされています。引き続き p2 インスタンスを使用する必要がある場合は、P2 インスタンスが必要な場合の対処方法 を参照してください。

これら両方のインスタンスファミリータイプで NVIDIA/CUDA ドライバーをインプレース更新すると、GPU ワークロードに障害が発生する可能性があります。

Amazon ECS で GPU の使用を開始する前に、以下について検討することをお勧めします。

- クラスターには、GPU 対応コンテナインスタンスと GPU 非対応コンテナインスタンスを混在させることができます。
- GPU ワークロードは外部インスタンスでの実行が可能です。外部インスタンスをクラスターに登録するときは、`--enable-gpu` フラグがインストールスクリプトに含まれていることを確認してください。詳細については、「Amazon ECS クラスターに外部インスタンスを登録する」を参照してください。
- エージェント設定ファイルで `ECS_ENABLE_GPU_SUPPORT` を `true` に設定する必要があります。詳細については、「the section called “コンテナエージェントの設定”」を参照してください。
- タスクの実行時またはサービスの作成時に、インスタンスタイプ属性を使用してタスク配置制約を設定することで、タスクが起動されるコンテナインスタンスを指定できます。これにより、リソースをより効果的に使用できます。詳細については、「Amazon ECS がタスクをコンテナインスタンスに配置する方法」を参照してください。

  以下の例では、デフォルトクラスター内の `g4dn.xlarge` コンテナインスタンスでタスクを起動しています。

  ```
  aws ecs run-task --cluster default --task-definition ecs-gpu-task-def \
       --placement-constraints type=memberOf,expression="attribute:ecs.instance-type ==
    g4dn.xlarge" --region us-east-2
  ```

- Amazon ECS は、コンテナ定義で指定された GPU リソース要件が適用されるコンテナごとに、コンテナランタイムとして NVIDIA のコンテナランタイムが使用されるように設定します。
- NVIDIA コンテナランタイムが適切に機能するためには、コンテナにいくつかの環境変数を設定する必要があります。これらの環境変数のリストについては、「Docker を使用した特殊設定」を参照してください。Amazon ECS は、`NVIDIA_VISIBLE_DEVICES`環境変数の値を Amazon ECS がコンテナに割り当てる GPU デバイス ID のリストに設定します。Amazon ECS は、これら以外の必須環境変数の設定は行いません。そのため、これらの必須変数がコンテナイメージで設定されていること、あるいはコンテナ定義内で設定されていることを確認する必要があります。
- p5 インスタンスタイプファミリーは、バージョン `20230929` 以降の Amazon ECS GPU 最適化 AMI でサポートされています。

- g4 インスタンスタイプファミリーは、バージョン 20230913 以降の Amazon ECS GPU 最適化 AMI でサポートされています。詳細については、「Amazon ECS に最適化された Linux AMI」を参照してください。Amazon ECS コンソールのクラスター作成ワークフローではサポートされていません。これらのインスタンスタイプを使用するには、Amazon EC2 コンソール、AWS CLI、または API を使用して、手動でインスタンスをクラスターに登録する必要があります。
- p4d.24xlarge インスタンスタイプは、CUDA 11 以降でのみ動作します。
- Amazon ECS GPU 最適化 AMI は IPv6 が有効になっているため、yum を使用するとき問題が生じます。これは、以下のコマンドで IPv4 を使用するように yum を構成することで解決できます。

```
echo "ip_resolve=4" >> /etc/yum.conf
```

- NVIDIA/CUDA ベースのイメージを使用しないコンテナイメージをビルドする場合は、NVIDIA_DRIVER_CAPABILITIES コンテナランタイム変数に以下のいずれかの値を設定します。
  - utility,compute
  - all

  変数の設定方法については、NVIDIA の Web サイトの「NVIDIA コンテナランタイムの制御」を参照してください。
- Windows コンテナでは、GPU はサポートされません。

Amazon ECS 向け GPU コンテナインスタンスの起動

Amazon EC2 上で Amazon ECS の GPU インスタンスを使用するには、起動テンプレートとユーザーデータファイルを作成し、インスタンスを起動する必要があります。

これにより、GPU 用に設定されたタスク定義を使用するタスクを実行できます。

起動テンプレートを使用する

起動テンプレートを作成します。

- Amazon ECS に最適化された AMI 用の GPU AMI ID を使用する起動テンプレートを作成してください。起動テンプレートの作成方法について、詳しくは「Amazon EC2 ユーザーガイド」の「Create a new launch template using parameters you define」を参照してください。

[Amazon マシンイメージ] には、前のステップの AMI ID を使用します。Systems Manager パラメータで AMI ID を指定する方法については、「Amazon EC2 ユーザーガイド」の「Specify a Systems Manager parameter in a launch template」を参照してください。

起動テンプレートの [ユーザーデータ] に以下を追加します。*cluster-name* をクラスターの名前に置き換えます。

```
#!/bin/bash
echo ECS_CLUSTER=cluster-name >> /etc/ecs/ecs.config;
echo ECS_ENABLE_GPU_SUPPORT=true >> /etc/ecs/ecs.config
```

AWS CLI の使用

AWS CLI を使用して、コンテナインスタンスを起動できます。

1. userdata.toml というファイルを作成します。このファイルは、インスタンスのユーザーデータに使用されます。*cluster-name* をクラスターの名前に置き換えます。

```
#!/bin/bash
echo ECS_CLUSTER=cluster-name >> /etc/ecs/ecs.config;
echo ECS_ENABLE_GPU_SUPPORT=true >> /etc/ecs/ecs.config
```

2. 次のコマンドを実行して、GPU AMI ID を取得します。これは次のステップで使用します。

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/gpu/
recommended --region us-east-1
```

3. 次のコマンドを実行して、GPU インスタンスを起動します。次のパラメータを必ず置き換えてください。

   - *subnet* を、インスタンスを起動するプライベートまたはパブリックサブネットの ID に置き換えます。
   - *gpu_ami* を、前のステップの AMI ID に置き換えます。
   - *t3.large* を、使用するインスタンスタイプに置き換えます。
   - *region* を、リージョンコードに置き換えます。

```
aws ec2 run-instances --key-name ecs-gpu-example \
```

```
    --subnet-id subnet \
    --image-id gpu_ami \
    --instance-type t3.large \
    --region region \
    --tag-specifications 'ResourceType=instance,Tags=[{Key=GPU,Value=example}]' \
    --user-data file://userdata.toml \
    --iam-instance-profile Name=ecsInstanceRole
```

4. 次のコマンドを実行して、コンテナインスタンスがクラスターに登録されていることを検証します。このコマンドを実行するときは、次のパラメータを必ず置き換えてください。

   - *cluster* を、自分のクラスター名に置き換えます。
   - *region* を、リージョンコードに置き換えます。

```
aws ecs list-container-instances --cluster cluster-name --region region
```

Amazon ECS タスク定義での GPU の指定

コンテナインスタンス上の GPU と Docker GPU ランタイムを利用するには、コンテナに必要な GPU の数をタスク定義内で必ず指定します。整数値、フラクショナル GPU の小数値、または ALL を指定できます。ALL を指定すると、コンテナインスタンス上のすべての GPU がコンテナに割り当てられます。0.25 などの小数値を指定すると、Amazon ECS はリクエストされた GPU 容量を提供するフラクショナル GPU インスタンスにタスクを配置します。GPU をサポートするコンテナが配置されると、Amazon ECS コンテナエージェントは必要な数の物理 GPU を適切なコンテナに固定します。タスク内でコンテナ用に予約されている、すべての GPU の数は、タスクが起動されたコンテナインスタンスで使用できる GPU の数を超えることはできません。詳細については、「コンソールを使用した Amazon ECS タスク定義の作成」を参照してください。

> **Important**
>
> GPU 要件をタスク定義で指定していない場合、タスクではデフォルトの Docker ランタイムが使用されます。

タスク定義での GPU 要件の JSON 形式は以下のとおりです。

```
{
  "containerDefinitions": [
```

```
     {
        ...
        "resourceRequirements" : [
            {
               "type" : "GPU",
               "value" : "2"
            }
        ],
     },
...
}
```

数値の代わりに値として ALL を指定して、コンテナインスタンス上のすべての GPU をコンテナに割り当てることもできます。

以下の例では、GPU 要件を指定する Docker コンテナの構文を示しています。このコンテナは 2 つの GPU を使用しており、nvidia-smi ユーティリティを実行した後に終了します。

```
{
  "containerDefinitions": [
    {
      "memory": 80,
      "essential": true,
      "name": "gpu",
      "image": "nvidia/cuda:11.0.3-base",
      "resourceRequirements": [
         {
           "type":"GPU",
           "value": "2"
         }
      ],
      "command": [
        "sh",
        "-c",
        "nvidia-smi"
      ],
      "cpu": 100
    }
  ],
  "family": "example-ecs-gpu"
}
```

次のタスク定義の例は、使用可能な GPU 数を出力する TensorFlow コンテナを示しています。タスクは Amazon ECS マネージドインスタンスで実行され、1 つの GPU が必要で、g4dn.xlarge インスタンスを使用します。

```
{
  "family": "tensorflow-gpu",
  "networkMode": "awsvpc",
  "executionRoleArn": "arn:aws:iam::account-id:role/ecsTaskExecutionRole",
  "containerDefinitions": [
    {
      "name": "tensorflow",
      "image": "tensorflow/tensorflow:latest-gpu",
      "essential": true,
      "command": [
        "python",
        "-c",
        "import tensorflow as tf; print('Num GPUs Available: ',
 len(tf.config.list_physical_devices('GPU')))"
      ],
      "resourceRequirements": [
        {
          "type": "GPU",
          "value": "1"
        }
      ],
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "/ecs/tensorflow-gpu",
          "awslogs-region": "region",
          "awslogs-stream-prefix": "ecs"
        }
      }
    }
  ],
  "requiresCompatibilities": [
    "MANAGED_INSTANCES"
  ],
  "cpu": "4096",
  "memory": "8192",
}
```

フラクショナル GPU の指定

Amazon ECS は、Amazon EC2 G6f インスタンスでのフラクショナル GPU スケジューリングをサポートしています。G6f インスタンスは、ハードウェアパーティション分割された NVIDIA L4 GPU の一部を提供します。各インスタンスは、専用の GPU メモリとコンピューティングを備えた固定 GPU スライス (1/8、1/4、または 1/2) を公開します。小数 GPU を使用するには、コンテナ定義の `resourceRequirements` に 10 進値を指定します。

```
"resourceRequirements": [
  {
    "type": "GPU",
    "value": "0.25"
  }
]
```

サポートされている小数値とそれに対応する G6f インスタンスタイプは次のとおりです。

GPU 値	GPU 分数	GPU メモリ	インスタンスのタイプ
`0.125`	1/8 GPU	3 GB	g6f.large、g6f.xlarge
`0.25`	1/4 GPU	6 GB	g6f.2xlarge
`0.5`	1/2 GPU	12 GB	g6f.4xlarge、gr6f.4xlarge

次のタスク定義の例では、NVIDIA L4 GPU (6 GB GPU メモリ) の 1/4 を使用する小数 GPU で軽量推論コンテナを実行します。

```
{
  "family": "fractional-gpu-inference",
  "networkMode": "awsvpc",
  "executionRoleArn": "arn:aws:iam::account-id:role/ecsTaskExecutionRole",
  "containerDefinitions": [
    {
      "name": "inference",
      "image": "nvidia/cuda:12.0.0-base-ubuntu22.04",
      "essential": true,
```

```
      "command": ["nvidia-smi"],
      "resourceRequirements": [
        {
          "type": "GPU",
          "value": "0.25"
        }
      ],
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "/ecs/fractional-gpu-inference",
          "awslogs-region": "region",
          "awslogs-stream-prefix": "ecs"
        }
      }
    }
  ],
  "requiresCompatibilities": [
    "MANAGED_INSTANCES"
  ],
  "cpu": "2048",
  "memory": "4096"
}
```

小数 GPU に関する考慮事項

- タスクごとに 1 つのコンテナのみが、小数 GPU 値をリクエストできます。コンテナが小数 GPU を指定する場合、そのタスク定義内の他のコンテナには GPU リソース要件を含めることはできません。
- 整数値 (1、2 など) は完全な GPU リクエストとして扱われ、完全な GPU 容量を持つインスタンスにのみ配置されます。小数値 (`0.125`、`0.25`、`0.5`) は小数 GPU リクエストとして扱われ、対応する小数 GPU パーティションを持つインスタンスに配置されます。
- 同じキャパシティープロバイダーには、小数 GPU インスタンス (G6f) と完全な GPU インスタンス (g5 や g6 など) の両方を含めることができます。Amazon ECS は、宣言された GPU 値に基づいてタスクを正しいインスタンスタイプにルーティングします。GPU ファミリーごとに個別のキャパシティープロバイダーは必要ありません。
- 小数 GPU スケジューリングは、Amazon ECS マネージドインスタンスと Amazon EC2 上の Amazon ECS で使用できます。Fargate と Amazon ECS Anywhere は、小数 GPU スケジューリングをサポートしていません。
- 完全な GPU ワークロードでは、引き続き `"value": "1"` のように整数値を使用します。

GPU の共有

GPU を共有する場合は、以下を設定する必要があります。

1. タスク定義から GPU リソース要件を削除して、Amazon ECS が共有する必要がある GPU を予約しないようにします。
2. GPU を共有する場合は、次のユーザーデータをインスタンスに追加します。これにより、NVIDIA がコンテナインスタンスのデフォルトの Docker コンテナランタイムになり、すべての Amazon ECS コンテナが GPU を使用できるようになります。詳細については、「Amazon EC2 ユーザーガイド」の「ユーザーデータ入力を使用して EC2 インスタンスを起動するときにコマンドを実行する」を参照してください。

   ```
   const userData = ec2.UserData.forLinux();
    userData.addCommands(
    'sudo rm /etc/sysconfig/docker',
    'echo DAEMON_MAXFILES=1048576 | sudo tee -a /etc/sysconfig/docker',
    'echo OPTIONS="--default-ulimit nofile=32768:65536 --default-runtime nvidia" |
    sudo tee -a /etc/sysconfig/docker',
    'echo DAEMON_PIDFILE_TIMEOUT=10 | sudo tee -a /etc/sysconfig/docker',
    'sudo systemctl restart docker',
   );
   ```

3. コンテナに NVIDIA_VISIBLE_DEVICES 環境変数を設定します。これを行うには、タスク定義内で環境変数を設定します。有効な値の詳細については、NVIDIA ドキュメントサイトの「GPU 列挙」を参照してください。

P2 インスタンスが必要な場合の対処方法

P2 インスタンスを使用する必要がある場合、次のいずれかのオプションを使用してインスタンスを使用し続けることができます。

どちらのオプションでも、インスタンスユーザーデータを変更する必要があります。詳細については、「Amazon EC2 ユーザーガイド」の「ユーザーデータ入力を使用して EC2 インスタンスを起動するときにコマンドを実行する」を参照してください。

サポートされている最新の GPU 最適化 AMI を使用する

GPU に最適化された AMI の 20230906 バージョンを使用して、インスタンスのユーザーデータに以下を追加できます。

cluster-name をクラスターの名前に置き換えます。

```
#!/bin/bash
echo "exclude=*nvidia* *cuda*" >> /etc/yum.conf
echo "ECS_CLUSTER=cluster-name" >> /etc/ecs/ecs.config
```

GPU に最適化された最新の AMI を使用し、ユーザーデータを更新する

以下をインスタンスのユーザーデータに追加します。これにより Nvidia 535/Cuda12.2 ドライバーがアンインストールされ、次に Nvidia 470/Cuda11.4 ドライバーがインストールされ、バージョンが修正されます。

```
#!/bin/bash
yum remove -y cuda-toolkit* nvidia-driver-latest-dkms*
tmpfile=$(mktemp)
cat >$tmpfile <<EOF
[amzn2-nvidia]
name=Amazon Linux 2 Nvidia repository
mirrorlist=\$awsproto://\$amazonlinux.\$awsregion.\$awsdomain/\$releasever/amzn2-
nvidia/latest/\$basearch/mirror.list
priority=20
gpgcheck=1
gpgkey=https://developer.download.nvidia.com/compute/cuda/repos/rhel7/
x86_64/7fa2af80.pub
enabled=1
exclude=libglvnd-*
EOF

mv $tmpfile /etc/yum.repos.d/amzn2-nvidia-tmp.repo
yum install -y system-release-nvidia cuda-toolkit-11-4 nvidia-driver-latest-
dkms-470.182.03
yum install -y libnvidia-container-1.4.0 libnvidia-container-tools-1.4.0 nvidia-
container-runtime-hook-1.4.0 docker-runtime-nvidia-1

echo "exclude=*nvidia* *cuda*" >> /etc/yum.conf
nvidia-smi
```

P2 と互換性のある GPU に最適化された独自の AMI を作成する

P2 インスタンスと互換性のある独自の Amazon ECS GPU 最適化 AMI を作成し、その AMI を使用して P2 インスタンスを起動できます。

1. 次のコマンドを実行して `amazon-ecs-ami repo` をクローンします。

```
git clone https://github.com/aws/amazon-ecs-ami
```

2. 必要な Amazon ECS エージェントとソースの Amazon Linux AMI バージョンを `release.auto.pkrvars.hcl` または `overrides.auto.pkrvars.hcl` に設定します。
3. P2 と互換性のあるプライベート EC2 AMI を構築するには、次のコマンドを実行します。

   リージョンをインスタンスリージョンがあるリージョンに置き換えます。

   ```
   REGION=region make al2keplergpu
   ```

4. 以下のインスタンスユーザーデータを含む AMI を使用して Amazon ECS クラスターに接続します。

   cluster-name をクラスターの名前に置き換えます。

   ```
   #!/bin/bash
   echo "ECS_CLUSTER=cluster-name" >> /etc/ecs/ecs.config
   ```

動画トランスコーディングワークロードでの Amazon ECS タスク定義

Amazon ECS で動画トランスコーディングワークロードを使用するには、Amazon EC2 VT1 インスタンスを登録します。これらのインスタンスを登録した後は、事前レンダリングされたライブ動画のトランスコーディングワークロードを、Amazon ECS タスクとして実行できます。Amazon EC2 VT1 インスタンスは、Xilinx U30 メディアトランスコーディングカードを使用して、事前レンダリングされたライブ動画のトランスコーディングワークロードを高速化します。

> **Note**
>
> Amazon ECS 以外のコンテナで動画トランスコーディングワークロードを実行する方法については、Xilinx のドキュメントを参照してください。

考慮事項

Amazon ECS で VT1 のデプロイを開始する前に、以下の点を考慮してください。

- クラスターには、VT1 コンテナインスタンスと VT1 以外のインスタンスを混在させることができます。

- アクセラレータ付きの AVC (H.264) および HEVC (H.265) コーデックを備えた、Xilinx U30 メディアトランスコーディングカードを使用する Linux アプリケーションが必要です。

  > **Important**
  >
  > 他のコーデックを使用するアプリケーションでは、VT1 インスタンスのパフォーマンスが強化されていない場合があります。

- U30 カードでは、トランスコーディングタスクを 1 つのみ実行できます。各カードには、2 つのデバイスが関連付けられています。VT1 インスタンスごとに、使用しているカードの数だけのトランスコーディングタスクを実行できます。
- サービスの作成時、またはスタンドアロンタスクの実行時に、インスタンスタイプ属性を使用してタスク配置制約を設定することができます。これにより、指定したコンテナインスタンスでタスクが起動されることを保証します。また、リソースを効果的に使用し、動画トランスコーディングワークロードのタスクが VT1 インスタンスに配置されるようにすることができます。詳細については、「Amazon ECS がタスクをコンテナインスタンスに配置する方法」を参照してください。

  次の例では、タスクが`default` クラスターの `vt1.3xlarge` インスタンス上で実行されます。

  ```
  aws ecs run-task \
       --cluster default \
       --task-definition vt1-3xlarge-xffmpeg-processor \
       --placement-constraints type=memberOf,expression="attribute:ecs.instance-type ==
   vt1.3xlarge"
  ```

- ホストコンテナインスタンス向けに用意された、特定の U30 カードを使用するようにコンテナを設定します。これを行うには、`linuxParameters` パラメータによりデバイスの詳細を指定します。詳細については、「タスク定義の要件」を参照してください。

VT1 AMI の使用

Amazon ECS コンテナインスタンス用 Amazon EC2 上で AMI を実行するには、2 つのオプションがあります。1 番目のオプションは、AWS Marketplace にある Xilinx 公式 AMI を使用することです。2 番目のオプションは、サンプルのリポジトリを利用して、独自の AMI を構築することです。

- Xilinx は、AWS Marketplace に AMI を提供しています。
- Amazon ECS は、動画トランスコーディングワークロード用の AMI を構築するために使用できる、サンプルのリポジトリを提供しています。この AMI には Xilinx U30 ドライバー備わりま

す。GitHub で、Packer スクリプトを含むリポジトリを入手できます。Packer の詳細については、「Packer documentation」(Packer ドキュメント) を参照してください。

タスク定義の要件

Amazon ECS で動画トランスコーディングコンテナを実行するには、アクセラレータ付きの H.264/AVC および H.265/HEVC コーデックを使用する動画トランスコーディングアプリケーションが、タスク定義に含まれている必要があります。Xilinx GitHub の手順に従うことで、コンテナイメージを構築できます。

タスク定義は、インスタンスタイプに固有である必要があります。これらのインスタンスタイプは、3xlarge、6xlarge、および 24xlarge です。コンテナでは、ホストコンテナインスタンス向けに用意された、特定の Xilinx U30 デバイスの使用を設定する必要があります。これは、`linuxParameters` パラメータを使用して設定します。次の表に、各インスタンスタイプに固有のカードとデバイス SoC の詳細を示します。

インスタンスタイプ	vCPU	RAM (GiB)	U30 アクセラレータカード	アドレス可能な XCU30 SoC デバイス	デバイスへのパス
vt1.3xlarge	12	24	1	2	`/dev/dri/renderD128 ,/dev/dri/renderD129`
vt1.6xlarge	24	48	2	4	`/dev/dri/renderD128 ,/dev/dri/renderD129 ,/dev/dri/renderD130 ,/dev/`

インスタンスタイプ	vCPU	RAM (GiB)	U30 アクセラレータカード	アドレス可能な XCU30 SoC デバイス	デバイスへのパス
					`dri/renderD131`

インスタンスタイプ	vCPU	RAM (GiB)	U30 アクセラレータカード	アドレス可能な XCU30 SoC デバイス	デバイスへのパス
vt1.24xlarge	96	182	8	16	/dev/dri/renderD128 ,/dev/dri/renderD129 ,/dev/dri/renderD130 ,/dev/dri/renderD131 ,/dev/dri/renderD132 ,/dev/dri/renderD133 ,/dev/dri/renderD134 ,/dev/dri/renderD135 ,/dev/dri/renderD136 ,/dev/dri/renderD137 ,/dev/dri/

インスタンスタイプ	vCPU	RAM (GiB)	U30 アクセラレータカード	アドレス可能な XCU30 SoC デバイス	デバイスへのパス
					renderD138 ,/dev/dri/renderD139 ,/dev/dri/renderD140 ,/dev/dri/renderD141 ,/dev/dri/renderD142 ,/dev/dri/renderD143

Important

タスク定義に EC2 インスタンスにないデバイスがリストされている場合、タスクの実行は失敗します。タスクが失敗した場合は、stoppedReason にエラーメッセージ `CannotStartContainerError: Error response from daemon: error gathering device information while adding custom device "/dev/dri/renderD130": no such file or directory` が表示されます。

Amazon ECS タスク定義での動画トランスコーディングの指定

次の例に、Amazon EC2 の Linux コンテナのタスク定義に使用される構文を示します。このタスク定義は、「Xilinx documentation」(Xilinx のドキュメント) で説明されている手順により構築されるコ

ンテナイメージ用です。この例を使用する場合は、image を独自のイメージで置き換え、/home/ec2-user ディレクトリにあるインスタンスにビデオファイルをコピーします。

vt1.3xlarge

1. vt1-3xlarge-ffmpeg-linux.json と言う名前で、以下の内容が記述されたテキストファイルを作成します。

```
{
    "family": "vt1-3xlarge-xffmpeg-processor",
    "requiresCompatibilities": ["EC2"],
    "placementConstraints": [
        {
            "type": "memberOf",
            "expression": "attribute:ecs.os-type == linux"
        },
        {
            "type": "memberOf",
            "expression": "attribute:ecs.instance-type == vt1.3xlarge"
        }
    ],
    "containerDefinitions": [
        {
            "entryPoint": [
                "/bin/bash",
                "-c"
            ],
            "command": ["/video/ecs_ffmpeg_wrapper.sh"],
            "linuxParameters": {
                "devices": [
                    {
                        "containerPath": "/dev/dri/renderD128",
                        "hostPath": "/dev/dri/renderD128",
                        "permissions": [
                            "read",
                            "write"
                        ]
                    },
                    {
                        "containerPath": "/dev/dri/renderD129",
                        "hostPath": "/dev/dri/renderD129",
                        "permissions": [
                            "read",
```

```
                            "write"
                        ]
                    }
                ]
            },
            "mountPoints": [
                {
                    "containerPath": "/video",
                    "sourceVolume": "video_file"
                }
            ],
            "cpu": 0,
            "memory": 12000,
            "image": "0123456789012.dkr.ecr.us-west-2.amazonaws.com/aws/xilinx-
xffmpeg",
            "essential": true,
            "name": "xilinix-xffmpeg"
        }
    ],
    "volumes": [
        {
            "name": "video_file",
            "host": {"sourcePath": "/home/ec2-user"}
        }
    ]
}
```

2. タスク定義を登録します。

```
aws ecs register-task-definition --family vt1-3xlarge-xffmpeg-processor --cli-
input-json file://vt1-3xlarge-xffmpeg-linux.json --region us-east-1
```

vt1.6xlarge

1. vt1-6xlarge-ffmpeg-linux.json と言う名前で、以下の内容が記述されたテキストファイルを作成します。

```
{
    "family": "vt1-6xlarge-xffmpeg-processor",
    "requiresCompatibilities": ["EC2"],
    "placementConstraints": [
        {
```

```
            "type": "memberOf",
            "expression": "attribute:ecs.os-type == linux"
        },
        {
            "type": "memberOf",
            "expression": "attribute:ecs.instance-type == vt1.6xlarge"
        }
    ],
    "containerDefinitions": [
        {
            "entryPoint": [
                "/bin/bash",
                "-c"
            ],
            "command": ["/video/ecs_ffmpeg_wrapper.sh"],
            "linuxParameters": {
                "devices": [
                    {
                        "containerPath": "/dev/dri/renderD128",
                        "hostPath": "/dev/dri/renderD128",
                        "permissions": [
                            "read",
                            "write"
                        ]
                    },
                    {
                        "containerPath": "/dev/dri/renderD129",
                        "hostPath": "/dev/dri/renderD129",
                        "permissions": [
                            "read",
                            "write"
                        ]
                    },
                    {
                        "containerPath": "/dev/dri/renderD130",
                        "hostPath": "/dev/dri/renderD130",
                        "permissions": [
                            "read",
                            "write"
                        ]
                    },
                    {
                        "containerPath": "/dev/dri/renderD131",
                        "hostPath": "/dev/dri/renderD131",
```

```
                        "permissions": [
                            "read",
                            "write"
                        ]
                    }
                ]
            },
            "mountPoints": [
                {
                    "containerPath": "/video",
                    "sourceVolume": "video_file"
                }
            ],
            "cpu": 0,
            "memory": 12000,
            "image": "0123456789012.dkr.ecr.us-west-2.amazonaws.com/aws/xilinx-
xffmpeg",
            "essential": true,
            "name": "xilinix-xffmpeg"
        }
    ],
    "volumes": [
        {
            "name": "video_file",
            "host": {"sourcePath": "/home/ec2-user"}
        }
    ]
}
```

2. タスク定義を登録します。

```
aws ecs register-task-definition --family vt1-6xlarge-xffmpeg-processor --cli-
input-json file://vt1-6xlarge-xffmpeg-linux.json --region us-east-1
```

vt1.24xlarge

1. vt1-24xlarge-ffmpeg-linux.json と言う名前で、以下の内容が記述されたテキストファイルを作成します。

```
{
    "family": "vt1-24xlarge-xffmpeg-processor",
    "requiresCompatibilities": ["EC2"],
```

```
    "placementConstraints": [
        {
            "type": "memberOf",
            "expression": "attribute:ecs.os-type == linux"
        },
        {
            "type": "memberOf",
            "expression": "attribute:ecs.instance-type == vt1.24xlarge"
        }
    ],
    "containerDefinitions": [
        {
            "entryPoint": [
                "/bin/bash",
                "-c"
            ],
            "command": ["/video/ecs_ffmpeg_wrapper.sh"],
            "linuxParameters": {
                "devices": [
                    {
                        "containerPath": "/dev/dri/renderD128",
                        "hostPath": "/dev/dri/renderD128",
                        "permissions": [
                            "read",
                            "write"
                        ]
                    },
                    {
                        "containerPath": "/dev/dri/renderD129",
                        "hostPath": "/dev/dri/renderD129",
                        "permissions": [
                            "read",
                            "write"
                        ]
                    },
                    {
                        "containerPath": "/dev/dri/renderD130",
                        "hostPath": "/dev/dri/renderD130",
                        "permissions": [
                            "read",
                            "write"
                        ]
                    },
                    {
```

```
                    "containerPath": "/dev/dri/renderD131",
                    "hostPath": "/dev/dri/renderD131",
                    "permissions": [
                        "read",
                        "write"
                    ]
                },
                {
                    "containerPath": "/dev/dri/renderD132",
                    "hostPath": "/dev/dri/renderD132",
                    "permissions": [
                        "read",
                        "write"
                    ]
                },
                {
                    "containerPath": "/dev/dri/renderD133",
                    "hostPath": "/dev/dri/renderD133",
                    "permissions": [
                        "read",
                        "write"
                    ]
                },
                {
                    "containerPath": "/dev/dri/renderD134",
                    "hostPath": "/dev/dri/renderD134",
                    "permissions": [
                        "read",
                        "write"
                    ]
                },
                {
                    "containerPath": "/dev/dri/renderD135",
                    "hostPath": "/dev/dri/renderD135",
                    "permissions": [
                        "read",
                        "write"
                    ]
                },
                {
                    "containerPath": "/dev/dri/renderD136",
                    "hostPath": "/dev/dri/renderD136",
                    "permissions": [
                        "read",
```

```
                        "write"
                    ]
                },
                {
                    "containerPath": "/dev/dri/renderD137",
                    "hostPath": "/dev/dri/renderD137",
                    "permissions": [
                        "read",
                        "write"
                    ]
                },
                {
                    "containerPath": "/dev/dri/renderD138",
                    "hostPath": "/dev/dri/renderD138",
                    "permissions": [
                        "read",
                        "write"
                    ]
                },
                {
                    "containerPath": "/dev/dri/renderD139",
                    "hostPath": "/dev/dri/renderD139",
                    "permissions": [
                        "read",
                        "write"
                    ]
                },
                {
                    "containerPath": "/dev/dri/renderD140",
                    "hostPath": "/dev/dri/renderD140",
                    "permissions": [
                        "read",
                        "write"
                    ]
                },
                {
                    "containerPath": "/dev/dri/renderD141",
                    "hostPath": "/dev/dri/renderD141",
                    "permissions": [
                        "read",
                        "write"
                    ]
                },
                {
```

```
                        "containerPath": "/dev/dri/renderD142",
                        "hostPath": "/dev/dri/renderD142",
                        "permissions": [
                            "read",
                            "write"
                        ]
                    },
                    {
                        "containerPath": "/dev/dri/renderD143",
                        "hostPath": "/dev/dri/renderD143",
                        "permissions": [
                            "read",
                            "write"
                        ]
                    }
                ]
            },
            "mountPoints": [
                {
                    "containerPath": "/video",
                    "sourceVolume": "video_file"
                }
            ],
            "cpu": 0,
            "memory": 12000,
            "image": "0123456789012.dkr.ecr.us-west-2.amazonaws.com/aws/xilinx-
xffmpeg",
            "essential": true,
            "name": "xilinix-xffmpeg"
        }
    ],
    "volumes": [
        {
            "name": "video_file",
            "host": {"sourcePath": "/home/ec2-user"}
        }
    ]
}
```

2. タスク定義を登録します。

```
aws ecs register-task-definition --family vt1-24xlarge-xffmpeg-processor --cli-
input-json file://vt1-24xlarge-xffmpeg-linux.json --region us-east-1
```

AWS Neuron 機械学習ワークロードでの Amazon ECS タスク定義

機械学習ワークロード用のクラスターには、Amazon EC2 Trn1、Amazon EC2 Trn2、Amazon EC2 Inf1 (Inf1 は EC2 起動タイプでのみサポートされています)、Amazon EC2 Inf2 インスタンスを使用できます。

Amazon EC2 Trn1 および Trn2 インスタンスには、AWS Trainium チップが搭載されています。これらのインスタンスは、クラウドでの機械学習用に高性能で低コストのトレーニングを提供します。Trn1 または Trn2 インスタンスで AWS Neuron を用いる機械学習フレームワークを使用して、機械学習推論モデルをトレーニングできます。その後、Inf1 インスタンス (Inf1 は EC2 起動タイプでのみサポートされています) または Inf2 インスタンスでモデルを実行して、AWS Inferentia チップのアクセラレーションを使用できます。

Amazon EC2 Inf1 インスタンスと Inf2 インスタンスは、AWS Inferentia チップを搭載しています。これらは、クラウドで高性能かつ最低レベルのコストの推論を提供します。

機械学習モデルは、専用の Software Developer Kit (SDK) である AWS Neuron を使用してコンテナにデプロイされます。この SDK は、AWS 機械学習チップの機械学習パフォーマンスを最適化するコンパイラ、ランタイム、およびプロファイリングツールからなります。 AWSNeuron は、TensorFlow、PyTorch、Apache MXNet などの一般的な機械学習フレームワークをサポートしています。

考慮事項

Amazon ECS での Neuron のデプロイを開始する前に、以下の点を考慮してください。

- 起動タイプに応じて、クラスターでは Trn1、Trn2、Inf1、Inf2、およびその他のインスタンスの組み合わせを含めることができます。
- AWS Neuron をサポートする機械学習フレームワークを使用するコンテナ内に、Linux アプリケーションが必要です。

> **Important**
>
> 他のフレームワークを使用するアプリケーションでは、Trn1、Trn2、Inf1、Inf2 インスタンスでパフォーマンスが向上しない場合があります。

- Amazon ECS は、Neuron デバイスアクセスを設定するための 2 つのアプローチに対応しています。

- マネージド Neuron デバイスの割り当て – コンテナ定義のタイプ `NeuronDevice` で `resourceRequirements` パラメータを使用します。Amazon ECS は Neuron デバイスを自動的に検出し、コンテナに割り当てます。マネージドインスタンスでのみ使用できます。詳細については、「マネージド Neuron デバイスの割り当て」を参照してください。
- 手動 Neuron デバイスの仕様 – `linuxParameters.devices` パラメータを使用して Neuron デバイスパスを明示的に指定します。EC2 起動タイプとマネージドインスタンスの両方で使用できます。詳細については、「手動 Neuron デバイスの仕様」を参照してください。

> **Important**
>
> 競合を避けるため、1 つのアプローチのみを一貫して使用してください。

マネージド Neuron デバイスの割り当て

マネージドインスタンスでは、コンテナ定義の `resourceRequirements` パラメータを使用して Neuron デバイスをリクエストできます。Amazon ECS は、インスタンス上の Neuron デバイスを自動的に検出し、タスクに割り当て、インスタンス上のすべての Neuron デバイスにアクセスできるコンテナを設定します。タスクにはすべてのデバイスへの排他的アクセスが必要なため、インスタンスごとに実行される Neuron タスクは 1 つのみです。

> **Note**
>
> `Inf1` インスタンスは EC2 起動タイプでのみサポートされています。Inf1 インスタンスを使用するには、「手動 Neuron デバイスの仕様」を参照してください。

Neuron インスタンスの選択

マネージドインスタンスワークロードの Neuron 対応インスタンスタイプを選択するには、キャパシティプロバイダーの起動テンプレートで `instanceRequirements` オブジェクトを使用します。次の属性を使用して、Neuron 対応インスタンスを選択できます。

- `acceleratorManufacturers` – `amazon-web-services` を使用して、AWS アクセラレーター (Inferentia と Trainium を含む) を持つインスタンスを選択します。
- `acceleratorNames` – `inferentia2`、`trainium`、または `trainium2` を使用して、特定のアクセラレーターチップを選択します。

- allowedInstanceTypes – inf* および trn* を使用して、名前で Neuron インスタンスタイプを選択します。

次の例では allowedInstanceTypes を使用しています。

```
{
    "instanceRequirements": {
        "allowedInstanceTypes": ["inf*", "trn*"]
    }
}
```

タスク定義

タスク定義で Neuron デバイスをリクエストするには、タイプ NeuronDevice と値 ALL の resourceRequirements エントリを追加します。これにより、インスタンス上のすべての Neuron デバイスへの排他的アクセスがコンテナに付与されます。

以下の制約が適用されます。

- resourceRequirements の NeuronDevice で指定できるコンテナ定義は最大 1 つです。
- 同じタスク定義の Neuron デバイスでは、resourceRequirements をタイプ NeuronDevice および linuxParameters.devices と組み合わせることはできません。

タスクが開始されたら、DescribeTasks API オペレーションを呼び出して Neuron デバイスの割り当てを検証できます。レスポンスには、割り当てられた Neuron デバイスの ID を示す、各コンテナの neuronDeviceIds フィールドが含まれます。DescribeContainerInstances API オペレーションを呼び出して、コンテナインスタンスの registeredResources および remainingResources フィールドに NEURON_DEVICES を表示することもできます。

タスク定義の例については、「Neuron タスク定義の例」を参照してください。

手動 Neuron デバイスの仕様

このアプローチでは、linuxParameters.devices パラメータを使用してタスク定義で AWS Trainium または AWS Inferentia デバイスパスを手動で指定します。このアプローチは、EC2 起動タイプとマネージドインスタンスの両方で機能します。

各 AWS Trainium または AWS Inferentia チップで実行できる推論または推論トレーニングのタスクは 1 つだけです。各タスクに異なるデバイスを割り当てることで、インスタンス上のチップの数だけタスクを実行できます。

EC2 起動タイプの場合、タスク配置の制約を設定するときにインスタンスタイプの属性を使用して、指定したインスタンスタイプでタスクが起動するようにできます。詳細については、「Amazon ECS がタスクをコンテナインスタンスに配置する方法」を参照してください。

タスク定義の要件

タスク定義は、1 つのインスタンスタイプに固有である必要があります。コンテナでは、ホストコンテナインスタンス向けに用意された、固有の AWS Trainium または AWS Inferentia デバイスを使用するよう設定する必要があります。これは、`linuxParameters` パラメータを使用して設定します。次の表に、各インスタンスタイプに固有のチップの詳細を示します。

インスタンスタイプ	vCPU	RAM (GiB)	AWS ML アクセラレーターチップ	デバイスへのパス
trn1.2xlarge	8	32	1	`/dev/neuron0`
trn1.32xlarge	128	512	16	`/dev/neuron0 ,/dev/neuron1 ,/dev/neuron2 ,/dev/neuron3 ,/dev/neuron4 ,/dev/neuron5 ,/dev/neuron6 ,/dev/neuron7 ,/dev/neuron8 ,/dev/neuron9 ,/dev/neuron10 ,/dev/neur`

インスタンスタイプ	vCPU	RAM (GiB)	AWS ML アクセラレーターチップ	デバイスへのパス
				on11 ,/dev/neuron12 ,/dev/neuron13 ,/dev/neuron14 ,/dev/neuron15
trn2.48xlarge	192	2048	16	/dev/neuron0 ,/dev/neuron1 ,/dev/neuron2 ,/dev/neuron3 ,/dev/neuron4 ,/dev/neuron5 ,/dev/neuron6 ,/dev/neuron7 ,/dev/neuron8 ,/dev/neuron9 ,/dev/neuron10 ,/dev/neuron11 ,/dev/neuron12 ,/dev/neuron13 ,/dev/neuron14 ,/dev/neuron15

インスタンスタイプ	vCPU	RAM (GiB)	AWS ML アクセラレーターチップ	デバイスへのパス
inf1.xlarge	4	8	1	`/dev/neuron0`
inf1.2xlarge	8	16	1	`/dev/neuron0`
inf1.6xlarge	24	48	4	`/dev/neuron0 ,/dev/neuron1 ,/dev/neuron2 ,/dev/neuron3`

インスタンスタイプ	vCPU	RAM (GiB)	AWS ML アクセラレーターチップ	デバイスへのパス
inf1.24xlarge	96	192	16	/dev/neuron0 ,/dev/neuron1 ,/dev/neuron2 ,/dev/neuron3 ,/dev/neuron4 ,/dev/neuron5 ,/dev/neuron6 ,/dev/neuron7 ,/dev/neuron8 ,/dev/neuron9 ,/dev/neuron10 ,/dev/neuron11 ,/dev/neuron12 ,/dev/neuron13 ,/dev/neuron14 ,/dev/neuron15
inf2.xlarge	4	16	1	/dev/neuron0
inf2.8xlarge	32	128	1	/dev/neuron0

インスタンスタイプ	vCPU	RAM (GiB)	AWS ML アクセラレーターチップ	デバイスへのパス
inf2.24xlarge	96	384	6	/dev/neuron0 ,/dev/neuron1 ,/dev/neuron2 ,/dev/neuron3 ,/dev/neuron4 ,/dev/neuron5
inf2.48xlarge	192	768	12	/dev/neuron0 ,/dev/neuron1 ,/dev/neuron2 ,/dev/neuron3 ,/dev/neuron4 ,/dev/neuron5 ,/dev/neuron6 ,/dev/neuron7 ,/dev/neuron8 ,/dev/neuron9 ,/dev/neuron10 ,/dev/neuron11

タスク定義の例については、「Neuron タスク定義の例」を参照してください。

マネージドインスタンス

マネージドインスタンスは、Neuron ドライバーを含む AMI を自動的に使用します。追加の AMI 設定は必要ありません。

EC2 起動タイプ

Amazon ECS では、Amazon Linux 2023 ベースの Amazon ECS 最適化 AMI が、AWS Trainium および AWS Inferentia のワークロード用に用意されています。これには、Docker 用の AWS Neuron ドライバーとランタイムが付属しています。この AMI により、Amazon ECS 上で機械学習推論ワークロードの実行が容易になります。

Amazon EC2 の Trn1、Inf1、Inf2 インスタンスを起動する際は、Amazon ECS に最適化された Amazon Linux 2023 (Neuron) AMI を使用することをお勧めします。

現在の Amazon ECS に最適化された Amazon Linux 2023 (Neuron) AMI を取得するには、AWS CLI で次のコマンドを使用します。

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2023/neuron/recommended
```

Neuron タスク定義の例

マネージドデバイスの割り当ての例

次の例は、`resourceRequirements` パラメータを使用してインスタンス上のすべての Neuron デバイスをリクエストするタスク定義を示しています。このアプローチは、マネージドインスタンスでのみ使用できます。

```
{
    "family": "ecs-neuron",
    "requiresCompatibilities": ["MANAGED_INSTANCES"],
    "networkMode": "awsvpc",
    "cpu": "8192",
    "memory": "16384",
    "executionRoleArn": "${YOUR_EXECUTION_ROLE}",
    "containerDefinitions": [
        {
            "name": "neuron-inference",
            "image": "763104351884.dkr.ecr.us-east-1.amazonaws.com/huggingface-vllm-inference-neuronx:0.11.0-optimum0.4.5-neuronx-py310-sdk2.26.1-ubuntu22.04",
            "essential": true,
            "command": [
                "--model", "${YOUR_HUGGING_FACE_MODEL_ID}",
                "--port", "8080",
                "--tensor-parallel-size", "2",
                "--allow-non-cached-model"
```

```
            ],
            "portMappings": [
                {
                    "containerPort": 8080,
                    "protocol": "tcp"
                }
            ],
            "resourceRequirements": [
                {
                    "type": "NeuronDevice",
                    "value": "ALL"
                }
            ]
        }
    ]
}
```

この例では、コンテナイメージに AWS Neuron 用に最適化された vLLM 推論サーバーが含まれます。イメージのエントリポイントは、HuggingFace からモデルをダウンロードして Neuron 用にコンパイルし、ポート 8080 で OpenAI 互換 API サーバーを起動します。*${YOUR_HUGGING_FACE_MODEL_ID}* をお使いの HuggingFace モデル ID に置き換えます。

手動デバイスの仕様の例

次の例は、EC2 起動タイプと `linuxParameters.devices` を使用して Neuron デバイスパスを指定するための `inf1.xlarge` の Linux タスク定義を示しています。

```
{
    "family": "ecs-neuron",
    "requiresCompatibilities": ["EC2"],
    "placementConstraints": [
        {
            "type": "memberOf",
            "expression": "attribute:ecs.os-type == linux"
        },
        {
            "type": "memberOf",
            "expression": "attribute:ecs.instance-type == inf1.xlarge"
        }
    ],
    "executionRoleArn": "${YOUR_EXECUTION_ROLE}",
    "containerDefinitions": [
        {
```

```
            "entryPoint": [
                "/usr/local/bin/entrypoint.sh",
                "--port=8500",
                "--rest_api_port=9000",
                "--model_name=resnet50_neuron",
                "--model_base_path=s3://amzn-s3-demo-bucket/resnet50_neuron/"
            ],
            "portMappings": [
                {
                    "hostPort": 8500,
                    "protocol": "tcp",
                    "containerPort": 8500
                },
                {
                    "hostPort": 8501,
                    "protocol": "tcp",
                    "containerPort": 8501
                },
                {
                    "hostPort": 0,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ],
            "linuxParameters": {
                "devices": [
                    {
                        "containerPath": "/dev/neuron0",
                        "hostPath": "/dev/neuron0",
                        "permissions": [
                            "read",
                            "write"
                        ]
                    }
                ],
                "capabilities": {
                    "add": [
                        "IPC_LOCK"
                    ]
                }
            },
            "cpu": 0,
            "memoryReservation": 1000,
```

```
            "image": "763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-
inference-neuron:1.15.4-neuron-py37-ubuntu18.04",
            "essential": true,
            "name": "resnet50"
        }
    ]
}
```

深層学習インスタンスでの Amazon ECS タスク定義

Amazon ECS で深層学習ワークロードを使用するには、Amazon EC2 DL1インスタンスをクラスターに登録します。Amazon EC2 DL1 インスタンスには、Habana Labs (インテル子会社) の Gaudi アクセラレータを搭載しています。Habana Gaudiアクセラレータには、Habana SynapseAI SDK を使用して接続します。この SDK は、TensorFlow および PyTorch による、一般的な機械学習フレームワークをサポートしています。

考慮事項

Amazon ECS 上で DL1 のデプロイを開始する前に、以下の点を考慮してください。

- クラスターには、DL1 コンテナインスタンスと DL1 以外のインスタンスを混在させることができます。
- サービスの作成時、またはスタンドアロンタスクの実行時、特にタスク配置制約を設定する場合には、インスタンスタイプ属性を使用して、タスクが起動されるコンテナインスタンスを確定できます。これにより、リソースが効果的に使用され、深層学習ワークロードのタスクが DL1 インスタンス上に配置されることが保証されます。詳細については、「Amazon ECS がタスクをコンテナインスタンスに配置する方法」を参照してください。

  次の例では、`dl1.24xlarge` クラスターの `default` インスタンスでタスクを実行します。

  ```
  aws ecs run-task \
       --cluster default \
       --task-definition ecs-dl1-task-def \
       --placement-constraints type=memberOf,expression="attribute:ecs.instance-type ==
   dl1.24xlarge"
  ```

DL1 AMI の使用

Amazon ECS 向けの Amazon EC2 DL1 インスタンス上で AMI を実行するためには、以下の 3 つのオプションがあります。

- Habana がここで提供している AWS Marketplace AMI。
- Amazon Web Services によって提供される Habana 深層学習 AMI。これには、Amazon ECS コンテナエージェントは含まれないため、このエージェントを個別にインストールする必要があります。
- GitHub リポジトリを通じて提供されるカスタム AMI をビルドするには、Packer を使用します。詳細については、「the Packer documentation」(Packer ドキュメント) を参照してください。

Amazon ECS タスク定義での深層学習の指定

Habana Gaudi アクセラレータ付きの深層学習コンテナを Amazon ECS で実行するには、AWS Deep Learning Containers が提供する Habana SynapseAI を使用して TensorFlow または PyTorch 向けに深層学習モデルを提供する、ビルド済みコンテナのコンテナ定義が、対象のタスク定義内に含まれている必要があります。

次のコンテナイメージには TensorFlow 2.7.0 と Ubuntu 20.04 が含まれています。Neuron 用に最適化された事前構築済みの深層学習コンテナの完全なリストは、GitHub で確認できます。詳細については、「Habana Training Containers」(Habana トレーニングコンテナ) を参照してください。

```
763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-training-habana:2.7.0-hpu-py38-synapseai1.2.0-ubuntu20.04
```

以下の例で、Amazon EC2 の Linux コンテナのタスク定義で使用する構文を示します。この例で使用するイメージには、`vault.habana.ai/gaudi-docker/1.1.0/ubuntu20.04/habanalabs/tensorflow-installer-tf-cpu-2.6.0:1.1.0-614` から入手が可能な Habana Labs のシステム管理インターフェイスツール (HL-SMI) が含まれています。

```
{
    "family": "dl-test",
    "requiresCompatibilities": ["EC2"],
    "placementConstraints": [
        {
            "type": "memberOf",
            "expression": "attribute:ecs.os-type == linux"
```

```
        },
        {
            "type": "memberOf",
            "expression": "attribute:ecs.instance-type == dl1.24xlarge"
        }
    ],
    "networkMode": "host",
    "cpu": "10240",
    "memory": "1024",
    "containerDefinitions": [
        {
            "entryPoint": [
                "sh",
                "-c"
            ],
            "command": ["hl-smi"],
            "cpu": 8192,
            "environment": [
                {
                    "name": "HABANA_VISIBLE_DEVICES",
                    "value": "all"
                }
            ],
            "image": "vault.habana.ai/gaudi-docker/1.1.0/ubuntu20.04/habanalabs/
tensorflow-installer-tf-cpu-2.6.0:1.1.0-614",
            "essential": true,
            "name": "tensorflow-installer-tf-hpu"
        }
    ]
}
```

64 ビット ARM ワークロードでの Amazon ECS タスク定義

Amazon ECS は 64 ビット ARM アプリケーションの使用をサポートしています。アプリケーションは、AWS Graviton プロセッサ搭載のプラットフォーム上で実行できます。この構成、さまざまなワークロードに適しています。対象となるのは、アプリケーションサーバー、マイクロサービス、ハイパフォーマンスコンピューティング、CPU ベースの機械学習推論、ビデオエンコーディング、電子回路設計オートメーション、ゲーム配信、オープンソースデータベース、インメモリーキャッシュなどのワークロードです。

考慮事項

64 ビット ARM アーキテクチャを使用するタスク定義のデプロイを開始する前に、次の点を考慮してください。

- アプリケーションは、Fargate または EC2 を使用できます。
- アプリケーションは Linux オペレーティングシステムのみで使用できます。
- Fargate タイプの場合、アプリケーションは Fargate プラットフォームバージョン `1.4.0` 以降を使用する必要があります。
- アプリケーションは、モニタリングに Fluent Bit または CloudWatch を使用できます。
- Fargate の場合、次の AWS リージョンでは 64 ビット ARM のワークロードには対応していません。
  - 米国東部 (バージニア北部)、`use1-az3` アベイラビリティーゾーン
- EC2 の場合、以下を参照して、使用したいインスタンスタイプが利用地域でサポートされていることを確認してください。
  - Amazon EC2 M6g インスタンス
  - Amazon EC2 T4g インスタンス
  - Amazon EC2 C6g インスタンス
  - Amazon EC2 R6gd インスタンス
  - Amazon EC2 X2gd インスタンス

  フィルターで Amazon EC2 の `describe-instance-type-offerings` コマンドを使用して、リージョンのインスタンスオファリングを表示することもできます。

```
aws ec2 describe-instance-type-offerings --filters Name=instance-
type,Values=instance-type --region region
```

  次の例では、米国東部 (バージニア北部) (us-east-1) リージョンで M6 インスタンスタイプが使用可能かどうかを確認します。

```
aws ec2 describe-instance-type-offerings --filters "Name=instance-type,Values=m6*" --
region us-east-1
```

  詳細については、「Amazon EC2 コマンドラインリファレンス」の「インスタンスタイプオファリングの説明」を参照してください。

Amazon ECS タスク定義での ARM アーキテクチャの指定

ARM アーキテクチャを使用するには、cpuArchitecture タスク定義パラメータに ARM64 を指定します。

次の例では、タスク定義で ARM アーキテクチャが指定されています。JSON 形式にあります。

```
{
    "runtimePlatform": {
        "operatingSystemFamily": "LINUX",
        "cpuArchitecture": "ARM64"
    },
...
}
```

次の例に、「hello world」を表示する ARM アーキテクチャのタスク定義を示します。

```
{
 "family": "arm64-testapp",
 "networkMode": "awsvpc",
 "containerDefinitions": [
    {
        "name": "arm-container",
        "image": "public.ecr.aws/docker/library/busybox:latest",
        "cpu": 100,
        "memory": 100,
        "essential": true,
        "command": [ "echo hello world" ],
        "entryPoint": [ "sh", "-c" ]
    }
 ],
 "requiresCompatibilities": [ "EC2" ],
 "cpu": "256",
 "memory": "512",
 "runtimePlatform": {
        "operatingSystemFamily": "LINUX",
        "cpuArchitecture": "ARM64"
  },
 "executionRoleArn": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole"
}
```

Amazon ECS ログを CloudWatch に送信する

タスクのコンテナを設定して CloudWatch Logs にログ情報を送信できます。タスクで Fargate を使用すると、コンテナからログを表示できます。EC2 を使用すると、コンテナからの異なるログを 1 か所で便利に表示できます。また、コンテナインスタンスのディスク容量をコンテナログが占有してしまうことも防止できます。

> **Note**
>
> タスクのコンテナによってログ記録される情報のタイプは、ENTRYPOINT コマンドによって大きく異なります。デフォルトでは、キャプチャされるログには、コンテナをローカルに実行した場合に通常はインタラクティブターミナルに表示されるコマンド出力 (STDOUT および STDERR I/O ストリーム) が表示されます。awslogs ログドライバーは、これらのログを Docker から CloudWatch Logs に渡します。Docker がログを処理する方法 (ファイルデータやストリームをキャプチャする別の方法) の詳細については、Docker ドキュメントの コンテナまたはサービスのログを表示する を参照してください。

your Amazon ECS コンテナインスタンスから CloudWatch Logs にシステムログを送信するには、「Amazon CloudWatch Logs ユーザーガイド」の「ログファイルをモニタリングする」と「CloudWatch Logs クォータ」を参照してください。

Fargate

タスクで Fargate を使用する場合、awslogs ログドライバーを有効化するには、必要な logConfiguration パラメータをタスク定義に追加する必要があります。詳細については、「Amazon ECS タスク定義の例: CloudWatch にログをルーティングする」を参照してください。

Fargate の Windows コンテナでは、タスク定義パラメータに & \ < > ^ | などの特殊文字が含まれている場合は、次のいずれかのオプションを実行します。

- パラメータ文字列全体を二重引用符で囲んだエスケープ (\) を追加する

  例

  ```
  "awslogs-multiline-pattern": "\"^[|DEBUG|INFO|WARNING|ERROR\"",
  ```

- 各特殊文字の周囲にエスケープ (^) 文字を追加する

  例

```
"awslogs-multiline-pattern": "^^[^|DEBUG^|INFO^|WARNING^|ERROR",
```

EC2

タスクで EC2 を使用する場合、awslogs ログドライバーをオンにするには、Amazon ECS コンテナインスタンスに、コンテナエージェントのバージョン 1.9.0 以降が必要です。エージェントのバージョンの確認と最新バージョンへの更新については、「Amazon ECS コンテナエージェントをアップデートする」を参照してください。

> **Note**
>
> Amazon ECS に最適化された AMI か、もしくは少なくともバージョン 1.9.0-1 の ecs-init パッケージを含むカスタム AMI を使用する必要があります。カスタム AMI を使用する場合、docker run ステートメント内で以下の環境変数を使用するか、または環境変数ファイルを使用することにより、エージェントの起動時に Amazon EC2 インスタンス上で awslogs ロギングドライバーが使用可能となることを指定する必要があります。
>
> ```
> ECS_AVAILABLE_LOGGING_DRIVERS=["json-file","awslogs"]
> ```

Amazon ECS コンテナインスタンスでは、コンテナインスタンスで起動する IAM ロールに logs:CreateLogStream および logs:PutLogEvents アクセス許可も必要になります。Amazon ECS で awslogs ログドライバーのサポートを有効にする前に Amazon ECS コンテナインスタンスを作成した場合は、このアクセス許可の追加が必要となる場合があります。ecsTaskExecutionRole はタスクに割り当てられた時点で使用され、また、正しいアクセス許可が含まれている必要があります。タスクの実行ロールの詳細については、「Amazon ECS タスク実行IAM ロール」を参照してください。コンテナインスタンスでコンテナインスタンス用の管理 IAM ポリシーを使用している場合、通常そのコンテナインスタンスには、適切なアクセス許可が付与されています。コンテナインスタンスのマネージド IAM ポリシーについては、「Amazon ECS コンテナインスタンスの IAM ロール」を参照してください。

Amazon ECS タスク定義の例: CloudWatch にログをルーティングする

コンテナがログを CloudWatch に送信する前に、タスク定義でコンテナの awslogs ログドライバーを指定する必要があります。ログパラメータの詳細については、「ストレージとログ記録」を参照してください

下にあるタスク定義 JSON には、各コンテナに指定された logConfiguration オブジェクトが含まれています。その 1 つは、awslogs-wordpress というロググループにログを送信する WordPress コンテナ用です。もう 1 つは、awslogs-mysql というロググループにログを送信する MySQL コンテナ用です。どちらのコンテナも awslogs-example ログストリームプレフィックスを使用します。

```
{
    "containerDefinitions": [
        {
            "name": "wordpress",
            "links": [
                "mysql"
            ],
            "image": "public.ecr.aws/docker/library/wordpress:latest",
            "essential": true,
            "portMappings": [
                {
                    "containerPort": 80,
                    "hostPort": 80
                }
            ],
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-create-group": "true",
                    "awslogs-group": "awslogs-wordpress",
                    "awslogs-region": "us-west-2",
                    "awslogs-stream-prefix": "awslogs-example"
                }
            },
            "memory": 500,
            "cpu": 10
        },
        {
            "environment": [
                {
                    "name": "MYSQL_ROOT_PASSWORD",
                    "value": "password"
                }
            ],
            "name": "mysql",
            "image": "public.ecr.aws/docker/library/mysql:latest",
            "cpu": 10,
```

```
            "memory": 500,
            "essential": true,
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-create-group": "true",
                    "awslogs-group": "awslogs-mysql",
                    "awslogs-region": "us-west-2",
                    "awslogs-stream-prefix": "awslogs-example",
                    "mode": "non-blocking",
                    "max-buffer-size": "25m"
                }
            }
        }
    ],
    "family": "awslogs-example"
}
```

次のステップ

- CloudWatch AWS CLI または API を使用してロググループの保持ポリシーを設定できます (オプション)。詳細は「AWS Command Line Interfaceリファレンス」の「put-retention-policy」を参照してください。
- `awslogs` ログドライバーを使用するタスク定義をコンテナ定義ログ設定に登録すると、タスク定義を使用してタスクを実行またはサービスを作成し、CloudWatch Logs へのログの送信を開始できます。詳細については、「Amazon ECS タスクとしてのアプリケーションの実行」および「Amazon ECS のローリング更新デプロイの作成」を参照してください。

Amazon ECS ログを AWS サービスまたは AWS Partner に送信する

FireLens for Amazon ECS では、タスク定義パラメータを使用してログを AWS サービスや AWS Partner Network (APN) の宛先にルーティングし、ログを保存および分析できます。AWS Partner Network は、プログラム、専門知識、リソースを活用して顧客向けサービスの構築、マーケティング、販売を行うパートナーのグローバルコミュニティです。詳細については、「AWS Partner」を参照してください。FireLens は Fluentd および Fluent Bit と連携しています。Fluent Bit イメージ用に AWS を提供していますが、Fluentd や Fluent Bit のイメージはご用意いただいたものを使用することもできます。

デフォルトでは、Amazon ECS は、Firelens コンテナを使用するコンテナより前に Firelens コンテナを起動するようにコンテナの依存関係を設定します。また Firelens コンテナは、それを使用するすべてのコンテナが停止した後で停止します。

この機能を使用するには、タスク用の IAM ロールを作成し、AWS のサービスを使用するために必要なアクセス許可をタスクに付与する必要があります。例えば、コンテナから Firehose にログをルーティングする場合、タスクには `firehose:PutRecordBatch` API を呼び出すためのアクセス許可が必要です。詳細については、IAM ユーザーガイドの「IAM ID アクセス許可の追加と削除」を参照してください。

以下の場合は、タスクに Amazon ECS タスク実行ロールが必要になることもあります。詳細については、「Amazon ECS タスク実行IAM ロール」を参照してください。

- タスクが Fargate でホストされていて、Amazon ECR からコンテナイメージをプルしたり、ログ設定で AWS Secrets Manager の機密データを参照したりする場合は、タスク実行 IAM ロールを含める必要があります。
- Amazon S3 でホストされているカスタム設定ファイルを使用する場合は、タスク実行 IAM ロールに `s3:GetObject` アクセス許可が含まれている必要があります。

FireLens for Amazon ECS を使用する際は、以下の点を考慮してください。

- コンソール上でコンテナ名を簡単に区別できるよう、`my_service_` をログコンテナ名に追加することをおすすめします。
- Amazon ECS はデフォルトで、アプリケーションコンテナと FireLens コンテナの間に開始コンテナ順序依存関係を追加します。アプリケーションコンテナと FireLens コンテナ間でコンテナ順序を指定すると、デフォルトの開始コンテナ順序が上書きされます。
- FireLens for Amazon ECS は、Linux の AWS Fargate と Linux の Amazon EC2 の両方でホストされたタスクでサポートされます。Windows コンテナは FireLens をサポートしていません。

  Windows コンテナの集中ロギングを設定する方法については、「Centralized logging for Windows containers on Amazon ECS using Fluent Bit」(Fluent Bit を使用した Amazon ECS での Windows コンテナの集中ロギング) を参照してください。
- Amazon ECS の FireLens は、CloudFormation テンプレートを使用して設定できます。詳細については、AWS CloudFormation ユーザーガイドの「AWS::ECS::TaskDefinition FirelensConfiguration」を参照してください。
- FireLens はポート 24224 でリッスンするため、FireLens ログルーターがタスク外に到達できないようにするには、タスクが使用するセキュリティグループでポート 24224 でのインバウンドト

ラフィックを許可してはなりません。awsvpc ネットワークモードを使用するタスクの場合、これは、そのタスクに関連付けられたセキュリティグループです。host ネットワークモードを使用するタスクでは、そのタスクをホストする Amazon EC2 インスタンスに関連付けられているセキュリティグループです。bridge ネットワークモードを使用するタスクの場合、ポート 24224 を使用するポートマッピングを作成しないでください。

- bridge ネットワークモードを使用するタスクの場合、FireLens 設定のコンテナは、それに依存するアプリケーションコンテナが開始する前に開始する必要があります。コンテナの開始順序を制御するには、タスク定義の依存関係条件を使用します。詳細については、「コンテナの依存関係」を参照してください。

> **Note**
>
> FireLens 設定のコンテナ定義で依存関係条件パラメータを使用する場合は、各コンテナに START または HEALTHY 条件要件があることを確認してください。

- デフォルトでは、FireLens はクラスターとタスクの定義名、およびクラスターの Amazon リソースネーム (ARN) をメタデータキーとして stdout/stderr コンテナログに追加します。メタデータ形式の例を次に示します。

```
"ecs_cluster": "cluster-name",
"ecs_task_arn": "arn:aws:ecs:region:111122223333:task/cluster-
name/f2ad7dba413f45ddb4EXAMPLE",
"ecs_task_definition": "task-def-name:revision",
```

ログにメタデータを含めたくない場合は、タスク定義の「firelensConfiguration」セクションで enable-ecs-log-metadata を false に設定します。

```
"firelensConfiguration":{
   "type":"fluentbit",
   "options":{
      "enable-ecs-log-metadata":"false",
      "config-file-type":"file",
      "config-file-value":"/extra.conf"
}
```

非ルートユーザーとして実行するように FireLens コンテナを設定できます。以下の点を考慮してください。

- 非ルートユーザーとして実行するように FireLens コンテナを設定するには、次のいずれかの形式でユーザーを指定する必要があります。
  - `uid`
  - `uid:gid`
  - `uid:group`

  コンテナ定義について、詳細は「Amazon Elastic Container Service API リファレンス」の「ContainerDefinition」を参照してください。

  FireLens コンテナは UNIX ソケット経由でアプリケーションログを受け取ります。Amazon ECS エージェントは `uid` を使用してソケットディレクトリの所有権を FireLens コンテナに割り当てます。
- 非ルートユーザーとして実行するための FireLens コンテナ設定は、Amazon ECS エージェントバージョン `1.96.0` 以降、および Amazon ECS 最適化 AMI のバージョン `v20250716` 以降でサポートされています。
- FireLens コンテナのユーザーを指定する場合、`uid` は一意でなければならず、タスク内の他のコンテナまたはコンテナインスタンスに属する他のプロセスには使用されません。

Amazon S3 でホストするファイルや Amazon S3 内のファイルなど、Amazon ECS で複数の設定ファイルを使用する方法については、「Init process for Fluent Bit on ECS, multi-config support」を参照してください。

構成例については、「Amazon ECS タスク定義の例: FireLens にログをルーティングする」を参照してください。

高スループットのログ有効化と設定について、詳細は「高スループットの Amazon ECS ログの設定」を参照してください。

高スループットの Amazon ECS ログの設定

ログスループットの高いシナリオでは、FireLens と Fluent Bit を組み合わせた `awsfirelens` ログドライバーを使用することをお勧めします。 Fluent Bit は、リソースを効率的に処理し、数百万のログレコードを処理できる軽量ログプロセッサです。ただし、大規模に最適なパフォーマンスを実現するには、その設定を調整する必要があります。

このセクションでは、システムの安定性を維持し、データ損失を回避しながら、高いログスループットを処理するための高度な Fluent Bit 最適化手法について説明します。

FireLens でカスタム設定ファイルを使用する方法については、「カスタム設定ファイルを使用する」を参照してください。その他の例については、GitHub の「Amazon ECS FireLens の例」を参照してください。

> **Note**
>
> `workers` や `threaded` など、このセクションの一部の設定オプションでは、Fluent Bit バージョン 3 以降の AWS が必要です。使用可能なバージョンについては、「AWS for Fluent Bit リリース」を参照してください。

チャンクを理解する

Fluent Bit はチャンクと呼ばれる単位でデータを処理します。INPUT プラグインがデータを受信すると、エンジンは OUTPUT 送信先に送信される前にメモリまたはファイルシステムに保存されるチャンクを作成します。

バッファリング動作は、INPUT セクションの `storage.type` 設定によって異なります。デフォルトでは、Fluent Bit はメモリバッファリングを使用します。高スループットまたは本番稼働シナリオでは、ファイルシステムのバッファリングはより良い耐障害性を提供します。

詳細については、Fluent Bit ドキュメントの「チャンク」および Fluent Bit リポジトリ例の AWS の「チャンクとは」を参照してください。

メモリバッファリング (デフォルト)

デフォルトでは、Fluent Bit はメモリバッファリング (`storage.type memory`) を使用します。`Mem_Buf_Limit` パラメータを使用して、INPUT プラグインあたりのメモリ使用量を制限できます。

次の例は、メモリバッファ入力設定を示します。

```
[INPUT]
    Name          tcp
    Tag           ApplicationLogs
    Port          5170
    storage.type  memory
    Mem_Buf_Limit 5MB
```

> **Important**
>
> プラグインで Mem_Buf_Limit を超えると、Fluent Bit は入力を一時停止し、新しいレコードは失われます。これにより、バックプレッシャーが発生し、アプリケーションが遅くなる可能性があります。Fluent Bit ログには次の警告が表示されます。
>
> ```
> [input] tcp.1 paused (mem buf overlimit)
> ```

メモリバッファリングは、ログスループットが低～中程度の単純なユースケースに適しています。データ損失が懸念される高スループットまたは本番環境のシナリオでは、代わりにファイルシステムのバッファリングを使用します。

詳細については、Fluent Bit ドキュメントの「バッファリングとメモリ」および Fluent Bit リポジトリ例の AWS の「メモリバッファリングのみ」を参照してください。

ファイルシステムのバッファリング

高スループットのシナリオでは、ファイルシステムのバッファリングを使用することをお勧めします。Fluent Bit がバッファリングとストレージを管理する方法の詳細については、 Fluent Bit ドキュメントの「バッファリングとストレージ」を参照してください。

ファイルシステムのバッファリングには、次の利点があります。

- バッファ容量が大きい — 通常、ディスクスペースはメモリよりも豊富です。
- 永続性 – バッファされたデータは Fluent Bit 再起動後も保持されます。
- グレースフルデグラデーション – 出力の失敗時に、メモリの枯渇を引き起こすのではなく、データがディスクに蓄積されます。

ファイルシステムのバッファリングを有効にするには、カスタム Fluent Bit 設定ファイルを指定します。次の例は、推奨される設定です。

```
[SERVICE]
    # Flush logs every 1 second
    Flush 1
    # Wait 120 seconds during shutdown to flush remaining logs
    Grace 120
    # Directory for filesystem buffering
    storage.path              /var/log/flb-storage/
```

```
    # Limit chunks stored 'up' in memory (reduce for memory-constrained environments)
    storage.max_chunks_up    32
    # Flush backlog chunks to destinations during shutdown (prevents log loss)
    storage.backlog.flush_on_shutdown On

[INPUT]
    Name forward
    unix_path /var/run/fluent.sock
    # Run input in separate thread to prevent blocking
    threaded true
    # Enable filesystem buffering for persistence
    storage.type filesystem

[OUTPUT]
    Name cloudwatch_logs
    Match *
    region us-west-2
    log_group_name /aws/ecs/my-app
    log_stream_name $(ecs_task_id)
    # Use multiple workers for parallel processing
    workers 2
    # Retry failed flushes up to 15 times
    retry_limit 15
    # Maximum disk space for buffered data for this output
    storage.total_limit_size 10G
```

主な設定パラメータ:

`storage.path`

Fluent Bit がバッファされたチャンクをディスクに保存するディレクトリ。

`storage.backlog.flush_on_shutdown`

有効にすると、Fluent Bit はシャットダウン中にすべてのバックログファイルシステムのチャンクを送信先にフラッシュしようとします。これにより、Fluent Bit が停止する前にデータ配信を確保できますが、シャットダウン時間が長くなる可能性があります。

`storage.max_chunks_up`

メモリに残っているチャンクの数。デフォルトは 128 チャンクで、各チャンクは最大 4～5 MB を使用できるため、500 MB 以上のメモリを消費する可能性があります。メモリに制約のある環境では、この値を小さくします。たとえば、バッファリングに 50 MB 使用できる場合は、これを 8～10 チャンクに設定します。

`storage.type filesystem`

入力プラグインのファイルシステムストレージを有効にします。名前にかかわらず、 Fluent Bit は `mmap` を使用してチャンクをメモリとディスクの両方にマッピングし、パフォーマンスを犠牲にすることなく永続性を提供します。

`storage.total_limit_size`

特定の OUTPUT プラグインのバッファリングされたデータの最大ディスク容量。この制限に達すると、その出力の最も古いレコードがドロップされます。サイズの詳細については、「を理解する`storage.total_limit_size`」を参照してください。

`threaded true`

Fluent Bit のメインイベントループとは別に、独自のスレッドで入力を実行します。これにより、遅い入力がパイプライン全体をブロックすることを防ぎます。

詳細については、Fluent Bit ドキュメントの「ファイルシステムのバッファリング」および Fluent Bit リポジトリ例の AWS の「ファイルシステムとメモリのバッファリング」を参照してください。

を理解する`storage.total_limit_size`

各 OUTPUT プラグインの `storage.total_limit_size` パラメータは、その出力のバッファされたデータの最大ディスク容量を制御します。この制限に達すると、その出力の最も古いレコードがドロップされ、新しいデータのためのスペースが確保されます。ディスク容量を完全に使い切ると、Fluent Bit はレコードのキューに失敗し、それらは失われます。

次の式を使用して、ログレートと必要な復旧ウィンドウに基づいて適切な `storage.total_limit_size` を計算します。

```
If log rate is in KB/s, convert to MB/s first:
log_rate (MB/s) = log_rate (KB/s) / 1000

storage.total_limit_size (GB) = log_rate (MB/s) × duration (hours) × 3600 (seconds/
hour) / 1000 (MB to GB)
```

次の表は、一般的なログレートと復旧ウィンドウの計算例を示しています。

ログレート	1 時間	6 時間	12 時間ごと	24 時間
0.25 MB/秒	0.9 GB	5.4 GB	10.8 GB	21.6 GB

ログレート	1 時間	6 時間	12 時間ごと	24 時間
0.5 MB/秒	1.8 GB	10.8 GB	21.6 GB	43.2 GB
1 MB/秒	3.6 GB	21.6 GB	43.2 GB	86.4 GB
5 MB/秒	18 GB	108 GB	216 GB	432 GB
10 MB/秒	36 GB	216 GB	432 GB	864 GB

ピークスループットを観察し、適切なバッファサイズを選択するには、メジャースループット FireLens サンプルを使用します。

式、計算例、ベンチマークを使用して、停止中のベストエフォートリカバリのためのランウェイを提供する適切な `storage.total_limit_size` を選択します。

Amazon ECS タスクストレージの要件

OUTPUT セクション全体のすべての `storage.total_limit_size` 値を合計し、オーバーヘッドのバッファを追加します。この合計により、Amazon ECS タスク定義に必要なストレージスペースが決まります。例えば、3 つの出力 × 各 10 GB = 30 GB + バッファ (5 ~ 10 GB) = 合計 35 ~ 40 GB が必要です。合計が使用可能なストレージを超える場合、Fluent Bit はレコードのキューに失敗し、それらは失われます。

次のストレージオプションが利用可能です。

バインドマウント (エフェメラルストレージ)

- AWS Fargate では、デフォルトは 20 GB のエフェメラルストレージ (最大 200 GB) です。タスク定義で `ephemeralStorage` を使用して設定します。詳細については、「AWS CloudFormation ユーザーガイド」の「EphemeralStorage」を参照してください。
- EC2 では、Amazon ECS 最適化 AMI (OS と Docker 間で共有) を使用する場合、デフォルトは 30 GB です。ルートボリュームのサイズを変更して増加させます。

Amazon EBS ボリューム

- 高可用性、高耐久性、高性能のブロックストレージを提供します。
- タスク定義で `storage.path` (デフォルト: `/var/log/flb-storage/`) を指すボリューム設定と `mountPoint` が必要です。
- 詳細については、「ボリューム設定を Amazon ECS のタスク定義の起動時刻まで延期する」を参照してください。

Amazon EFS ボリューム

- シンプルでスケーラブルなファイルストレージを提供します。
- タスク定義で `storage.path` (デフォルト: `/var/log/flb-storage/`) を指すボリューム設定と `mountPoint` が必要です。
- 詳細については、「タスク定義内での Amazon EFS ファイルシステムの指定」を参照してください。

データボリュームの詳細については、「Amazon ECS タスクのストレージオプション」を参照してください。

出力設定の最適化

ネットワークの問題、サービスの停止、送信先スロットリングにより、ログの配信が妨げられる場合があります。適切な出力設定により、データ損失のない耐障害性を保証します。

出力フラッシュが失敗した場合、 Fluent Bit はオペレーションを再試行できます。次のパラメータは再試行動作を制御します。

`retry_limit`

レコードをドロップする前の最初の試行後の最大再試行回数。デフォルトは 1 です。例えば、`retry_limit 3` は合計 4 回の試行 (最初の 1 回 + 再試行の 3 回) を意味します。本番環境では、数分の停止に対応できるよう、エクスポネンシャルバックオフを用いた再試行数を 15 回以上に設定することをお勧めします。

無制限の再試行では、`no_limits` または `False` に設定します。

- メモリバッファリングでは、無制限の再試行により、メモリ制限に達すると入力プラグインが一時停止します。
- ファイルシステムのバッファリングでは、`storage.total_limit_size` に達すると最も古いレコードがドロップされます。

> **Important**
>
> すべての再試行回数 (最初の 1 回 + 再試行の `retry_limit` 回) を使い切ると、レコードはドロップされます。`auto_retry_requests true` (デフォルト) を使用する AWS プラグインは、Fluent Bit の再試行メカニズムの前に追加の再試行レイヤーを提供します。詳細については、Fluent Bit ドキュメントの「再試行を設定する」を参照してください。

例えば、デフォルト設定 (scheduler.base 5、scheduler.cap 2000、net.connect_timeout 10s) を使用した retry_limit 3 では、約 70 秒のスケジューラ待機時間 (10 秒 + 20 秒 + 40 秒)、40 秒のネットワーク接続タイムアウト (4 回の試行 x 10 秒)、および AWS プラグインの再試行が提供され、ネットワーク条件と OS TCP タイムアウトに応じて合計約 2 ~ 10 分になります。

`scheduler.base`

再試行間の基本秒数 (デフォルト: 5)。10 秒をお勧めします。

`scheduler.cap`

再試行間の最大秒数 (デフォルト: 2000)。60 秒をお勧めします。

再試行間の待機時間は、ジッターを伴うエクスポネンシャルバックオフを使用します。

```
wait_time = random(base, min(base × 2^retry_number, cap))
```

例: `scheduler.base 10` および `scheduler.cap 60` を使用:

- 最初の再試行: 10 ~ 20 秒のランダム待機
- 2 回目の再試行: 10 ~ 40 秒のランダム待機
- 3 回目の再試行以降: 10 ~ 60 秒のランダム待機 (上限あり)

詳細については、Fluent Bitドキュメントの「再試行の待機時間を設定する」および「ネットワーク」を参照してください。

`workers`

並列出力処理のスレッド数。複数のワーカーが同時フラッシュを許可し、多くのチャンクを処理する際のスループットを向上させます。

`auto_retry_requests`

Fluent Bit の組み込み再試行メカニズムの前に追加の再試行レイヤーを提供する AWS プラグイン固有の設定。デフォルトは `true` です。有効にすると、AWS 出力プラグインは、リクエストが失敗したフラッシュと見なされ、`retry_limit` 設定の対象となる前に、失敗したリクエストを内部で再試行します。

[SERVICE] セクションの Grace パラメータは、バッファされたデータをフラッシュするためにシャットダウン中に Fluent Bit が待機する時間を設定します。Grace 期間はコンテナの stopTimeout と調整する必要があります。SIGKILL を受信する前に Fluent Bit がフラッシュを完了できるように、stopTimeout が Grace 期間を超えていることを確認します。たとえば、Grace が 120 秒の場合、 stopTimeout を 150 秒に設定します。

次の例は、高スループットシナリオで推奨されるすべての設定を含む完全な Fluent Bit 設定を示しています。

```
[SERVICE]
    # Flush logs every 1 second
    Flush 1
    # Wait 120 seconds during shutdown to flush remaining logs
    Grace 120
    # Directory for filesystem buffering
    storage.path             /var/log/flb-storage/
    # Limit chunks stored 'up' in memory (reduce for memory-constrained environments)
    storage.max_chunks_up    32
    # Flush backlog chunks to destinations during shutdown (prevents log loss)
    storage.backlog.flush_on_shutdown On
    # Minimum seconds between retries
    scheduler.base           10
    # Maximum seconds between retries (exponential backoff cap)
    scheduler.cap            60

[INPUT]
    Name forward
    unix_path /var/run/fluent.sock
    # Run input in separate thread to prevent blocking
    threaded true
    # Enable filesystem buffering for persistence
    storage.type filesystem

[OUTPUT]
    Name cloudwatch_logs
    Match *
    region us-west-2
    log_group_name /aws/ecs/my-app
    log_stream_name $(ecs_task_id)
    # Use multiple workers for parallel processing
    workers 2
    # Retry failed flushes up to 15 times
    retry_limit 15
```

```
    # Maximum disk space for buffered data for this output
    storage.total_limit_size 10G
```

データ損失シナリオを理解する

長時間の停止中や出力先に問題がある場合、レコードが失われる可能性があります。このガイドの設定に関する推奨事項は、データ損失を最小限に抑えるためのベストエフォートアプローチですが、長期にわたる障害の間のゼロ損失を保証することはできません。これらのシナリオを理解することは、レジリエンスを最大化するように Fluent Bit を設定するのに役立ちます。

レコードは 2 つの方法で失われます: ストレージがいっぱいになると最も古いレコードがドロップされる、またはシステムがより多くのデータを受け入れることができないと最新のレコードが拒否される。

最も古いレコードがドロップされた

最も古いバッファされたレコードは、再試行回数を使い切ったとき、または `storage.total_limit_size` がいっぱいになり、新しいデータのためのスペースを確保する必要がある場合にドロップされます。

再試行制限を超えた

AWS プラグインの再試行 (`auto_retry_requests true` の場合) + 最初の 1 回の Fluent Bit 試行 + `retry_limit` 回の再試行の後に発生します。軽減するには、無制限の再試行のために OUTPUT プラグインごとに `retry_limit no_limits` を設定します。

```
[OUTPUT]
    Name                        cloudwatch_logs
    Match                       ApplicationLogs
    retry_limit                 no_limits
    auto_retry_requests         true
```

> **Important**
>
> 無制限の再試行は、再試行回数を使い切ったことによるレコードのドロップを防止しますが、`storage.total_limit_size` がいっぱいになる可能性があります。

ストレージ制限に達した (ファイルシステムのバッファリング)

設定した storage.total_limit_size でバッファリングできる時間よりも長く出力先が使用できない場合に発生します。例えば、1 MB/秒のログレートで 10 GB のバッファを使用すると、約 2.7 時間のバッファリングが可能になります。軽減するには、OUTPUT プラグインごとに storage.total_limit_size を増やし、適切な Amazon ECS タスクストレージをプロビジョニングします。

```
[OUTPUT]
    Name                        cloudwatch_logs
    Match                       ApplicationLogs
    storage.total_limit_size    10G
```

最新のレコードが拒否された

最新のレコードは、ディスク容量を使い切ったとき、または Mem_Buf_Limit のために入力が一時停止されたときにドロップされます。

ディスク容量を使い切った (ファイルシステムのバッファリング)

ディスク容量を完全を使い切ったときに発生します。Fluent Bit は新しいレコードのキューに失敗し、それらは失われます。軽減するには、すべての storage.total_limit_size 値を合計し、適切な Amazon ECS タスクストレージをプロビジョニングします。詳細については、「Amazon ECS タスクストレージの要件」を参照してください。

メモリ制限に達した (メモリバッファリング)

出力先が使用できず、メモリバッファがいっぱいになったときに発生します。一時停止した入力プラグインは、新しいレコードの受け入れを停止します。軽減するには、耐障害性を高めるために storage.type filesystem を使用するか、Mem_Buf_Limit を増やします。

データ損失を最小限にするためのベストプラクティス

データ損失を最小限にするには、次のベストプラクティスを検討してください:

- ファイルシステムのバッファリングを使用する — 停止時の耐障害性を向上させるように storage.type filesystem を設定します。
- ストレージを適切なサイズにする — ログレートと希望する復旧期間に基づいて storage.total_limit_size を計算します。

- 適切なディスクをプロビジョニングする — Amazon ECS タスクに十分なエフェメラルストレージ、Amazon EBS、または Amazon EFS があることを確認します。
- 再試行動作を設定する — `retry_limit` (再試行回数を使い切るとレコードがドロップされます) と `no_limits` (無期限に再試行されますが、ストレージがいっぱいになる可能性があります) のバランスを取ります。

マルチ送信先ログを使用して信頼性を高める

複数の送信先にログを送信すると、単一障害点がなくなります。たとえば、CloudWatch Logs で機能停止が発生した場合でも、ログは Amazon S3 に到達します。

マルチ送信先ログには、次の利点があります。Amazon S3 出力プラグインは、gzip 形式や Parquet 形式などの圧縮オプションもサポートしているため、ストレージコストを削減できます。詳細については、「Fluent Bit ドキュメント」の「S3 圧縮」を参照してください。

マルチ送信先ログには、次の利点があります。

- 冗長性 – 一方の送信先が失敗しても、ログはもう一方の送信先に到達します。
- 復旧 – 一方のシステムのギャップを他方のシステムから再構築します。
- 耐久性 – ログを Amazon S3 にアーカイブして長期保持します。
- コスト最適化 – 最近のログを CloudWatch Logs などの高速クエリサービスに短期間保存し、すべてのログを低コストの Amazon S3 ストレージにアーカイブして長期保持します。

次の Fluent Bit 設定は、CloudWatch Logs と Amazon S3 の両方にログを送信します。

```
[OUTPUT]
    Name cloudwatch_logs
    Match *
    region us-west-2
    log_group_name /aws/ecs/my-app
    log_stream_name $(ecs_task_id)
    workers 2
    retry_limit 15

[OUTPUT]
    Name s3
    Match *
    bucket my-logs-bucket
    region us-west-2
```

```
    total_file_size 100M
    s3_key_format /fluent-bit-logs/$(ecs_task_id)/%Y%m%d/%H/%M/$UUID
    upload_timeout 10m
    # Maximum disk space for buffered data for this output
    storage.total_limit_size 5G
```

どちらの出力も同じ `Match *` パターンを使用するため、すべてのレコードは両方の送信先に個別に送信されます。どちらか一方の送信先が停止しても、ログは引き続き他の送信先に流れ続け、失敗したフラッシュは後で再試行できるようにファイルシステムバッファに蓄積されます。

tail 入力プラグインでファイルベースのログを使用する

ログ損失が重大な懸念事項となる高スループットのシナリオでは、別の方法として、アプリケーションがディスク上のファイルにログを書き込み、`tail` 入力プラグインを使用してそれらを読み取るように Fluent Bit を設定します。このアプローチは、Docker ログドライバーレイヤーを完全にバイパスします。

tail プラグインを使用したファイルベースのログ記録には、次の利点があります。

- オフセット追跡 – tail プラグインはデータベースファイルにファイルオフセットを保存できるため (DB オプションを使用)、Fluent Bit 再起動した際に耐久性があります。これにより、コンテナの再起動時にログが消失するのを防ぐことができます。
- 入力レベルのバッファリング – `Mem_Buf_Limit` を使用して入力プラグインでメモリバッファ制限を直接設定できるため、メモリ使用量をより詳細に制御できます。
- Docker のオーバーヘッドを回避 – Docker のログバッファを経由することなく、ログはファイルから Fluent Bit に直接送信されます。

このアプローチを使用するには、アプリケーションが `stdout` ではなくファイルにログを書き込む必要があります。アプリケーションコンテナと Fluent Bit コンテナの両方が、ログファイルが保存されている共有ボリュームをマウントします。

次の例は、ベストプラクティスを使用した tail 入力設定を示しています。

```
[INPUT]
    Name tail
    # File path or glob pattern to tail
    Path /var/log/app.log
    # Database file for storing file offsets (enables resuming after restart)
    DB /var/log/flb_tail.db
```

```
    # when true, controls that only fluent-bit will access the database (improves
performance)
    DB.locking true
    # Skip long lines instead of skipping the entire file
    Skip_Long_Lines On
    # How often (in seconds) to check for new files matching the glob pattern
    Refresh_Interval 10
    # Extra seconds to monitor a file after rotation to account for pending flush
    Rotate_Wait 30
    # Maximum size of the buffer for a single line
    Buffer_Max_Size 10MB
    # Initial allocation size for reading file data
    Buffer_Chunk_Size 1MB
    # Maximum memory buffer size (tail pauses when full)
    Mem_Buf_Limit 75MB
```

tail 入力プラグインを使用する場合は、次の点を考慮してください。

- ディスクの枯渇を防ぐために、アプリケーションログのログローテーションを実装します。基盤となるボリュームメトリクスをモニタリングしてパフォーマンスを測定します。
- ログ形式に基づいて、Ignore_Older、Read_from_Head および複数行パーサーなどの設定を検討してください。

詳細については、「Fluent Bit ドキュメント」の「Tail」を参照してください。ベストプラクティスについては、「Fluent Bit トラブルシューティングガイド」の AWS の「ベストプラクティスを使用した Tail 設定」を参照してください。

FireLens に直接ログ記録する

awsfirelens ログドライバーがタスク定義で指定されている場合、Amazon ECS コンテナエージェントは次の環境変数をコンテナに挿入します。

FLUENT_HOST

FireLens コンテナに割り当てられた IP アドレス。

> **Note**
>
> bridge ネットワークモードの EC2 を使用している場合、FireLens ログルーターコンテナ (コンテナ定義に firelensConfiguration オブジェクトがあるコンテナ) を再起動した後で、アプリケーションコンテナの FLUENT_HOST 環境変数が不正確になる可能性

があります。これは、FLUENT_HOST が動的 IP アドレスであり、再起動後に変更される場合があるためです。アプリケーションコンテナから FLUENT_HOST IP アドレスへの直接的なロギングは、アドレスの変更後に失敗することがあります。個々のコンテナの再起動に関する詳細については、「コンテナ再起動ポリシーを使用して Amazon ECS タスク内の個々のコンテナを再起動する」を参照してください。

`FLUENT_PORT`

Fluent Forward プロトコルがリッスンしているポート。

これらの環境変数を使用して、`stdout` に書き込む代わりに Fluent Forward プロトコルを使用してアプリケーションコードから Fluent Bit ログルーターに直接ログを記録できます。このアプローチでは、Docker ログドライバーレイヤーをバイパスします。これにより、次の利点が得られます。

- 低レイテンシー – Docker のログインフラストラクチャを経由することなく、ログは Fluent Bit に直接送信されます。
- 構造化ログ – JSON エンコーディングのオーバーヘッドなしで、構造化ログデータをネイティブに送信します。
- より良い制御 – アプリケーションは独自のバッファリングとエラー処理ロジックを実装できます。

次の Fluent ロガーライブラリは Fluent Forward プロトコルをサポートし、ログを Fluent Bit に直接送信するために使用できます。

- Go – fluent-logger-golang
- Python – fluent-logger-python
- Java – fluent-logger-java
- Node.js – fluent-logger-node
- Ruby – fluent-logger-ruby

Docker バッファ制限を設定する

タスク定義を作成する際は、`log-driver-buffer-limit` で値を指定することで、メモリにバッファリングされるログの行数を指定できます。これにより、Docker と Fluent Bit の間のバッファが

制御されます。詳細については、Docker ドキュメントの「Fluentd ロギングドライバー」を参照してください。

このオプションは、スループットが高いために Docker がバッファメモリを使い果たし、新しいメッセージを追加するためにバッファメッセージを破棄する可能性がある場合に使用します。

このオプションを使用する場合は、次の点を考慮してください。

- このオプションは、EC2 およびプラットフォームバージョン 1.4.0 以降の Fargate でサポートされています。
- このオプションは、logDriver が awsfirelens に設定されている場合にのみ有効です。
- デフォルトのバッファ制限は 1048576 (ログの行数) です。
- バッファ制限は、0 ログ行以上、536870912 ログ行未満である必要があります。
- このバッファに使用されるメモリの最大量は、各ログ行のサイズとバッファのサイズ積です。たとえば、アプリケーションのログ行が平均 2 KiB の場合、4096 のバッファ制限では最大 8 MiB を使用します。タスクレベルで割り当てられるメモリの合計量は、ログドライバーのメモリバッファに加えて、すべてのコンテナに割り当てられたメモリ量よりも大きくなければなりません。

次のタスク定義は、log-driver-buffer-limit を設定する方法を示しています。

```
{
    "containerDefinitions": [
        {
            "name": "my_service_log_router",
            "image": "public.ecr.aws/aws-observability/aws-for-fluent-bit:3",
            "cpu": 0,
            "memoryReservation": 51,
            "essential": true,
            "firelensConfiguration": {
                "type": "fluentbit"
            }
        },
        {
            "essential": true,
            "image": "public.ecr.aws/docker/library/httpd:latest",
            "name": "app",
            "logConfiguration": {
                "logDriver": "awsfirelens",
                "options": {
                    "Name": "firehose",
```

```
                    "region": "us-west-2",
                    "delivery_stream": "my-stream",
                    "log-driver-buffer-limit": "52428800"
                }
            },
            "dependsOn": [
                {
                    "containerName": "my_service_log_router",
                    "condition": "START"
                }
            ],
            "memoryReservation": 100
        }
    ]
}
```

Amazon ECS の Fluent Bit イメージリポジトリの AWS

AWS は、CloudWatch Logs と Firehose の両方のプラグインに Fluent Bit イメージを提供します。Fluent Bit は、リソース使用率が Fluentd よりも低いため、ログルーターとして使用することをお勧めします。詳細については、「CloudWatch Logs for Fluent Bit」および「Amazon Kinesis Firehose for Fluent Bit」を参照してください。

AWS for Fluent Bit イメージは、Amazon ECR Public Gallery と Amazon ECR リポジトリの両方の Amazon ECR で利用が可能で、高可用性を実現できます。

Amazon ECR Public Gallery

AWS for Fluent Bit イメージは Amazon ECR Public Gallery で利用できます。これはパブリックリポジトリであり、すべての AWS リージョン リージョンから使用できるため、AWS for Fluent Bit イメージのダウンロード先として推奨されます 詳細については、Amazon ECR Public Gallery の「aws-for-fluent-bit」を参照してください。

Linux

Amazon ECR Public Gallery の AWS for Fluent Bit イメージは、ARM64 または x86-64 アーキテクチャの Amazon Linux オペレーティングシステムをサポートしています。

目的のイメージタグを使用してリポジトリ URL を指定することにより、Amazon ECR Public Gallery からAWS for Fluent Bit イメージをプルできます。利用可能な画像タグは、Amazon ECR Public Gallery の [Image tags (画像タグ)] タブにあります。

Docker CLI で使用する構文を以下に示します。

```
docker pull public.ecr.aws/aws-observability/aws-for-fluent-bit:tag
```

たとえば、次の Docker CLI コマンドを使用すれば、Fluent Bit リリースの AWS の「3.x」ファミリの最新イメージをプルできます。

```
docker pull public.ecr.aws/aws-observability/aws-for-fluent-bit:3
```

> **Note**
>
> 認証されていないプルは許可されますが、認証されたプルよりもレート制限が低くなります。プルする前に、次のコマンドを使用して AWS アカウントの使用を認証します。
>
> ```
> aws ecr-public get-login-password --region us-east-1 | docker login --username
>  AWS --password-stdin public.ecr.aws
> ```

AWS 3.0.0 向けの Fluent Bit

Fluent Bit バージョン `2.x` の既存 AWS に加えて、Fluent Bit の AWS は新しいメジャーバージョン `3.x` をサポートします。新しいメジャーバージョンには、イメージを Amazon Linux 2 から Amazon Linux 2023 にアップグレードし、Fluent Bit バージョン `1.9.10` を `4.1.1` にアップグレードすることが含まれます。詳細は GitHub の「AWS for Fluent Bit repository」を参照してください。

次の例は、Fluent Bit `3.x` イメージの AWS の更新済みタグを示しています。

Fluent Bit イメージの AWS にはマルチアーキテクチャタグを使用できます。

```
docker pull public.ecr.aws/aws-observability/aws-for-fluent-bit:3
```

Windows

Amazon ECR Public Gallery の AWS for Fluent Bit イメージは、次のオペレーティングシステムを備えた `AMD64` アーキテクチャをサポートしています。

- Windows Server 2022 Full
- Windows Server 2022 Core
- Windows Server 2019 Full

- Windows Server 2019 Core

AWS Fargate の Windows コンテナは FireLens をサポートしません。

目的のイメージタグを使用してリポジトリ URL を指定することにより、Amazon ECR Public Gallery からAWS for Fluent Bit イメージをプルできます。利用可能な画像タグは、Amazon ECR Public Gallery の [Image tags (画像タグ)] タブにあります。

Docker CLI で使用する構文を以下に示します。

```
docker pull public.ecr.aws/aws-observability/aws-for-fluent-bit:tag
```

例えば、次の Docker CLI コマンドを使用して、AWS for Fluent Bit イメージの最新安定版をプルできます。

```
docker pull public.ecr.aws/aws-observability/aws-for-fluent-bit:windowsservercore-
stable
```

> **Note**
>
> 認証されていないプルは許可されますが、認証されたプルよりもレート制限が低くなります。プルする前に、次のコマンドを使用して AWS アカウントの使用を認証します。
>
> ```
> aws ecr-public get-login-password --region us-east-1 | docker login --username
>  AWS --password-stdin public.ecr.aws
> ```

Amazon ECR

AWS for Fluent Bit イメージは、Amazon ECR で高可用性を活用できます。次のコマンドを使用して、イメージ URI の取得および特定の AWS リージョンでイメージの可用性を確立できます。

Linux

最新の安定している AWS for Fluent Bit イメージの URI は、次のコマンドを使用して取得できます。

```
aws ssm get-parameters \
      --names /aws/service/aws-for-fluent-bit/stable \
```

```
      --region us-east-1
```

次のコマンドを使用して Systems Manager パラメータストアのパラメータをクエリすると、AWS for Fluent Bit イメージのすべてのバージョンを一覧表示できます。

```
aws ssm get-parameters-by-path \
      --path /aws/service/aws-for-fluent-bit \
      --region us-east-1
```

AWS for Fluent Bit イメージの最新安定版は、Systems Manager パラメータストア名を参照することにより、CloudFormation テンプレート内に見つかります。次にの例を示します。

```
Parameters:
  FireLensImage:
    Description: Fluent Bit image for the FireLens Container
    Type: AWS::SSM::Parameter::Value<String>
    Default: /aws/service/aws-for-fluent-bit/stable
```

> **Note**
>
> コマンドが失敗するか、出力がない場合、イメージはコマンドが呼び出される AWS リージョンでは使用できません。

Windows

最新の安定している AWS for Fluent Bit イメージの URI は、次のコマンドを使用して取得できます。

```
aws ssm get-parameters \
      --names /aws/service/aws-for-fluent-bit/windowsservercore-stable \
      --region us-east-1
```

次のコマンドを使用して Systems Manager パラメータストアのパラメータをクエリすると、AWS for Fluent Bit イメージのすべてのバージョンを一覧表示できます。

```
aws ssm get-parameters-by-path \
      --path /aws/service/aws-for-fluent-bit/windowsservercore \
      --region us-east-1
```

最新の安定している AWS for Fluent Bit イメージは、Systems Manager パラメータストア名を参照することにより、CloudFormation テンプレートで参照できます。次にの例を示します。

```
Parameters:
  FireLensImage:
    Description: Fluent Bit image for the FireLens Container
    Type: AWS::SSM::Parameter::Value<String>
    Default: /aws/service/aws-for-fluent-bit/windowsservercore-stable
```

Amazon ECS タスク定義の例: FireLens にログをルーティングする

FireLens でカスタムログルーティングを使用するには、タスク定義で以下を指定する必要があります。

- FireLens 設定を含むログルーターコンテナ。コンテナは `essential` とマークすることが推奨されます。
- `awsfirelens` ログドライバーを指定するログ設定を含む 1 つ以上のアプリケーションコンテナ。
- タスクでログをルーティングするために必要なアクセス許可を含むタスク IAM ロールの、Amazon リソースネーム (ARN)。

AWS マネジメントコンソール を使用して新しいタスク定義を作成する場合、ログルーターコンテナを簡単に追加できる FireLens 統合セクションがあります。詳細については、「コンソールを使用した Amazon ECS タスク定義の作成」を参照してください。

Amazon ECS はログ設定を変換し、Fluentd または Fluent Bit 出力設定を生成します。出力設定は、`/fluent-bit/etc/fluent-bit.conf` (Fluent Bit) および `/fluentd/etc/fluent.conf` (Fluentd) のログルーティングコンテナにマウントされます。

> **Important**
>
> FireLens は、ポート 24224 でリッスンします。したがって、FireLens ログルーターがタスクの外に到達できないようにするには、タスクが使用するセキュリティグループで、ポート 24224 での入力トラフィックを許可してはなりません。`awsvpc` ネットワークモードを使用する場合、このセキュリティグループは、タスクに関連付けられたセキュリティグループです。`host` ネットワークモードを使用する場合、これはタスクをホストする Amazon EC2 インスタンスに関連付けられているセキュリティグループです。`bridge` ネットワークモードを使用するタスクの場合、ポート 24224 を使用するポートマッピングを作成しないでください。

デフォルトでは、Amazon ECS は、ログのソースを識別するのに役立つ追加のフィールドをログエントリに追加します。

- `ecs_cluster` - タスクが所属するクラスターの名前。
- `ecs_task_arn` – コンテナが属しているタスクの完全な Amazon リソースネーム (ARN)。
- `ecs_task_definition` - タスクが使用しているタスク定義名とリビジョン。
- `ec2_instance_id` - コンテナがホストされている Amazon EC2 インスタンス ID。このフィールドは、EC2 起動タイプを使用するタスクでのみ有効です。

メタデータが必要ない場合は、`enable-ecs-log-metadata` を `false` に設定できます。

以下のタスク定義の例では、Fluent Bit を使用してログを CloudWatch Logs にルーティングするログルーターコンテナを定義しています。また、これによりアプリケーションコンテナを定義します。このコンテナでは、ログ設定を使用してログを Amazon Data Firehose にルーティングし、イベントのバッファリングに使用されるメモリを 2MiB に設定します。

> **Note**
>
> タスク定義のその他の例については、GitHub の Amazon ECS FireLens の例を参照してください。

```
{
  "family": "firelens-example-firehose",
  "taskRoleArn": "arn:aws:iam::123456789012:role/ecs_task_iam_role",
  "containerDefinitions": [
    {
            "name": "log_router",
            "image": "public.ecr.aws/aws-observability/aws-for-fluent-bit:3",
            "cpu": 0,
            "memoryReservation": 51,
            "portMappings": [],
            "essential": true,
            "environment": [],
            "mountPoints": [],
            "volumesFrom": [],
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
```

```
                    "awslogs-group": "/ecs/ecs-aws-firelens-sidecar-container",
                    "mode": "non-blocking",
                    "awslogs-create-group": "true",
                    "max-buffer-size": "25m",
                    "awslogs-region": "us-east-1",
                    "awslogs-stream-prefix": "firelens"
                },
                "secretOptions": []
            },
            "systemControls": [],
            "firelensConfiguration": {
                "type": "fluentbit"
            }
        },
    {
      "essential": true,
      "image": "public.ecr.aws/docker/library/httpd:latest",
      "name": "app",
      "logConfiguration": {
        "logDriver": "awsfirelens",
        "options": {
          "Name": "firehose",
          "region": "us-west-2",
          "delivery_stream": "my-stream",
          "log-driver-buffer-limit": "1048576"
        }
      },
      "memoryReservation": 100
    }
  ]
}
```

`logConfiguration` オブジェクトのオプションとして指定されたキーバリューペアは、Fluentd または Fluent Bit 出力設定の生成に使用されます。Fluent Bit 出力定義のコード例は次のとおりです。

```
[OUTPUT]
    Name   firehose
    Match  app-firelens*
    region us-west-2
    delivery_stream my-stream
```

> **Note**
>
> FireLens は `match` 設定を管理します。タスク定義では `match` 設定を指定しません。

カスタム設定ファイルを使用する

カスタム設定ファイルを指定できます。設定ファイルの形式は、使用しているログルーターでネイティブな形式を使用します。詳細については、「Fluentd 設定ファイルの構文」および「YAML 設定」を参照してください。

カスタム設定ファイルでは、`bridge` または `awsvpc` ネットワークモードを使用するタスクについて、TCP 経由で Fluentd または Fluent Bit の転送入力を設定しないでください。入力設定は FireLens により追加されています。

カスタム設定ファイルを指定するには、FireLens 設定に次のオプションを含める必要があります。

`config-file-type`

カスタム設定ファイルのソースの場所。使用できるオプションは、`s3` または `file` です。

> **Note**
>
> AWS Fargate でホストされるタスクは、`file` 設定ファイルタイプのみをサポートします。ただし、Fluent Bit init コンテナの AWS を使用して、AWS Fargate の Amazon S3 でホストされている設定ファイルを使用できます。詳細については、GitHub の「Init process for Fluent Bit on ECS, multi-config support」を参照してください。

`config-file-value`

カスタム設定ファイルのソース。`s3` 設定ファイルタイプを使用する場合、設定ファイルの値は Amazon S3 バケットとファイルの完全な ARN です。`file` 設定ファイルタイプを使用する場合、設定ファイルのこの値は、コンテナイメージ内、またはそのコンテナにマウントされているボリューム上に存在する設定ファイルへの完全パスです。

> **Important**
>
> カスタム設定ファイルを使用する場合、FireLens が使用するパスとは異なるパスを指定する必要があります。Amazon ECS は Fluent Bit に `/fluent-bit/etc/fluent-bit.conf` ファイルパスと Fluentd に `/fluentd/etc/fluent.conf` を指定します。

次の例は、カスタム設定を指定するときに必要な構文を示しています。

> **Important**
>
> Amazon S3 でホストされるカスタム設定ファイルを指定するには、適切なアクセス許可を持つタスク実行 IAM ロールが作成されている必要があります。

次に、カスタム設定を指定する際に必要な構文を示します。

```
{
  "containerDefinitions": [
    {
      "essential": true,
      "image": "906394416424.dkr.ecr.us-west-2.amazonaws.com/aws-for-fluent-bit:3",
      "name": "log_router",
      "firelensConfiguration": {
        "type": "fluentbit",
        "options": {
          "config-file-type": "s3 | file",
          "config-file-value": "arn:aws:s3:::amzn-s3-demo-bucket/fluent.conf
 | filepath"
        }
      }
    }
  ]
}
```

> **Note**
>
> AWS Fargate でホストされるタスクは、`file` 設定ファイルタイプのみをサポートします。ただし、Fluent Bit init コンテナの AWS を使用して、AWS Fargate の Amazon S3 でホス

トされている設定ファイルを使用できます。詳細については、GitHub の「Init process for Fluent Bit on ECS, multi-config support」を参照してください。

Amazon ECS での AWS 以外のコンテナイメージの使用

プライベートレジストリを使用して認証情報を AWS Secrets Manager に保存し、タスク定義内で参照します。これは、タスク定義で認証が必要な AWS の外部にあるプライベートレジストリーに存在するコンテナイメージを参照できます。この機能は、Amazon ECS Anywhere を使用して Fargate、Amazon EC2 インスタンス、および外部インスタンスでホストされるタスクでサポートされています。

Important

タスク定義で、Amazon ECR に保存されたイメージを参照している場合、このトピックは適用されません。詳細については、Amazon Elastic Container Registry ユーザーガイドの「Amazon ECS で Amazon ECR イメージを使用する」を参照してください。

Amazon EC2 インスタンスでホストされるタスクの場合は、この機能のためにコンテナエージェントのバージョン `1.19.0` 以降が必要です。ただし、最新のコンテナエージェントのバージョンを使用することをお勧めします。エージェントのバージョンの確認方法と最新バージョンへの更新方法については、「Amazon ECS コンテナエージェントをアップデートする」を参照してください。

Fargate でホストされているタスクの場合、この機能にはプラットフォームバージョン `1.2.0` 以降が必要です。詳細については、「Amazon ECS 向け Fargate プラットフォームバージョン」を参照してください。

コンテナの定義内で、作成したシークレットの詳細 で `repositoryCredentials` オブジェクトを指定します。参照されるシークレットは、タスクが使用するものと異なる AWS リージョン、または異なるアカウントに置くことができます。

Note

Amazon ECS API、AWS CLI、または AWS SDK を使用しており、起動するタスクと同じ AWS リージョン にシークレットが存在する場合は、シークレットの完全な ARN または名前のどちらも使用可能です。シークレットが別のアカウントに存在する場合は、シークレッ

トの完全な ARN を指定する必要があります。AWS マネジメントコンソール を使用する場合は、シークレットの完全な ARN を常に指定する必要があります。

必要なパラメータが含まれるタスク定義のスニペットを、以下に示します。

次のパラメータを置き換えます。

- *private-repo* をプライベートリポジトリのホスト名に
- *private-image* をイメージ名に
- *arn:aws:secretsmanager:region:aws_account_id:secret:secret_name* をシークレット Amazon リソースネーム (ARN) に

```
"containerDefinitions": [
    {
        "image": "private-repo/private-image",
        "repositoryCredentials": {
            "credentialsParameter":
 "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name"
        }
    }
]
```

> **Note**
>
> プライベートレジストリ認証を有効にするもうひとつの方法として、Amazon ECS コンテナエージェントの環境変数を使用してプライベートレジストリを認証する方法があります。このメソッドは、Amazon EC2 インスタンスでホストされるタスクでのみサポートされています。詳細については、「プライベート Docker イメージ用の Amazon ECS コンテナインスタンスを設定する」を参照してください。

プライベートレジストリを使用するには

1. タスク定義にはタスク実行ロールが必要です。このロールを使用して、コンテナエージェントでコンテナイメージをプルできます。詳細については、「Amazon ECS タスク実行IAM ロール」を参照してください。

プライベートレジストリ認証を使用すると、Amazon ECS タスクは、認証情報を必要とする AWS の外部 (Docker Hub、Quay.io、独自のプライベートレジストリなど) のプライベートレジストリからコンテナイメージをプルすることができます。この機能は Secrets Manager を使用してレジストリ認証情報を安全に保存し、`repositoryCredentials` パラメータを使用してタスク定義で参照します。

プライベートレジストリ認証の設定の詳細については、「Amazon ECS での非 AWS コンテナイメージの使用」を参照してください。

プライベートレジストリ認証情報を含むシークレットにアクセスできるようにするには、以下のアクセス許可を、インラインポリシーとしてタスクの実行ロールに追加します。詳細については、「IAM ポリシーの追加と削除」を参照してください。

- `secretsmanager:GetSecretValue`— Secrets Manager からプライベートレジストリ認証情報を取得するために必要です。
- `kms:Decrypt` – シークレットでカスタムの KMS キーを使用し、デフォルトのキーを使用しない場合にのみ必須です。カスタムキーの Amazon リソースネーム (ARN) は、リソースとして追加する必要があります。

次の例では、インラインポリシーによりアクセス許可を追加しています。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "secretsmanager:GetSecretValue"
            ],
            "Resource": [
                "arn:aws:secretsmanager:us-
east-1:111122223333:secret:secret_name",
                "arn:aws:kms:us-east-1:111122223333:key/key_id"
            ]
        }
    ]
```

```
}
```

2. プライベートレジストリ認証情報のシークレットを作成するには、AWS Secrets Manager を使用します。シークレットの作成方法について、詳細は「AWS Secrets Manager ユーザーガイド」の「AWS Secrets Manager シークレットを作成する」を参照してください。

   以下の形式でプライベートレジストリの認証情報を入力します。

   ```
   {
     "username" : "privateRegistryUsername",
     "password" : "privateRegistryPassword"
   }
   ```

3. タスク定義を登録する 詳細については、「the section called “コンソールを使用したタスク定義の作成”」を参照してください。

コンテナ再起動ポリシーを使用して Amazon ECS タスク内の個々のコンテナを再起動する

タスク定義で定義されている必須のコンテナと必須ではないコンテナそれぞれに再起動ポリシーを有効にして、一時的な障害をより迅速に克服し、タスクの可用性を維持することができます。コンテナの再起動ポリシーを有効にすると、コンテナが終了した場合、Amazon ECS はタスクを置き換えることなくコンテナを再起動できます。

再起動ポリシーは、デフォルトではコンテナに対して有効になっていません。コンテナの再起動ポリシーを有効にすると、コンテナを再起動しない終了コードを指定できます。これらは、終了コード `0` のように再起動を必要としない成功を示す終了コードである可能性があります。再起動を試みる前に、コンテナが正常に実行する必要のある時間を指定することもできます。これらのパラメータの詳細については、「再起動ポリシー」を参照してください。これらの値を指定するタスク定義の例については、「Amazon ECS タスク定義でコンテナ再起動ポリシーを指定する」を参照してください。

Amazon ECS タスクメタデータエンドポイントまたは CloudWatch Container Insights を使用して、コンテナが再起動した回数をモニタリングできます。タスクメタデータエンドポイントの詳細については、「Amazon ECS タスクメタデータエンドポイントバージョン 4」および「Fargate のタスク用の Amazon ECS タスクメタデータエンドポイントバージョン 4」を参照してください。Amazon ECS Container Insights メトリクスの一覧については、「Amazon CloudWatch ユーザーガイド」の「Amazon ECS Container Insights メトリクス」を参照してください。

コンテナ再起動ポリシーは、Fargate、Amazon EC2 インスタンスに加えて、Amazon ECS Anywhere を使用する外部インスタンスでホストされるタスクでサポートされています。

考慮事項

コンテナの再起動ポリシーを有効にする前に、次の点を考慮してください。

- Fargate の Windows コンテナでは、再起動ポリシーはサポートされません。
- Amazon EC2 インスタンスでホストされるタスクの場合は、この機能のためにコンテナエージェントのバージョン `1.86.0` 以降が必要です。ただし、最新のコンテナエージェントのバージョンを使用することをお勧めします。エージェントのバージョンの確認方法と最新バージョンへの更新方法については、「Amazon ECS コンテナエージェントをアップデートする」を参照してください。
- `bridge` ネットワークモードの EC2 を使用している場合、FireLens ログルーターコンテナ (コンテナ定義に `firelensConfiguration` オブジェクトがあるコンテナ) を再起動した後で、アプリケーションコンテナの `FLUENT_HOST` 環境変数が不正確になる可能性があります。これは、`FLUENT_HOST` が動的 IP アドレスであり、再起動後に変更される場合があるためです。アプリケーションコンテナから `FLUENT_HOST` IP アドレスへの直接的なロギングは、アドレスの変更後に失敗することがあります。`FLUENT_HOST` の詳細については、「高スループットの Amazon ECS ログの設定」を参照してください。
- Amazon ECS エージェントは、コンテナの再起動ポリシーを処理します。予期しない理由で Amazon ECS エージェントが失敗するか実行されなくなったりすると、コンテナは再起動されません。
- ポリシーで定義された再起動試行期間は、Amazon ECS がコンテナを再起動するまでにコンテナが実行される必要のある期間 (秒単位) を決定します。

Amazon ECS タスク定義でコンテナ再起動ポリシーを指定する

タスク定義でコンテナの再起動ポリシーを指定するには、コンテナ定義内で `restartPolicy` オブジェクトを指定します。`restartPolicy` オブジェクトの詳細については、「再起動ポリシー」を参照してください。

以下は、Web サーバーをセットアップするために、Fargate で Linux コンテナを使用するタスク定義です。コンテナ定義には、コンテナの再起動ポリシーを有効にするために `enabled` が `true` に設定された `restartPolicy` オブジェクトが含まれています。コンテナは再起動する前に 180 秒間実行する必要があります。成功を示す終了コード `0` で終了した場合、コンテナは再起動しません。

```
{
  "containerDefinitions": [
    {
      "command": [
        "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample App</title>
 <style>body {margin-top: 40px; background-color: #333;} </style> </head><body>
 <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1>
 <h2>Congratulations!</h2> <p>Your application is now running on a container in Amazon
 ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html && httpd-
foreground\""
      ],
      "entryPoint": ["sh", "-c"],
      "essential": true,
      "image": "public.ecr.aws/docker/library/httpd:2.4",
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "/ecs/fargate-task-definition",
          "awslogs-region": "us-east-1",
          "awslogs-stream-prefix": "ecs"
        }
      },
      "name": "sample-fargate-app",
      "portMappings": [
        {
          "containerPort": 80,
          "hostPort": 80,
          "protocol": "tcp"
        }
      ],
      "restartPolicy": {
        "enabled": true,
        "ignoredExitCodes": [0],
        "restartAttemptPeriod": 180
      }
    }
  ],
  "cpu": "256",
  "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
  "family": "fargate-task-definition",
  "memory": "512",
  "networkMode": "awsvpc",
  "runtimePlatform": {
```

```
    "operatingSystemFamily": "LINUX"
  },
  "requiresCompatibilities": ["FARGATE"]
}
```

コンテナ定義で `restartPolicy` オブジェクトを含むタスク定義を登録した後、そのタスク定義でタスクを実行するか、サービスを作成することができます。詳細については、「Amazon ECS タスクとしてのアプリケーションの実行」および「Amazon ECS のローリング更新デプロイの作成」を参照してください。

Amazon ECS コンテナに機密データを渡す

認証情報などの機密データを、コンテナ内のデータベースなどに安全に渡すことができます。

API キーやデータベース認証情報などのシークレットは、アプリケーションが他のシステムにアクセスするためによく使用されます。多くの場合、ユーザー名とパスワード、証明書、または API キーで構成されます。これらのシークレットへのアクセスは、IAM を使用する特定の IAM プリンシパルに限定され、またランタイムにコンテナに挿入される必要があります。

シークレットは、AWS Secrets Manager または Amazon EC2 Systems Manager Parameter Store からコンテナにシームレスに挿入できます。これらのシークレットは、タスク内で以下のいずれかとして参照できます。

1. `secrets` コンテナ定義パラメータを使用する環境変数として参照されます。
2. ロギングプラットフォームが認証を必要とする場合、`secretOptions` として参照されます。詳細については、「ログの設定オプション」を参照してください。
3. コンテナの取得元のレジストリーが認証を必要とする場合、これらは、`repositoryCredentials` コンテナ定義パラメータを使用するイメージによって取得されるシークレットとして参照されます。Amazon ECR Public Gallery からイメージを取得するときは、この方法を使用してください。詳細については、「タスクのプライベートレジストリの認証」を参照してください。

シークレットの管理を設定するときは、次を行うことをお勧めします。

シークレットマテリアルの保存に AWS Secrets Manager または AWS Systems Manager Parameter Store を使用する

API キー、データベース認証情報、およびその他のシークレットマテリアルは、Secrets Manager または暗号化されたパラメータとして Systems Manager Parameter Store に安全に保存する必要があ

ります。これらのサービスは、どちらも AWS KMS を使用して機密データを暗号化する管理されたキー値ストアである点で似ています。ただし、Secrets Manager には、シークレットを自動的にローテーションし、ランダムなシークレットを生成し、アカウント間でシークレットを共有する機能も含まれています。これらの機能を利用するには、Secrets Manager を使用します。それ以外の場合は、Systems Manager パラメータストアで暗号化されたパラメータを使用します。

> **Important**
>
> シークレットが変更された場合は、新しいデプロイを強制するか、新しいタスクを起動して最新のシークレット値を取得する必要があります。詳細については、以下の各トピックを参照してください。
>
> - タスク - タスクを停止してから開始します。詳細については、「Amazon ECS タスクの停止」および「Amazon ECS タスクとしてのアプリケーションの実行」を参照してください。
> - サービス - サービスを更新し、[新しいデプロイの強制] オプションを使用します。詳細については、「Amazon ECS サービスを更新する」を参照してください。

暗号化された Amazon S3 バケットからデータを取得する

暗号化された Amazon S3 バケットにシークレットを保存し、タスクロールを使用してそのシークレットへのアクセスを制限する必要があります。こうすることで、`docker inspect` の実行時に環境変数の値が誤ってログに漏れて公開されるのを防ぐことができます。これを行う場合、Amazon S3 バケットからシークレットを読み取るようにアプリケーションを作成する必要があります。手順については、「Amazon S3 バケット向けのサーバー側のデフォルトの暗号化動作の設定」を参照してください。

サイドカーコンテナを使用してシークレットをボリュームにマウントする

環境変数はデータ漏えいのリスクが高いため、AWS Secrets Manager からシークレットを読みとり、共有ボリュームに書き込むサイドカーコンテナを実行する必要があります。このコンテナは、Amazon ECS コンテナの順序付けを使用することでアプリケーションコンテナより先に実行して終了できます。これを行うと、アプリケーションコンテナでシークレットが書き込まれたボリュームが後からマウントされます。Amazon S3 バケットメソッドと同様に、共有ボリュームからシークレットを読み取るようにアプリケーションを作成する必要があります。ボリュームの範囲はタスクに限定されるため、タスクが停止するとボリュームは自動的に削除されます。例については、Task-def.json プロジェクトを参照してください。

Amazon EC2 では、シークレットが書き込まれるボリュームは、AWS KMS カスタマーマネージドキーで暗号化できます。AWS Fargate では、ボリュームストレージはサービスマネージドキーを使用して自動的に暗号化されます。

個々の環境変数を Amazon ECS コンテナに渡す

> **Important**
>
> 機密データは、AWS Secrets Manager secrets または AWS Systems Manager Parameter Store のパラメータに保存することをお勧めします。詳細については、「Amazon ECS コンテナに機密データを渡す」を参照してください。
> タスク定義で指定された環境変数は、許可されたすべてのユーザーとロールが、タスク定義の `DescribeTaskDefinition` アクションを読み取ることができます。

環境変数は、以下の方法でコンテナに渡すことができます。

- `environment` コンテナ定義パラメータを個別に使用します。これは、docker container run の `--env` オプションにマッピングされます。
- `environmentFiles` コンテナ定義パラメータを使用して、環境変数を含む 1 つ以上のファイルを一括で一覧表示します。ファイルは、Amazon S3 でホストされている必要があります。これは、docker run の `--env-file` オプションにマッピングされます。

以下は、個々の環境変数の指定方法を示すタスク定義のスニペットです。

```
{
    "family": "",
    "containerDefinitions": [
        {
            "name": "",
            "image": "",
            ...
            "environment": [
                {
                    "name": "variable",
                    "value": "value"
                }
            ],
            ...
```

```
        }
    ],
    ...
}
```

環境変数を Amazon ECS コンテナに渡す

> **Important**
>
> 機密データは、AWS Secrets Manager secrets または AWS Systems Manager Parameter Store のパラメータに保存することをお勧めします。詳細については、「Amazon ECS コンテナに機密データを渡す」を参照してください。
> 環境変数ファイルは Amazon S3 のオブジェクトのため、Amazon S3 のセキュリティに関するすべての考慮事項が適用されます。
> Fargate の Windows コンテナと Windows コンテナでは、`environmentFiles` パラメータを使用できません。

環境変数ファイルを作成して Amazon S3 に保存し、環境変数をコンテナに渡すことができます。

ファイルに環境変数を指定することで、環境変数を一括で挿入できます。コンテナ定義内で、環境変数ファイルを含む `environmentFiles` バケットのリストを使用して Amazon S3 オブジェクトを指定します。

Amazon ECS は環境変数にサイズ制限を適用しませんが、環境変数ファイルが大きいとディスク容量が一杯になってしまう可能性があります。環境変数ファイルを使用する各タスクは、ファイルのコピーをディスクにダウンロードします。Amazon ECS は、タスクのクリーンアップの一環としてファイルを削除します。

サポートされている環境変数については、「高度なコンテナ定義パラメータ - 環境」を参照してください。

コンテナ定義で環境変数ファイルを指定する際には、以下の点を考慮してください。

- Amazon EC2 上の Amazon ECS タスクでこの機能を使用するには、コンテナインスタンスにバージョン `1.39.0` 以降のコンテナエージェントが必要です。エージェントのバージョンの確認方法と最新バージョンへの更新方法については、「Amazon ECS コンテナエージェントをアップデートする」を参照してください。

- AWS Fargate 上のAmazon ECS タスクでは、この機能を使用するには、タスクでプラットフォームバージョン `1.4.0` 以降 (Linux) を使用する必要があります。詳細については、「Amazon ECS 向け Fargate プラットフォームバージョン」を参照してください。

  変数がオペレーティングシステムプラットフォームでサポートされていることを確認します。詳細については、「the section called “コンテナ定義”」および「the section called “その他のタスク定義パラメータ”」を参照してください。
- ファイルには、ファイル拡張子 `.env` と UTF-8 エンコーディングを使用する必要があります。
- タスク実行ロールは、Amazon S3 の追加アクセス許可でこの機能を使用するのに必要です。これにより、コンテナエージェントは Amazon S3 から環境変数ファイルをプルできます。詳細については、「Amazon ECS タスク実行IAM ロール」を参照してください。
- 1 つのタスク定義につき 10 ファイルという制限があります。
- 環境ファイルの各行には、`VARIABLE=VALUE` 形式で環境変数を含む必要があります。スペースまたは引用符は、Amazon ECS ファイルの値の一部として含まれます。# で始まる行はコメントとして扱われ、無視されます。環境変数ファイルの構文の詳細については、Docker ドキュメントの「Set environment variables (-e, --env, --env-file)」を参照してください。

  次に、適切な構文を示します。

  ```
  #This is a comment and will be ignored
  VARIABLE=VALUE
  ENVIRONMENT=PRODUCTION
  ```
- コンテナ定義に `environment` パラメータを使用して環境変数が指定されている場合は、環境ファイルに含まれる変数よりも優先されます。
- 同じ変数を含む複数の環境ファイルが指定されている場合、それらのファイルは入力順に処理されます。これは、変数の最初の値が使用され、重複する変数の後続の値は無視されることを意味します。一意の変数名を使用することをお勧めします。
- 環境ファイルがコンテナをオーバーライドするように指定されている場合、そのファイルが適用されます。さらに、コンテナ定義で指定されているその他の環境ファイルは、すべて無視されます。
- Fargate には次のルールが適用されます。
  - このファイルはネイティブの Docker env ファイルと同様に処理されます。
  - 空白で Amazon S3 に保存されている環境変数を参照するコンテナ定義は、コンテナに表示されません。
  - シェルエスケープ処理はサポートされていません。
  - コンテナのエントリポイントが `VARIABLE` 値を解釈します。

例

以下は、環境変数ファイルの指定方法を示すタスク定義のスニペットを示します。

```
{
    "family": "",
    "containerDefinitions": [
        {
            "name": "",
            "image": "",
            ...
            "environmentFiles": [
                {
                    "value": "arn:aws:s3:::amzn-s3-demo-
bucket/envfile_object_name.env",
                    "type": "s3"
                }
            ],
            ...
        }
    ],
    ...
}
```

Amazon ECS で Secrets Manager シークレットをプログラムで伝達する

アプリケーション内でプレーンテキストの機密情報をハードコーディングする代わりに、Secrets Manager を使用して機密データを保存することができます。

機密データの取得にはこの方法が推奨されます。これにより、以後、Secrets Manager のシークレットが更新された場合には、アプリケーションが自動的に最新バージョンのシークレットを取得するようになります。

Secrets Manager でシークレットを作成します。Secrets Manager シークレットの作成後、アプリケーションコードを更新して、そのシークレットを取得します。

Secrets Manager で機密データの保護を行う前に、以下の考慮事項を確認してください。

- CreateSecret API の `SecretString` パラメータで作成されたシークレットであるテキストデータを格納するシークレットのみがサポートされます。CreateSecret API の `SecretBinary` パラメータで作成されたシークレットであるバイナリデータを格納するシークレットはサポートされていません。

- セキュリティ制御を強化するために、インターフェイス VPC エンドポイントを使用します。Secrets Manager 用に、インターフェイス VPC エンドポイントを作成する必要があります。VPC エンドポイントについて、詳細は「AWS Secrets Manager ユーザーガイド」の「VPC エンドポイントの作成」を参照してください。
- タスクで使用する VPC は、DNS 解決を使用している必要があります。
- タスク定義では、Secrets Manager の追加アクセス許可を持つタスク ロールを使用する必要があります。詳細については、「Amazon ECS タスクの IAM ロール」を参照してください。

Secrets Manager シークレットを作成する

Secrets Manager コンソールを使用して、機密データ用のシークレットを作成できます。シークレットの作成方法について、詳細は「AWS Secrets Manager ユーザーガイド」の「AWS Secrets Manager シークレットを作成する」を参照してください。

Secrets Manager シークレットをプログラムにより取得するようにアプリケーションを更新する

アプリケーションから直接 Secrets Manager API を呼び出し、シークレットを取得することができます。詳細は「AWS Secrets Manager ユーザーガイド」の「AWS Secrets Manager からのシークレットの取得」を参照してください。

AWS Secrets Manager に保存されている機密データを取得するには、AWS SDK コードサンプルコードライブラリの「AWS SDK を使用する AWS Secrets Manager のコードサンプル」を参照してください。

Amazon ECS で Systems Manager Parameter Store シークレットをプログラムで伝達する

Systems Manager Parameter Store により、シークレットの安全な保管および管理ができます。パスワード、データベース文字列、EC2 インスタンス ID、AMI ID、ライセンスコードなどのデータをアプリケーションにハードコートする代わりに、パラメータ値として保存できます。値はプレーンテキストまたは暗号化されたデータとして保存できます。

機密データの取得にはこの方法が推奨されます。これにより、以後、Secrets Manager Parameter Store のパラメータが更新された場合にアプリケーションが自動的に最新バージョンを取得するようになります。

Systems Manager Parameter Store で機密データの保護を行う前に、以下の考慮事項を確認してください。

- テキストデータを格納するシークレットのみがサポートされます。バイナリデータを格納するシークレットはサポートされません。
- セキュリティ制御を強化するために、インターフェイス VPC エンドポイントを使用します。
- タスクで使用する VPC は、DNS 解決を使用している必要があります。
- EC2 を使用するタスクでは、この機能を使用するのに Amazon ECS エージェント設定変数 `ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE=true` を使用する必要があります。コンテナインスタンスの作成時に `/etc/ecs/ecs.config` ファイルに追加するか、既存のインスタンスに追加して ECS エージェントを再起動できます。詳細については、「Amazon ECS コンテナエージェントの設定」を参照してください。
- タスク定義では、Systems Manager パラメーター ストアに対する追加のアクセス許可を持つタスク ロールを使用する必要があります。詳細については、「Amazon ECS タスクの IAM ロール」を参照してください。

パラメータを作成する

Systems Manager コンソールを使用すると、機密データ用に Systems Manager Parameter Store のパラメータを作成できます。詳細は「AWS Systems Manager ユーザーガイド」の「Systems Manager パラメータを作成する (コンソール)」または「Systems Manager パラメータを作成する (AWS CLI)」を参照してください。

Systems Manager Parameter Store のシークレットをプログラムで取得するようにアプリケーションを更新する

Systems Manager Parameter Store のパラメータに保存されている機密データを取得するには、AWS SDK コードサンプルコードライブラリの「AWS SDK を使用した Systems Manager のコードサンプル」を参照してください。

Amazon ECS 環境変数経由で Secrets Manager シークレットを伝達する

シークレットを環境変数として挿入する場合、シークレットの内容全体、シークレット内の特定の JSON キーを指定できます。これは、コンテナに公開される機密データの制御に役立ちます。シークレットのバージョニングについて、詳細は「AWS Secrets Manager ユーザーガイド」の「Secrets Manager シークレットの内容」を参照してください。

環境変数を使用して Secrets Manager シークレットをコンテナに挿入する場合は、以下を考慮する必要があります。

- 重要なデータは、コンテナが最初に開始されたときにコンテナに挿入されます。シークレットを後で更新またはローテーションすると、コンテナには更新された値が自動的に送信されなくなります。この場合は、新しいタスクを起動する必要があります。または、タスクがサービスの一部である場合は、サービスを更新し、[Force new deployment] (新しいデプロイの強制) オプションを使用して、新しいタスクの起動をサービスに強制できます。
- コンテナとコンテナログ、およびデバッグツールで実行されるアプリケーションは、環境変数にアクセスできます。
- AWS Fargate 上の Amazon ECS タスクでは、以下の点を考慮してください。
  - シークレットの内容全体を環境変数として挿入したり、ログ設定にシークレットを挿入したりするには、プラットフォームバージョン `1.3.0` 以降を使用する必要があります。詳細については、「Amazon ECS 向け Fargate プラットフォームバージョン」を参照してください。
  - 特定の JSON キーまたはシークレットのバージョンを環境変数またはログ設定に挿入するには、プラットフォームバージョン `1.4.0` 以降 (Linux) または `1.0.0` (Windows) を使用する必要があります。詳細については、「Amazon ECS 向け Fargate プラットフォームバージョン」を参照してください。
- EC2 上の Amazon ECS タスクでは、以下の点を考慮する必要があります。
  - シークレットの特定の JSON キーやバージョンを使用してシークレットを挿入するには、コンテナインスタンスにバージョン `1.37.0` 以降のコンテナエージェントが必要です。ただし、最新のコンテナエージェントのバージョンを使用することをお勧めします。エージェントのバージョンの確認と最新バージョンへの更新については、「Amazon ECS コンテナエージェントをアップデートする」を参照してください。

    シークレットの内容全体を環境変数として挿入したり、ログ設定にシークレットを挿入したりするには、コンテナインスタンスにバージョン `1.22.0` 以降のコンテナエージェントが必要です。
- インターフェイス VPC エンドポイントを使用してセキュリティコントロールを強化し、プライベートサブネットを介して Secrets Manager に接続します。Secrets Manager 用に、インターフェイス VPC エンドポイントを作成する必要があります。VPC エンドポイントについて、詳細は「AWS Secrets Manager ユーザーガイド」の「VPC エンドポイントの作成」を参照してください。Secrets Manager と Amazon VPC の使用の詳細については、「How to connect to Secrets Manager service within a Amazon VPC」を参照してください。
- `awslogs` ログドライバーを使用するように設定された Windows タスクの場合は、コンテナインスタンスで `ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE` 環境変数も設定する必要があります。次の構文を使用します。

```
<powershell>
[Environment]::SetEnvironmentVariable("ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE",
 $TRUE, "Machine")
Initialize-ECSAgent -Cluster <cluster name> -EnableTaskIAMRole -LoggingDrivers
 '["json-file","awslogs"]'
</powershell>
```

- タスク定義では、Secrets Manager の追加アクセス許可を持つタスク実行ロールを使用する必要があります。詳細については、「Amazon ECS タスク実行IAM ロール」を参照してください。

AWS Secrets Manager シークレットを作成する

Secrets Manager コンソールを使用して、機密データ用のシークレットを作成できます。詳細については、AWS Secrets Manager ユーザーガイドの「Create an AWS Secrets Manager secret」を参照してください。

コンテナ定義に環境変数を追加します。

コンテナの定義内では、以下を指定できます。

- コンテナに設定する環境変数の名前が含まれている `secrets` オブジェクト
- Secrets Manager シークレットの Amazon リソースネーム (ARN)。
- コンテナに渡す機密データが含まれている追加のパラメータ

次の例は、Secrets Manager シークレットに指定する必要がある完全な構文を示しています。

```
arn:aws:secretsmanager:region:aws_account_id:secret:secret-name:json-key:version-
stage:version-id
```

次のセクションでは、追加のパラメータについて説明します。追加のパラメータはオプションですが、これらを使用しないでデフォルト値を使用する場合は、コロン : を含める必要があります。以下の例でより詳細なコンテキストを示します。

`json-key`

キーと値のペアのキーの名前を指定します。値は設定する環境変数の値です。JSON 形式の値のみがサポートされます。JSON キーを指定しないと、シークレットの内容全体が使用されます。

`version-stage`

使用するシークレットのバージョンのステージングラベルを指定します。バージョンのステージングラベルを指定した場合、バージョン ID は指定できません。バージョンのステージを指定しないと、デフォルトの動作として、AWSCURRENT ステージングラベルのシークレットが取得されます。

ステージングラベルは、シークレットが更新またはローテーションされたときに、シークレットのさまざまなバージョンを追跡するために使用します。シークレットの各バージョンには、1 つ以上のステージングラベルと 1 つの ID があります。

`version-id`

使用するシークレットのバージョンの固有 ID を指定します。バージョン ID を指定した場合、バージョンのステージングラベルは指定できません。バージョン ID を指定しないと、デフォルトの動作として、AWSCURRENT ステージングラベルのシークレットが取得されます。

バージョン ID は、シークレットが更新またはローテーションされたときに、シークレットのさまざまなバージョンを追跡するために使用します。シークレットの各バージョンには ID があります。詳細については、AWS Secrets Managerユーザーガイドの「AWS Secrets Managerの主な用語と概念」を参照してください。

コンテナの定義の例

以下の例では、コンテナの定義で Secrets Manager シークレットを参照する方法を示します。

Exampleシークレット全体を参照する

次に示すのは、Secrets Manager シークレットのテキスト全体を参照するときの形式を示すタスク定義のスニペットです。

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name-AbCdEf"
    }]
  }]
}
```

コンテナ内から、このシークレットの値にアクセスするには、$environment_variable_name を呼び出します。

Exampleシークレット全体を参照する

次に示すのは、複数の Secrets Manager シークレットのテキスト全体を参照するときの形式を示すタスク定義のスニペットです。

```
{
  "containerDefinitions": [{
     "secrets": [
      {
        "name": "environment_variable_name1",
         "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name-AbCdEf"
      },
      {
        "name": "environment_variable_name2",
         "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name-abcdef"
      },
      {
        "name": "environment_variable_name3",
        "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name-ABCDEF"
      }
    ]
  }]
}
```

コンテナ内から、このシークレットの値にアクセスするには、$environment_variable_name1、$environment_variable_name2、$environment_variabl を呼び出します。

Exampleシークレット内の特定のキーを参照する

次に示すのは、シークレットの内容と、シークレットに関連付けられているバージョンのステージングラベルおよびバージョン ID を表示する get-secret-value コマンドの出力例です。

```
{
    "ARN": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf",
    "Name": "appauthexample",
    "VersionId": "871d9eca-18aa-46a9-8785-981ddEXAMPLE",
```

```
    "SecretString": "{\"username1\":\"password1\",\"username2\":\"password2\",
\"username3\":\"password3\"}",
    "VersionStages": [
        "AWSCURRENT"
    ],
    "CreatedDate": 1581968848.921
}
```

前のコンテナの定義の出力で特定のキーを参照するには、ARN の最後にキー名を指定します。

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-
AbCdEf:username1::"
    }]
  }]
}
```

Example特定のシークレットバージョンを参照する

次に示すのは、シークレットの暗号化されていない内容と、シークレットのすべてのバージョンのメタデータを表示する describe-secret コマンドの出力例です。

```
{
    "ARN": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf",
    "Name": "appauthexample",
    "Description": "Example of a secret containing application authorization data.",
    "RotationEnabled": false,
    "LastChangedDate": 1581968848.926,
    "LastAccessedDate": 1581897600.0,
    "Tags": [],
    "VersionIdsToStages": {
        "871d9eca-18aa-46a9-8785-981ddEXAMPLE": [
            "AWSCURRENT"
        ],
        "9d4cb84b-ad69-40c0-a0ab-cead3EXAMPLE": [
            "AWSPREVIOUS"
        ]
    }
}
```

前のコンテナの定義の出力で特定のバージョンのステージングラベルを参照するには、ARN の最後にキー名を指定します。

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf::AWSPREVIOUS:"
    }]
  }]
}
```

前のコンテナの定義の出力で特定のバージョン ID を参照するには、ARN の最後にキー名を指定します。

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf:::9d4cb84b-ad69-40c0-a0ab-cead3EXAMPLE"
    }]
  }]
}
```

Exampleシークレットの特定のキーおよびバージョンのステージングラベルを参照する

シークレット内の特定のキーと特定のバージョンのステージングラベルの両方を参照する方法は次のとおりです。

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf:username1:AWSPREVIOUS:"
    }]
  }]
}
```

特定のキーとバージョン ID を指定するには、次の構文を使用します。

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-
AbCdEf:username1::9d4cb84b-ad69-40c0-a0ab-cead3EXAMPLE"
    }]
  }]
}
```

環境変数で指定されたシークレットを使用してタスク定義を作成する方法については、「コンソールを使用した Amazon ECS タスク定義の作成」を参照してください。

Amazon ECS 環境変数を使用して Systems Manager パラメータを伝達する

Amazon ECS を使用すると、AWS Systems Manager Parameter Store のパラメータに機密データを保存した上で、コンテナの定義からそれを参照することによって、コンテナに機密データを取り込むことができます。

環境変数を使用して、コンテナに Systems Manager シークレットを注入する場合は、以下を考慮する必要があります。

- 重要なデータは、コンテナが最初に開始されたときにコンテナに挿入されます。シークレットを後で更新またはローテーションすると、コンテナには更新された値が自動的に送信されなくなります。この場合は、新しいタスクを起動する必要があります。または、タスクがサービスの一部である場合は、サービスを更新し、[Force new deployment] (新しいデプロイの強制) オプションを使用して、新しいタスクの起動をサービスに強制できます。
- AWS Fargate 上の Amazon ECS タスクでは、以下の点を考慮する必要があります。
  - シークレットの内容全体を環境変数として挿入したり、ログ設定にシークレットを挿入したりするには、プラットフォームバージョン `1.3.0` 以降を使用する必要があります。詳細については、「Amazon ECS 向け Fargate プラットフォームバージョン」を参照してください。
  - 特定の JSON キーまたはシークレットのバージョンを環境変数または log 設定に挿入するには、プラットフォームバージョン `1.4.0` 以降 (Linux) または `1.0.0` (Windows) を使用する必要があります。詳細については、「Amazon ECS 向け Fargate プラットフォームバージョン」を参照してください。
- EC2 上の Amazon ECS タスクでは、以下の点を考慮する必要があります。
  - シークレットの特定の JSON キーやバージョンを使用してシークレットを挿入するには、コンテナインスタンスにバージョン `1.37.0` 以降のコンテナエージェントが必要です。ただし、最

新のコンテナエージェントのバージョンを使用することをお勧めします。エージェントのバージョンの確認と最新バージョンへの更新については、「Amazon ECS コンテナエージェントをアップデートする」を参照してください。

シークレットの内容全体を環境変数として挿入したり、ログ設定にシークレットを挿入したりするには、コンテナインスタンスにバージョン `1.22.0` 以降のコンテナエージェントが必要です。

- セキュリティ制御を強化するために、インターフェイス VPC エンドポイントを使用します。Systems Manager 用に、インターフェイス VPC エンドポイントを作成する必要があります。VPC エンドポイントについては、「AWS Systems Manager ユーザーガイド」の「Systems Manager のために VPC エンドポイントを使用して EC2 インスタンスのセキュリティを強化する」を参照してください。
- タスク定義では、Systems Manager パラメーター ストアに対する追加のアクセス許可を持つタスク実行ロールを使用する必要があります。詳細については、「Amazon ECS タスク実行IAM ロール」を参照してください。
- `awslogs` ログドライバーを使用するように設定された Windows タスクの場合は、コンテナインスタンスで `ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE` 環境変数も設定する必要があります。次の構文を使用します。

```
<powershell>
[Environment]::SetEnvironmentVariable("ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE",
 $TRUE, "Machine")
Initialize-ECSAgent -Cluster <cluster name> -EnableTaskIAMRole -LoggingDrivers
 '["json-file","awslogs"]'
</powershell>
```

Systems Manager パラメータを作成する

Systems Manager コンソールを使用すると、機密データ用に Systems Manager Parameter Store のパラメータを作成できます。詳細は「AWS Systems Manager ユーザーガイド」の「Systems Manager パラメータを作成する (コンソール)」または「Systems Manager パラメータを作成する (AWS CLI)」を参照してください。

コンテナ定義に環境変数を追加します。

タスク定義でのコンテナ定義内では、コンテナに設定する環境変数の名前と、コンテナに渡す機密データが含まれている Systems Manager Parameter Store パラメータの ARN 全体を使用して `secrets` を指定します。詳細については、「`secrets`」を参照してください。

以下に示すのは、Systems Manager パラメータストアのパラメータを参照するときの形式を示すタスク定義のスニペットです。起動するタスクと同じリージョンに Systems Manager パラメータストアのパラメータが存在する場合は、パラメータの完全な ARN または名前のどちらも使用できます。パラメータが別のリージョンに存在する場合は、完全な ARN を指定する必要があります。

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:ssm:region:aws_account_id:parameter/parameter_name"
    }]
  }]
}
```

環境変数で指定されたシークレットを使用してタスク定義を作成する方法については、「コンソールを使用した Amazon ECS タスク定義の作成」を参照してください。

Systems Manager Parameter Store のシークレットをプログラムで取得するようにアプリケーションを更新する

Systems Manager Parameter Store のパラメータに保存されている機密データを取得するには、AWS SDK コードサンプルコードライブラリの「AWS SDK を使用した Systems Manager のコードサンプル」を参照してください。

Amazon ECS ログ記録設定のシークレットを伝達する

`logConfiguration` に `secretOptions` パラメータを使用することで、ロギングに使用される機密データを渡すことができます。

シークレットは Secrets Manager または Systems Manager に保存できます。

Secrets Manager を使用する

コンテナの定義内で `logConfiguration` を指定するときに、コンテナに設定するログドライバーオプションの名前と、コンテナに渡す機密データが含まれている Secrets Manager シークレットの ARN 全体を使用して `secretOptions` を指定できます。シークレットの作成の詳細については、「AWS Secrets Manager の作成」を参照してください。

以下に示すのは、Secrets Manager シークレットを参照するときの形式を示すタスク定義のスニペットです。

```
{
```

```
  "containerDefinitions": [{
    "logConfiguration": [{
      "logDriver": "splunk",
      "options": {
        "splunk-url": "https://your_splunk_instance:8088"
      },
      "secretOptions": [{
        "name": "splunk-token",
        "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name-
AbCdEf"
      }]
    }]
  }]
}
```

コンテナ定義に環境変数を追加します。

コンテナの定義内で、コンテナに設定する環境変数の名前と、コンテナに渡す機密データが含まれている Systems Manager パラメータストアのパラメータの ARN 全体を使用して secrets を指定できます。詳細については、「secrets」を参照してください。

以下に示すのは、Systems Manager パラメータストアのパラメータを参照するときの形式を示すタスク定義のスニペットです。起動するタスクと同じリージョンに Systems Manager パラメータストアのパラメータが存在する場合は、パラメータの完全な ARN または名前のどちらも使用できます。パラメータが別のリージョンに存在する場合は、完全な ARN を指定する必要があります。

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:ssm:region:aws_account_id:parameter/parameter_name"
    }]
  }]
}
```

環境変数で指定されたシークレットを使用してタスク定義を作成する方法については、「コンソールを使用した Amazon ECS タスク定義の作成」を参照してください。

使用アイテム Systems Manager

ログ設定には機密データを注入できます。コンテナの定義内で logConfiguration を指定するときに、コンテナに設定するログドライバーオプションの名前と、コンテナに渡す機密データが含まれ

ている Systems Manager パラメータストアのパラメータの ARN 全体を使用して `secretOptions` を指定できます。

> **Important**
>
> 起動するタスクと同じリージョンに Systems Manager パラメータストアのパラメータが存在する場合は、パラメータの完全な ARN または名前のどちらも使用できます。パラメータが別のリージョンに存在する場合は、完全な ARN を指定する必要があります。

以下に示すのは、Systems Manager パラメータストアのパラメータを参照するときの形式を示すタスク定義のスニペットです。

```
{
  "containerDefinitions": [{
    "logConfiguration": [{
      "logDriver": "fluentd",
      "options": {
        "tag": "fluentd demo"
      },
      "secretOptions": [{
        "name": "fluentd-address",
        "valueFrom": "arn:aws:ssm:region:aws_account_id:parameter:/parameter_name"
      }]
    }]
  }]
}
```

Amazon ECS の Secrets Manager シークレットを使用した機密データの指定

Amazon ECS では、機密データを AWS Secrets Manager シークレットに保存し、コンテナ定義でそれを参照することで、機密データをコンテナに挿入することができます。詳細については、「Amazon ECS コンテナに機密データを渡す」を参照してください。

Secrets Manager シークレットを作成して、Amazon ECS タスク定義でシークレットを参照し、コンテナ内の環境変数をクエリすることでシークレットの内容を表示して正しく動作したことを確認する方法について説明します。

前提条件

このチュートリアルでは、以下の前提条件が完了済みであることを前提としています。

- 「Amazon ECS を使用するようにセットアップする」のステップを完了していること。
- ユーザーに、Secrets Manager および Amazon ECS リソースを作成するのに必要な IAM アクセス許可があります。

ステップ 1:Secrets Manager シークレットを作成する

Secrets Manager コンソールを使用して、機密データ用のシークレットを作成できます。このチュートリアルでは、後にコンテナで参照するユーザー名とパスワードを保存するための基本的なシークレットを作成します。詳細については、AWS Secrets Manager ユーザーガイドの「Create an AWS Secrets Manager secret」を参照してください。

[key/value pairs to be stored in this secret] (このシークレットに格納されたキーおよび値のペア) は、チュートリアルの最後にコンテナに存在する環境変数の値です。

[Secret ARN] (シークレット ARN) を保存し、後のステップのタスク実行 IAM ポリシーとタスク定義で参照できるようにします。

ステップ 2: タスク実行ロールにシークレットアクセス許可を追加する

Amazon ECS で Secrets Manager シークレットから機密データを取得するには、タスク実行ロールにシークレットアクセス許可が必要です。詳細については、「Secrets Manager または Systems Manager のアクセス許可」を参照してください。

ステップ 3: タスク定義を作成する

Amazon ECS コンソールを使用して、Secrets Manager シークレットを参照するタスク定義を作成します。

シークレットを指定するタスク定義を作成するには

IAM コンソールを使用して、必要なアクセス許可を持つタスク実行ロールを更新します。

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで、タスク定義 を選択します。
3. [Create new task definition] (新しいタスク定義の作成)、[Create new task definition with JSON] (JSON で新しいタスク定義を作成) の順に選択します。
4. JSON エディタボックスで、以下のタスク定義 JSON テキストを入力して、ステップ 1 で作成した Secrets Manager シークレットの完全な ARN と、ステップ 2 で更新したタスク定義ロールを指定します。[保存] を選択します。

5.

```
{
    "executionRoleArn": "arn:aws:iam::aws_account_id:role/ecsTaskExecutionRole",
    "containerDefinitions": [
        {
            "entryPoint": [
                "sh",
                "-c"
            ],
            "portMappings": [
                {
                    "hostPort": 80,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ],
            "command": [
                "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample
 App</title> <style>body {margin-top: 40px; background-color: #333;} </style> </
head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</
h1> <h2>Congratulations!</h2> <p>Your application is now running on a container in
 Amazon ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html &&
 httpd-foreground\""
            ],
            "cpu": 10,
            "secrets": [
                {
                    "valueFrom":
 "arn:aws:secretsmanager:region:aws_account_id:secret:username_value",
                    "name": "username_value"
                }
            ],
            "memory": 300,
            "image": "public.ecr.aws/docker/library/httpd:2.4",
            "essential": true,
            "name": "ecs-secrets-container"
        }
    ],
    "family": "ecs-secrets-tutorial"
}
```

6. [作成] を選択します。

ステップ 4: クラスターを作成する

Amazon ECS コンソールを使用してコンテナインスタンスを含むクラスターを作成し、タスクを実行します。利用可能なリソースを使用して登録された少なくとも 1 つのコンテナインスタンスを持つ既存のクラスターがあり、このチュートリアル用に作成されたタスク定義の 1 つのインスタンスを実行できる場合は、次のステップに進みます。

このチュートリアルでは、Amazon ECS最適化Amazon Linux 2 AMIを使用して、1つの`t2.micro`コンテナ・インスタンスでクラスタを作成します。

EC2 用にクラスターを作成する方法については、「the section called “Amazon EC2 ワークロード用のクラスターを作成する”」を参照してください。

ステップ 5: タスクを実行する

Amazon ECS コンソールを使用し、作成したタスク定義を使用してタスクを実行できます。このチュートリアルでは、前のステップで作成したクラスターを使用し、EC2 を使用してタスクを実行します。

タスクを実行する方法については、「the section called “タスクとしてのアプリケーションの実行”」を参照してください。

ステップ 6: 確認する

以下のステップを使用して、すべてのステップが正常に完了し、コンテナに環境変数が適切に作成されたことを確認できます。

環境変数が作成されたことを確認するには

1. コンテナインスタンスのパブリック IP アドレスまたは DNS アドレスを見つけます。
   a. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
   b. ナビゲーションペインで [クラスター] を選択した後、作成してあるクラスターを選択します。
   c. [インフラストラクチャ] を選択した後、コンテナインスタンスを選択します。
   d. インスタンスの [パブリック IP] または [パブリック DNS] を記録します。
2. macOS または Linux コンピュータを使用している場合は、以下のコマンドでインスタンスに接続します (パスとアドレスはプライベートキーへのパスとインスタンスのパブリックアドレスに置き換えます)。

```
$ ssh -i /path/to/my-key-pair.pem ec2-user@ec2-198-51-100-1.compute-1.amazonaws.com
```

Windows コンピュータを使用している場合は、「Amazon EC2 ユーザーガイド」の「PuTTY を使用して Linux インスタンスに接続する」を参照してください。

> **Important**
>
> インスタンス接続時の問題について、詳細は「Amazon EC2 ユーザーガイド」の「インスタンスへの接続に関するトラブルシューティング」を参照してください。

3. インスタンスで実行するコンテナを一覧表示します。ecs-secrets-tutorial コンテナのコンテナ ID をメモしておきます。

```
docker ps
```

4. 前のステップの出力のコンテナ ID を使用して ecs-secrets-tutorial コンテナに接続します。

```
docker exec -it container_ID /bin/bash
```

5. echo コマンドを使用して環境変数の値を出力します。

```
echo $username_value
```

このチュートリアルが成功すると、次のような出力が表示されます。

```
password_value
```

> **Note**
>
> あるいは、env (または printenv) コマンドを使用して、コンテナ内の環境変数をすべて一覧表示できます。

ステップ 7: クリーンアップ

このチュートリアルが終了したら、未使用のリソースに対する料金が発生しないように、それに関連付けられたリソースをクリーンアップする必要があります。

リソースをクリーンアップするには

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。
3. [Clusters] (クラスター) ページで、クラスターを選択します。
4. [Delete Cluster] を選択します。
5. 確認ボックスで、delete ***cluster name*** と入力し、[削除] を選択します。
6. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
7. ナビゲーションペインで Roles (ロール) を選択してください。
8. ロールの一覧で `ecsTaskExecutionRole` を探し、選択します。
9. [アクセス許可] を選択した後、[ECSSecrets チュートリアル] の横にある [X] を選択します。[を削除] を選択します。
10. [https://console.aws.amazon.com/secretsmanager/]で、Secrets Manager コンソール を開きます。
11. 作成した [username_value] シークレットを選択し、[Actions (アクション)]、[Delete secret (シークレットの削除)] の順に選択します。

Amazon ECS マネージドインスタンスの Amazon ECS タスク定義パラメータ

タスク定義は、タスクファミリー、AWS Identity and Access Management (IAM) タスクロール、ネットワークモード、コンテナ定義、ボリューム、および容量に分類されます。ファミリとコンテナの定義は、タスク定義の必須項目です。これに対して、タスクロール、ネットワークモード、ボリューム、および容量は省略することができます。

これらのパラメータを JSON ファイルで使用し、タスク定義を設定できます。

Amazon ECS マネージドインスタンスの各タスク定義パラメータについて、以下に詳述します。

ファミリー

`family`

タイプ: 文字列

必須: はい

タスク定義を登録するときに、ファミリー (複数バージョンのタスク定義の名前のようなもの) を指定する必要があります。登録したタスク定義には、リビジョン番号が与えられます。特定のファミリーに登録した最初のタスク定義には、リビジョン 1 が与えられます。その後に登録したタスク定義には、連番でリビジョン番号が与えられます。

Capacity

タスク定義の登録時、Amazon ECS がタスク定義の検証基準となる容量を指定できます。タスク定義が指定された互換性を検証しない場合、クライアント例外が返されます。詳細は「Amazon ECS 起動タイプ」を参照してください。

タスク定義では、以下のパラメータが使用できます。

`requiresCompatibilities`

タイプ: 文字列配列

必須: いいえ

有効な値: `MANAGED_INSTANCES`

タスク定義が検証された容量。これにより、タスク定義で使用されているすべてのパラメータが、Amazon ECS マネージドインスタンスの要件を満たしていることの確認処理が開始されます。

タスクロール

`taskRoleArn`

タイプ: 文字列

必須: いいえ

タスク定義を登録するときに、IAM ロールのタスクロールを割り当てて、タスクのコンテナに、関連するポリシーに指定された AWS API を呼び出すためのアクセス権限を付与できます。詳細については、「Amazon ECS タスクの IAM ロール」を参照してください。

タスク実行ロール

`executionRoleArn`

タイプ: 文字列

必須: 条件による

ユーザーに代わって AWS API コールを実行するアクセス許可を Amazon ECS コンテナエージェントに付与するタスク実行ロールの Amazon リソースネーム (ARN)。詳細については、「Amazon ECS タスク実行IAM ロール」を参照してください。

> **Note**
> タスク実行 IAM ロールは、タスクの要件に応じて必要です。ロールは、プライベート ECR イメージのプルと `awslogs` ログドライバーの使用に必要です。

ネットワークモード

`networkMode`

タイプ: 文字列

必須: いいえ

デフォルト: `awsvpc`

タスクのコンテナで使用するネットワークモード。Amazon ECS マネージドインスタンスでホストされている Amazon ECS タスクの場合、有効な値は `awsvpc` と `host` となります。ネットワークモードが指定されていない場合、デフォルトのネットワークモードは `awsvpc` です。

ネットワークモードが `host` である場合、タスクはネットワークの分離をバイパスし、コンテナはホストのネットワークスタックを直接使用します。

> **Important**
>
> host ネットワークモードをによりタスクを実行する場合、より良いセキュリティーのために、ルートユーザー (UID 0) を使用してコンテナを実行しないでください。セキュリティのベストプラクティスとしては、常にルート以外のユーザーを使用します。

ネットワークモードが awsvpc の場合は、タスクに Elastic Network Interface が割り当てられるため、タスク定義を使用したサービスの作成時またはタスクの実行時に NetworkConfiguration を指定する必要があります。詳細については、「Amazon ECS マネージドインスタンスの Amazon ECS タスクネットワーキング」を参照してください。

host および awsvpc のネットワークモードは、Amazon EC2 のネットワークスタックを使用するので、コンテナのネットワークパフォーマンスが最大限発揮されます。host および awsvpc ネットワークモードにおいて公開されるコンテナポートは、対応するホストポート (host ネットワークモードの場合)、またはアタッチされた Elastic Network Interface ポート (awsvpc ネットワークモードの場合) に直接マッピングされます。このため、動的ホストポートマッピングは使用できません。

ランタイムプラットフォーム

operatingSystemFamily

タイプ: 文字列

必須: いいえ

デフォルト: LINUX

タスク定義を登録する際、オペレーティングシステムファミリを指定します。

このフィールドで有効な値は LINUX です。

サービスで使用されるすべてのタスク定義は、このパラメータに対して同じ値を設定する必要があります。

タスク定義がサービスの一部である場合、この値はサービスの platformFamily 値と一致する必要があります。

`cpuArchitecture`

タイプ: 文字列

必須: 条件による

タスク定義を登録する際は、CPU アーキテクチャを指定します。有効な値は `X86_64` および `ARM64` です。

値を指定しない場合、Amazon ECS はキャパシティプロバイダーの設定に基づいて、使用可能な CPU アーキテクチャにタスクを配置しようとします。タスクが特定の CPU アーキテクチャに配置されるようにするには、タスク定義で `cpuArchitecture` の値を指定します。

サービスで使用されるすべてのタスク定義は、このパラメータに対して同じ値を設定する必要があります。

`ARM64` の詳細については、「the section called “64 ビット ARM ワークロードでのタスク定義”」を参照してください。

タスクサイズ

タスク定義の登録時に、そのタスクが使用する CPU とメモリの合計量を指定できます。これは、コンテナ定義レベルの `cpu` および `memory` の値とは異なります。Amazon EC2 インスタンスでホストされるタスクの場合、これらのフィールドは省略可能です。

> **Note**
>
> タスクレベル CPU およびメモリのパラメータは Windows コンテナでは無視されます。Windows コンテナではコンテナレベルリソースを指定することをお勧めします。

`cpu`

タイプ: 文字列

必須: 条件による

タスクに適用される CPU ユニットのハード制限。JSON ファイルでは、CPU 値を CPU ユニットまたは仮想 CPU (vCPU) の文字列として指定できます。例えば、CPU 値を 1 vCPU (vCPU) ま

たは 1024 (CPU ユニット) として指定できます。タスク定義が登録されると、vCPU 値は、CPU ユニットを示す整数に変換されます。

このフィールドはオプションです。クラスターに、要求された CPU ユニットが利用できる登録済みのコンテナインスタンスがない場合、タスクは失敗します。サポートされている値は、0.125 vCPU から 10 vCPU までです。

`memory`

タイプ: 文字列

必須: 条件による

タスクに適用されるメモリのハード制限です。タスク定義のメモリの値は、メビバイト (MiB) またはギガバイト (GB) の文字列として指定できます。例えば、メモリの値を 3072 (MiB) または 3 GB (GB) のいずれかで指定できます。タスク定義が登録されると、GB 値は、MiB を示す整数に変換されます。

このフィールドはオプションです。任意の値を使用できます。タスクレベルでメモリ値が指定されている場合、コンテナレベルのメモリ値の設定は省略可能です。クラスターに、要求されたメモリが利用できる登録済みのコンテナインスタンスがない場合、タスクは失敗します。特定のインスタンスタイプについて、タスクにできるだけ多くのメモリを提供することで、リソースの使用率を最大化することができます。詳細については、「Amazon ECS Linux コンテナインスタンスのメモリを予約する」を参照してください。

その他のタスク定義パラメータ

以下のタスク定義パラメータは、Amazon ECS コンソールから、[Configure via JSON (JSON による設定)] オプションを使用してタスク定義を登録する際に使用します。詳細については、「コンソールを使用した Amazon ECS タスク定義の作成」を参照してください。

トピック

- エフェメラルストレージ
- IPC モード
- PID モード
- プロキシ設定
- タグ

- Elastic Inference アクセラレーター (廃止)
- 配置の制約事項
- ボリューム

エフェメラルストレージ

`ephemeralStorage`

> **Note**
>
> このパラメータは、Amazon ECS マネージドインスタンス上で実行されるタスクには対応していません。

タイプ: EphemeralStorage オブジェクト

必須: いいえ

タスクに割り当てるエフェメラルストレージの容量(GB 単位)。このパラメーターは、AWS Fargate でホストされるタスクにおいて、利用可能なエフェメラルストレージの総量をデフォルトの容量を超えて拡張する際に使用します。詳細については、「the section called “バインドマウント”」を参照してください。

IPC モード

`ipcMode`

型: 文字列

必須: いいえ

タスクのコンテナで使用する IPC リソースの名前空間。有効な値は `host`、`task` または `none` です。`host` が指定されている場合、同一のコンテナインスタンス上にある (`host` IPC モードを指定した) タスク内のすべてのコンテナは、ホスト Amazon EC2 インスタンスと同じ IPC リソースを共有します。`task` が指定されている場合、指定されたタスク内のすべてのコンテナは同じ IPC リソースを共有します。`none` が指定されている場合、タスクのコンテナ内の IPC リソースはプライベートです。タスク内またはコンテナインスタンスの他のコンテナと共有されることはありません。値を指定しない場合、IPC リソース名前空間の共有はコンテナランタイム設定によって異なります。

PID モード

`pidMode`

型: 文字列

必須: いいえ

タスクのコンテナで使用するプロセス名前空間。有効な値は `host` または `task` です。`host` が指定されている場合、同じコンテナインスタンスで host PID モードを指定したタスク内のすべてのコンテナは、ホスト Amazon EC2 インスタンスと同じプロセス名前空間を共有します。`task` が指定されている場合、指定したタスク内のすべてのコンテナは同じプロセス名前空間を共有します。値が指定されていない場合、デフォルトはプライベート名前空間です。

host PID モードを使用する場合は、意図せずプロセス名前空間が公開されるリスクが高いことに注意してください。

プロキシ設定

`proxyConfiguration`

> **Note**
> このパラメータは、Amazon ECS マネージドインスタンス上で実行されるタスクには対応していません。

タイプ: ProxyConfiguration オブジェクト

必須: いいえ

App Mesh プロキシ設定の詳細。

タグ

タスク定義に適用し、サービスの分類と整理に役立つメタデータ。各タグはキーとオプションの値で構成されます。両方を定義します。

タグには以下のベーシックな制限があります。

- リソースあたりのタグの最大数は 50 です
- タグキーはリソースごとにそれぞれ一意である必要があります。また、各タグキーに設定できる値は 1 つのみです。
- キーの最大長 - 128 文字 (Unicode) (UTF-8)
- 値の最大長 - 256 文字 (Unicode) (UTF-8)
- 複数のサービス間およびリソース間でタグ付けスキーマを使用する場合、他のサービスでも許可される文字に制限が適用されることがあるのでご注意ください。一般的に使用が許可される文字は、UTF-8 で表現可能な文字、数字、スペース、および以下の文字です: +、-、=、.、_、:、/、@。
- タグのキーと値は大文字と小文字が区別されます。
- `aws:`、`AWS:`、またはその大文字または小文字の組み合わせを、キーまたは値のプレフィックスとして使用しないでください。これらの文字列は AWS による使用のために予約されています。このプレフィックスが含まれるタグのキーや値を編集または削除することはできません。このプレフィックスを持つタグは、リソースあたりのタグ数の制限には計算されません。

`key`

タイプ: 文字列

必須: いいえ

タグを構成するキーと値のペアの一部。キーは、より具体的なタグ値のカテゴリのように動作する、一般的なラベルです。

`value`

タイプ: 文字列

必須: いいえ

タグを構成するキーと値のペアのオプションの一部。値はタグカテゴリ (キー) の記述子として機能します。

Elastic Inference アクセラレーター (廃止)

> **Note**
>
> Amazon Elastic Inference (EI) は利用できなくなりました。

`inferenceAccelerator`

> **Note**
> このパラメータは、Amazon ECS マネージドインスタンス上で実行されるタスクには対応していません。

タイプ: InferenceAccelerator オブジェクト

必須: いいえ

タスク内のコンテナに使用する Elastic Inference アクセラレーター。

配置の制約事項

`placementConstraints`

タイプ: TaskDefinitionPlacementConstraint オブジェクトの配列

必須: いいえ

タスクに使用する、配置制約オブジェクトの配列。タスクごとに最大 10 個の制約を指定できます (この制限数には、タスク定義内の制約と、実行時に指定される制約が含まれます)。

Amazon ECS は、Amazon ECS マネージドインスタンスで実行されるタスクの `distinctInstace` および `memberOf` の配置制約に対応しています。`memberOf` 配置制約を使用するタスクでは、次の属性がサポートされています。

- `ecs.subnet-id`
- `ecs.availability-zone`
- `ecs.cpu-architecture`
- `ecs.instance-type`

配置制約について、詳細は「Amazon ECS がタスクに使用するコンテナインスタンスを定義する」を参照してください。

ボリューム

タスク定義を登録するときは、オプションでタスクのボリュームのリストを指定できます。これにより、タスクでデータボリュームを使用できます。

ボリュームタイプおよびその他のパラメータについて、詳細は「the section called “タスクのストレージオプション”」を参照してください。

name

タイプ: 文字列

必須: はい

ボリュームの名前。最大 255 文字の英字 (大文字と小文字)、数字、ハイフンを使用できます。この名前は、コンテナ定義 sourceVolume の mountPoints パラメータで参照されます。

host

タイプ: HostVolumeProperties オブジェクト

必須: いいえ

このパラメータは、バインドマウントのホストボリュームを使用する場合に指定します。host パラメータの内容により、バインドマウントのホストボリュームがホストコンテナインスタンスで保持されるかどうかと、その保存場所が決まります。host パラメータが空の場合は、システムがデータボリュームにホストパスを割り当てます。ただし、関連付けられたコンテナが実行を停止した後も、データが保持されるという保証はありません。

sourcePath

タイプ: 文字列

必須: いいえ

host パラメータを使用するときは、sourcePath を指定して、コンテナに提示されるホストインスタンス上のパスを宣言します。このパラメータが空の場合は、システムがホストパスを割り当てます。host パラメータにファイルの場所 sourcePath が含まれている場合、データボリュームは、手動で削除されるまでホストインスタンス上の指定された場所に保持されます。sourcePath の値がホストインスタンスに存在しない場合は、システムが値を作成します。その場所が存在する場合は、ソースパスフォルダの内容がエクスポートされます。

Amazon ECS マネージドインスタンスでは、ホストファイルシステムの一部が読み取り専用になっています。sourcePath は、/var または /tmp などの書き込み可能なディレクトリをポイントしている必要があります。詳細については、「Amazon ECS でのバインドマウントの使用」を参照してください。

`dockerVolumeConfiguration`

> **Note**
>
> このパラメータは、Amazon ECS マネージドインスタンス上で実行されるタスクには対応していません。

タイプ: DockerVolumeConfiguration オブジェクト

必須: いいえ

このパラメータは、Docker ボリュームを使用する場合に指定します。

`efsVolumeConfiguration`

タイプ: EFSVolumeConfiguration オブジェクト

必須: いいえ

このパラメータは、タスクストレージに Amazon EFS ファイルシステムを使用するときに指定します。

`fsxWindowsFileServerVolumeConfiguration`

> **Note**
>
> このパラメータは、Amazon ECS マネージドインスタンス上で実行されるタスクには対応していません。

タイプ: FSxWindowsFileServerVolumeConfiguration オブジェクト

必須: いいえ

このパラメータは、タスクストレージに Amazon FSx for Windows File Server ファイルシステムを使用する場合に指定します。

`configuredAtLaunch`

タイプ: ブール値

必須: いいえ

起動時にボリュームを設定するかどうかを示します。これは、スタンドアロンタスクまたはサービスの一部として作成されたタスクの Amazon EBS ボリュームを作成するために使用されます。各タスク定義リビジョンでは、ボリューム設定で起動時に設定できるボリュームは 1 つだけです。

コンテナ定義

タスク定義を登録するときは、コンテナインスタンスの Docker デーモンに渡されるコンテナ定義のリストを指定する必要があります。以下のパラメータをコンテナ定義で使用できます。

トピック

- 名前
- 画像
- メモリ
- CPU
- ポートマッピング
- プライベートリポジトリの認証情報
- Essential
- エントリポイント
- コマンド
- 作業ディレクトリパス
- 詳細コンテナ定義パラメータ
- Linux パラメータ

名前

`name`

タイプ: 文字列

必須: はい

コンテナの名前。最大 255 文字の英字 (大文字と小文字)、数字、ハイフン、アンダースコアを使用できます。タスク定義で複数のコンテナをリンクしている場合、あるコンテナの name を別のコンテナの links に入力できます。これにより、コンテナ同士を接続します。

画像

`image`

タイプ: 文字列

必須: はい

コンテナの開始に使用するイメージ。この文字列は Docker デーモンに直接渡されます。デフォルトでは、Docker Hub レジストリのイメージを使用できます。*repository-url*/*image*:*tag* または *repository-url*/*image*@*digest* で他のリポジトリを指定することもできます。最大 255 文字の英字 (大文字と小文字)、数字、ハイフン、アンダースコア、コロン、ピリオド、スラッシュ、シャープ記号を使用できます。このパラメータは、docker create-container コマンドの Image と Docker run コマンドの IMAGE パラメータにマッピングされます。

- 新しいタスクが開始されると、Amazon ECS コンテナエージェントは、指定されたイメージおよびタグの最新バージョンをプルしてコンテナで使用します。ただし、リポジトリイメージの後続の更新がすでに実行されているタスクに反映されることはありません。
- タスク定義のイメージパスでタグまたはダイジェストを指定しない場合、Amazon ECS コンテナエージェントは latest タグを使用して指定されたイメージを取り込みます。
- リポジトリイメージに後から加えられた更新内容が、実行中のタスクに反映されることはありません。
- プライベートレジストリのイメージがサポートされています。詳細については、「Amazon ECS での AWS 以外のコンテナイメージの使用」を参照してください。
- Amazon ECR リポジトリのイメージは、registry/repository:tag または registry/repository@digest の完全な命名規則 (例えば、*aws_account_id*.dkr.ecr.*region*.amazonaws.com/*my-web-app*:*latest* や、*aws_account_id*.dkr.ecr.*region*.amazonaws.com/*my-web-app*@*sha256:94afd1f2e64d908bc90dbca0035a5b567EXAMPLE*) を使用して指定します。
- Docker Hub の公式リポジトリのイメージでは、1 つの名前 (例: ubuntu または mongo) を使用します。

- Docker Hub の他のリポジトリのイメージは、組織名で修飾されます (例: `amazon/amazon-ecs-agent`)。
- 他のオンラインリポジトリのイメージは、さらにドメイン名で修飾されます (例: `quay.io/assemblyline/ubuntu`)。

`versionConsistency`

タイプ: 文字列

有効な値: `enabled|disabled`

必須: いいえ

コンテナ定義で指定されたコンテナイメージタグを Amazon ECS がイメージダイジェストに解決するかどうかを指定します。デフォルトでは、この動作は `enabled` です。コンテナの値を `disabled` として設定した場合、Amazon ECS はコンテナイメージタグをダイジェストに解決せず、コンテナ定義で指定された元のイメージ URI をデプロイ用に使用します。コンテナイメージの解決の詳細については、「コンテナイメージの解決」を参照してください。

メモリ

`memory`

タイプ: 整数

必須: いいえ

コンテナに適用されるメモリの量 (MiB 単位)。コンテナは、ここで指定したメモリを超えようとすると、強制終了されます。タスク内のすべてのコンテナ用に予約されるメモリの合計量は、タスクの `memory` 値より小さくする必要があります (指定されている場合)。このパラメータは docker create-container コマンドの `Memory` にマッピングされ、`--memory` オプションは docker run にマッピングされます。

Docker デーモン 20.10.0 以降によって、コンテナ用として 6 MiB 以上のメモリが予約されます。従って、このコンテナに対しては 6 MiB 未満のメモリは指定しないようにします。

Docker デーモン 19.03.13-ce 以降では、コンテナ用として 4 MiB 以上のメモリが予約されます。このため、このコンテナ用には 4 MiB 未満のメモリを指定しないようにします。

> **Note**
>
> リソースの使用率を最大化することを目的に、特定のインスタンスタイプにおいて、タスクにできるだけ多くのメモリを提供する場合には、「Amazon ECS Linux コンテナインスタンスのメモリを予約する」を参照してください。

`memoryReservation`

タイプ: 整数

必須: いいえ

コンテナ用に予約するメモリのソフト制限 (MiB 単位)。システムメモリに競合がある場合、Docker はコンテナのメモリ量をこのソフト制限値内に維持しようとします。ただし、コンテナは必要に応じてより多くのメモリを使用することができます。コンテナは、`memory` パラメーターで指定されたハード制限 (該当する場合) またはコンテナインスタンス上の利用可能なすべてのメモリのいずれか早く達する方まで使用できます。このパラメータは docker create-container コマンドの `MemoryReservation` にマッピングされ、`--memory-reservation` オプションは docker run にマッピングされます。

タスクレベルでメモリ値を指定しない場合、コンテナ定義で `memory` または `memoryReservation` の一方または両方に 0 以外の整数を指定する必要があります。両方を指定する場合、`memory` は `memoryReservation` より大きいことが必要です。`memoryReservation` を指定する場合、コンテナが配置されているコンテナインスタンスで使用可能なメモリリソース量から、上記の値が減算されます。それ以外の場合は、`memory` の値が使用されます。

例えば、コンテナが通常 128 MiB のメモリを使用しているが、短期間にメモリが 256 MiB にバーストする場合があるとします。128 MiB の `memoryReservation` と 300 MiB の `memory` ハード制限を設定できます。この設定では、コンテナが、コンテナインスタンスの残りのリソースから 128 MiB のメモリのみを予約できます。同時に、この構成により、コンテナは必要に応じてより多くのメモリ リソースを使用できるようになります。

Docker デーモン 20.10.0 以降によって、コンテナ用として 6 MiB 以上のメモリが予約されます。従って、このコンテナに対しては 6 MiB 未満のメモリは指定しないようにします。

Docker デーモン 19.03.13-ce 以降では、コンテナ用として 4 MiB 以上のメモリが予約されます。このため、このコンテナ用には 4 MiB 未満のメモリを指定しないようにします。

> **Note**
>
> リソースの使用率を最大化することを目的に、特定のインスタンスタイプにおいて、タスクにできるだけ多くのメモリを提供する場合には、「Amazon ECS Linux コンテナインスタンスのメモリを予約する」を参照してください。

CPU

`cpu`

タイプ: 整数

必須: いいえ

コンテナ用に予約されている `cpu` ユニットの数。このパラメータは docker create-container コマンドの `CpuShares` にマッピングされ、`--cpu-shares` オプションは docker run にマッピングされます。

EC2 キャパシティプロバイダーを使用するタスクでは、このフィールドはオプションです。その場合、タスク内のすべてのコンテナ用に予約される CPU の合計量が、タスクレベルの `cpu` の値を下回ることが唯一の要件となります。

> **Note**
>
> [Amazon EC2 インスタンス] 詳細ページのインスタンスタイプに一覧表示されている vCPU 数に 1,024 を乗算して、EC2 インスタンスタイプごとに使用可能な CPU ユニットの数を判断できます。

Linux コンテナは、割り当てられた CPU ユニットと同じ比率を使用して、割り当てられていない CPU ユニットをコンテナインスタンス上の他コンテナと共有します。例えば、単一コンテナタスクを単一コアインスタンスタイプで実行する場合、そのコンテナ用に 512 個の CPU ユニットを指定しており、そのタスクがコンテナインスタンスで実行される唯一のタスクであると、そのコンテナは 1,024 個の CPU ユニットすべてをいつでも使用できます。ただし、そのコンテナインスタンスで同じタスクの別のコピーを起動した場合、各タスクについて、必要に応じて最低で 512 CPU ユニットが保証されます。さらに、各コンテナは他のコンテナが使用していなければ、より高い CPU 使用率にフロートする可能性があります。両方のタスクが常に 100% アクティブであった場合、512 CPU ユニットに制限されます。

Linux コンテナインスタンスでは、コンテナインスタンス上の Docker デーモンは、CPU 値を使用して、実行中のコンテナに対する相対 CPU 配分比率を計算します。詳細については、Docker ドキュメントの「CPU 配分の制約」を参照してください。Linux kernel が許可する CPU の有効な最小共有値は 2 です。ただし、CPU のパラメータは必要なく、コンテナ定義で 2 以下の CPU 値を使用できます。CPU 値が 2 未満の場合 (null を含む)、動作は Amazon ECS コンテナエージェントのバージョンによって異なります。

- エージェントのバージョンが 1.1.0 以下である: Null と 0 の CPU 値は Docker に 0 として渡され、Docker はそれを 1,024 個の CPU 配分に変換します。1 の CPU 値は Docker に 1 として渡され、Linux カーネルはそれを 2 個の CPU 配分に変換します。
- エージェントのバージョンが 1.2.0 以上である: Null、0、1 の CPU 値は Docker に 2 として渡されます。

Windows コンテナインスタンスでは、CPU 制限は絶対制限またはクォータとして適用されます。Windows コンテナはタスク定義で指定された一定量の CPU にのみアクセスします。ヌルまたはゼロの CPU 値は `0` として Docker に渡され、これを Windows は 1 つの CPU の 1% として解釈します。

ポートマッピング

`portMappings`

タイプ: オブジェクト配列

必須: いいえ

ポートマッピングは、コンテナのネットワークポートを外部に公開します。これは、クライアントがアプリケーションにアクセスすることを可能にします。これは、同じタスク内にあるコンテナ間の通信にも使用されます。

`awsvpc` ネットワークモードを使用するタスク定義では、`containerPort` のみを指定します。`hostPort` は常に無視され、コンテナポートはホスト上の番号が大きいランダムなポートに自動的にマップされます。

このパラメータのほとんどのフィールド (`containerPort`、`hostPort`、`protocol` を含む) は docker create-container コマンドの `PortBindings` にマッピングされ、`--publish` オプションは docker run にマッピングされます。タスク定義でネットワークモードを `host` に設定している場合、ホストポートは未定義とするか、ポートマッピングのコンテナポートと一致させる必要があります。

> **Note**
>
> タスクが RUNNING ステータスに達すると、手動および自動で割り当てられたホストとコンテナポートが次の場所で表示されます:
>
> - コンソール: 指定されたタスクのコンテナ詳細の [Network Bindings] セクション。
> - AWS CLI: networkBindings コマンド出力の describe-tasks セクション。
> - API: DescribeTasks レスポンス。
> - メタデータ: タスクメタデータのエンドポイント。

`appProtocol`

タイプ: 文字列

必須: いいえ

ポートマッピングに使用されるアプリケーションプロトコル。このパラメータは Service Connect にのみ適用されます。アプリケーションが使用するプロトコルと一貫性を持つように、このパラメータを設定することをお勧めします。このパラメータを設定すると、Amazon ECS はサーバー コネクト プロキシにプロトコル固有の接続処理を追加します。このパラメータを設定すると、Amazon ECS は Amazon ECS コンソールと CloudWatch にプロトコル固有のテレメトリを追加します。

このパラメータ値を設定しないと、TCP が使用されます。ただし、Amazon ECS では、TCP 用のプロトコル固有のテレメトリーは追加されません。

詳細については、「the section called “ Service Connect”」を参照してください。

有効なプロトコル値: `"HTTP" | "HTTP2" | "GRPC"`

`containerPort`

タイプ: 整数

必須: はい (`portMappings` を使用する場合)

ユーザーが指定したホストポートまたは自動的に割り当てられたホストポートにバインドされるコンテナポートの番号。

`awsvpc` ネットワークモードを使用するタスクの場合は、`containerPort` を使用して公開ポートを指定します。

`containerPortRange`

タイプ: 文字列

必須: いいえ

動的にマッピングされたホストポート範囲にバインドされるコンテナのポート番号の範囲。

このパラメータは、`register-task-definition` API を使用してのみ設定できます。このオプションは、`portMappings` パラメータで使用できます。詳細については、「AWS Command Line Interface リファレンス」の「register-task-definition」を参照してください。

`containerPortRange` を指定するときは、以下のルールが適用されます。

- `awsvpc` ネットワークモードを使用する必要があります。
- コンテナインスタンスには、少なくともコンテナエージェントのバージョン 1.67.0 と `ecs-init` パッケージのバージョン 1.67.0-1 が必要です。
- 各コンテナにつき、最大 100 のポートレンジを指定できます。
- `hostPortRange` は指定しません。`hostPortRange` の値は次のように設定されます。
  - `awsvpc` ネットワークモードのタスク内のコンテナでは、`hostPort` は `containerPort` と同じ値に設定されます。これは静的マッピング戦略です。
- `containerPortRange` の有効な値は 1 ~ 65535 です。
- 1 つのポートは、コンテナごとに 1 つのポートマッピングにのみ含められます。
- 重複するポート範囲は指定できません。
- 範囲内の最初のポートは、最後のポートよりも小さくなければなりません。
- Docker では、ポートの数が多い場合は、Docker デーモン設定ファイルの Docker-proxy をオフにすることをお勧めします。

  詳細については、GitHub の Issue #11185 を参照してください。

  Docker デーモン設定ファイルの docker-proxy をオフにする方法については、Amazon ECS 開発者ガイドの「Docker デーモン」を参照してください。

DescribeTasks を呼び出すと、コンテナポートにバインドされているホストポートである `hostPortRange` を表示できます。

ポート範囲は、EventBridge に送信される Amazon ECS タスクイベントに含まれません。詳細については、「the section called “EventBridge を使用して Amazon ECS エラーへの対応を自動化する”」を参照してください。

`hostPortRange`

型: 文字列

必須: いいえ

ネットワークバインディングで使用されるホストのポート番号範囲。これは Docker によって割り当てられ、Amazon ECS エージェントによって配信されます。

`hostPort`

タイプ: 整数

必須: いいえ

コンテナ用に予約するコンテナインスタンスのポート番号。

`hostPort` は、空白のままにするか、`containerPort` と同じ値にできます。

Docker バージョン 1.6.0 以降のデフォルトの一時ポート範囲は、インスタンスの `/proc/sys/net/ipv4/ip_local_port_range` にリストされています。このカーネルパラメータが使用できない場合、`49153-65535` から始まるデフォルトのエフェメラルポート範囲が使用されます。エフェメラルポート範囲では、ホストポートを指定しないでください。これらの範囲は自動割り当て用に予約済みです。一般的に、32768 より小さい番号のポートは一時ポート範囲に含まれません。

デフォルトの予約済みポートは、SSH 用の 22、Docker ポートの 2375 および 2376、Amazon ECS コンテナエージェントポートの `51678-51680` です。実行中のタスクに対して以前にユーザーが指定したホストポートも、タスクの実行中に予約されます。タスクが停止すると、ホストポートはリリースされます。現在の予約済みポートは、describe-container-instances 出力の `remainingResources` に表示されます。コンテナインスタンスには、デフォルトの予約済みポートを含めて、一度に最大 100 個の予約済みポートを割り当てられます。自動的に割り当てられるポートは、この 100 個の予約済みポートクォータにはカウントされません。

`name`

タイプ: 文字列

必須: いいえ。サービスに Service Connect および VPC Lattice を設定する場合は必要です。

ポートマッピングに使用される名前。このパラメータは Service Connect および VPC Lattice にのみ適用されます。このパラメータは、サービスの Service Connect および VPC Lattice 設定で使用する名前です。

詳細については、「Service Connect を使用して Amazon ECS サービスを短縮名で接続する」を参照してください。

次の例では、Service Connect および VPC Lattice の 2 つの必須フィールドが使用されています。

```
"portMappings": [
    {
        "name": string,
        "containerPort": integer
    }
]
```

`protocol`

タイプ: 文字列

必須: いいえ

ポートマッピングに使用されるプロトコル。有効な値は、`tcp` および `udp` です。デフォルト: `tcp`。

> **Important**
>
> `tcp` は Service Connect でのみサポートされます。このフィールドが設定されていない場合は、`tcp` が暗示されることに注意してください。

ホストポートを指定する際は、以下の構文を使用します。

```
"portMappings": [
    {
        "containerPort": integer,
        "hostPort": integer
    }
    ...
]
```

自動割り当てのホストポートが必要な場合は、以下の構文を使用します。

```
"portMappings": [
    {
        "containerPort": integer
    }
    ...
]
```

プライベートリポジトリの認証情報

`repositoryCredentials`

タイプ: RepositoryCredentials オブジェクト

必須: いいえ

プライベートレジストリ認証用のリポジトリ認証情報。

詳細については、「Amazon ECS での AWS 以外のコンテナイメージの使用」を参照してください。

`credentialsParameter`

タイプ: 文字列

必須: はい (`repositoryCredentials` を使用する場合)

プライベートリポジトリの認証情報が含まれているシークレットの Amazon リソースネーム (ARN)。

詳細については、「Amazon ECS での AWS 以外のコンテナイメージの使用」を参照してください。

> **Note**
>
> Amazon ECS API、AWS CLI、または AWS SDK を使用する場合、起動するタスクと同じリージョンにシークレットが存在する場合は、シークレットの完全な ARN または名前のどちらも使用できます。AWS マネジメントコンソール を使用する場合は、シークレットの完全な ARN を指定する必要があります。

必要なパラメータが含まれるタスク定義のスニペットを、以下に示します。

```
"containerDefinitions": [
    {
        "image": "private-repo/private-image",
        "repositoryCredentials": {
            "credentialsParameter":
 "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name"
        }
    }
]
```

Essential

`essential`

タイプ: ブール値

必須: いいえ

コンテナの `essential` パラメータが `true` とマークされている場合、そのコンテナが何らかの理由で失敗または停止すると、タスクに含まれる他のすべてのコンテナは停止されます。コンテナの `essential` パラメータが `false` とマークされている場合、その失敗はタスクに含まれる残りのコンテナに影響を与えません。このパラメータを省略した場合、コンテナは必須と見なされます。

すべてのタスクには少なくとも 1 つの必須コンテナが必要です。アプリケーションが複数のコンテナで構成される場合は、用途が共通するコンテナをコンポーネントにグループ化し、それらのコンポーネントを複数のタスク定義に分ける必要があります。詳細については、「the section called “アプリケーションの構築”」を参照してください。

エントリポイント

`entryPoint`

タイプ: 文字列配列

必須: いいえ

コンテナに渡されるエントリポイント。このパラメータは docker create-container コマンドの Entrypoint にマッピングされ、--entrypoint オプションは docker run にマッピングされます。

```
"entryPoint": ["string", ...]
```

コマンド

command

タイプ: 文字列配列

必須: いいえ

コンテナに渡されるコマンド。このパラメータは docker create-container コマンドの Cmd にマッピングされ、COMMAND パラメータは docker run にマッピングされます。複数の引数がある場合、各引数は配列において区切られた文字列である必要があります。

```
"command": ["string", ...]
```

作業ディレクトリパス

workingDirectory

タイプ: 文字列

必須: いいえ

コマンドを実行するコンテナ内の作業ディレクトリ。このパラメータは docker create-container コマンドの WorkingDir にマッピングされ、--workdir オプションは docker run にマッピングされます。

詳細コンテナ定義パラメータ

以下のコンテナ定義用の詳細パラメータは、Amazon ECS コンテナインスタンスでのコンテナの起動に使用する Docker run コマンドの拡張機能を追加します。

トピック

- 再起動ポリシー
- ヘルスチェック
- 環境
- セキュリティ
- ネットワーク設定
- ストレージとログ記録
- リソースの要件
- コンテナのタイムアウト
- コンテナの依存関係
- システムコントロール
- インタラクティブ
- 擬似ターミナル

再起動ポリシー

`restartPolicy`

コンテナ再起動ポリシーと関連する設定パラメータ。コンテナの再起動ポリシーを設定すると、Amazon ECS はタスクを置き換えることなくコンテナを再起動できます。詳細については、「コンテナ再起動ポリシーを使用して Amazon ECS タスク内の個々のコンテナを再起動する」を参照してください。

`enabled`

タイプ: ブール値

必須: はい

コンテナに対して再起動ポリシーを有効にするかどうかを指定します。

`ignoredExitCodes`

タイプ: 整数配列

必須: いいえ

Amazon ECS が無視し、再起動を試みない終了コードのリスト。最大 50 個のコンテナ終了コードを指定できます。デフォルトでは、Amazon ECS はいずれの終了コードも無視しません。

`restartAttemptPeriod`

タイプ: 整数

必須: いいえ

再起動を試みる前にコンテナが実行する必要がある時間 (秒単位) 。コンテナを再起動できるのは、`restartAttemptPeriod` 秒に 1 回のみです。コンテナがこの期間実行できずに早く終了した場合、コンテナは再起動されません。最小の `restartAttemptPeriod` は 60 秒、最大の `restartAttemptPeriod` は 1,800 秒を指定できます。デフォルトでは、コンテナは再起動の前に 300 秒間実行する必要があります。

ヘルスチェック

`healthCheck`

コンテナに対するヘルスチェックのコマンドと、コンテナのための関連する設定パラメータです。詳細については、「コンテナのヘルスチェックを使用して Amazon ECS タスク状態を判定する」を参照してください。

`command`

正常状態かどうかを決定するために、コンテナが実行するコマンドが格納された文字列配列。この文字列配列の先頭には、コマンド引数を直接実行するための `CMD`、またはコンテナのデフォルトシェルでコマンドを実行するための `CMD-SHELL` を付加できます。これらのいずれも指定しない場合は `CMD` が使用されます。

AWS マネジメントコンソール にタスク定義を登録する場合は、カンマで区切ったコマンドリストを使用してください。これらのコマンドは、タスク定義が作成された後に文字列に変換されます。ヘルスチェックに対する入力の例を次に示します。

```
CMD-SHELL, curl -f http://localhost/ || exit 1
```

AWS マネジメントコンソール JSON パネル、AWS CLI、または API を使用してタスク定義を登録するときは、コマンドのリストを角かっこで囲みます。ヘルスチェックに対する入力の例を次に示します。

```
[ "CMD-SHELL", "curl -f http://localhost/ || exit 1" ]
```

`stderr` が出力されない終了コード 0 は成功を示し、0 以外の終了コードは失敗を示します。

`interval`

各ヘルスチェック間の時間間隔 (秒単位)。5 ～ 300 秒を指定できます。デフォルトの値は 30 秒です。

`timeout`

成功まで待機しているヘルスチェックが、失敗したと見なされるまでの期間 (秒単位) です。2 ～ 60 秒を指定できます。デフォルト値は 5 秒です。

`retries`

コンテナが異常と見なされるまでに、失敗したヘルスチェックを再試行する回数です。1～10 回を指定できます。デフォルト値は 3 回の再試行です。

`startPeriod`

失敗したヘルスチェックの再試行が最大回数に達する前に、コンテナにブートストラップための時間を提供するオプションの猶予期間です。0 ~ 300 秒の値を指定できます。デフォルトでは、`startPeriod` は無効となっています。

ヘルスチェックが `startPeriod` 内で成功した場合、コンテナは正常であるとみなされ、その後の失敗は最大再試行回数にカウントされます。

環境

`cpu`

タイプ: 整数

必須: いいえ

Amazon ECS コンテナエージェントがコンテナ用に予約した `cpu` ユニットの数。Linux では、このパラメータは「Create a container」セクションの `CpuShares` にマッピングされます。

このフィールドは、Amazon ECS マネージドインスタンスで実行されるタスクではオプションです。タスク内のすべてのコンテナのために予約されている CPU の合計量は、タスクレベルの `cpu` 値以上にはできません。

Linux コンテナは、割り当てられた CPU ユニットと同じ比率を使用して、割り当てられていない CPU ユニットをコンテナインスタンス上の他コンテナと共有します。例えば、そのコンテナ用に 512 個の CPU ユニットが指定された単一コアインスタンスタイプで、唯一のコンテナタスクを実行するとします。さらに、そのタスクはコンテナインスタンスで実行される唯一のタスクで

す。この場合のコンテナは、1,024 個の CPU ユニット配分を任意のタイミングで使用できます。一方、そのコンテナインスタンスで同じタスクのコピーを別途起動したと仮定します。各タスクには、必要に応じて少なくとも 512 個の CPU ユニットが保証されます。同様に、その他のコンテナが残りの CPU を使用していない場合、各コンテナは CPU 使用率を高められます。ただし、両方のタスクが常に 100% アクティブである場合、使用できるのは 512 CPU ユニットに制限されます。

Linux コンテナインスタンスでは、コンテナインスタンス上の Docker デーモンは、CPU 値を使用して、実行中のコンテナに対する相対 CPU 配分比率を計算します。Linux カーネルが許可する最小の有効な CPU 共有値は 2 で、Linux カーネルが許可する最大の有効な CPU 共有値は 262,144 です。ただし、CPU のパラメータは必須ではなく、コンテナ定義では 2 未満の CPU 値および 262144 より大きな CPU 値を使用できます。CPU 値が 2 未満 (null を含む) および 262144 より大きな値である場合、動作は Amazon ECS コンテナエージェントのバージョンによって異なります。

その他の例については、「Amazon ECS で CPU およびメモリリソースの管理方法」を参照してください。

`gpu`

タイプ: ResourceRequirement オブジェクト

必須: いいえ

Amazon ECS コンテナエージェントがコンテナ用に予約した物理 GPUs の数。整数値、フラクショナル GPU の小数値、または `ALL` を指定できます。`ALL` を指定すると、コンテナインスタンス上のすべての GPU がコンテナに割り当てられます。10 進値 (`0.125`、`0.25`、`0.5` など) を指定すると、Amazon ECS はリクエストされた GPU 容量を提供する小数 GPU インスタンス (G6f) にタスクを配置します。タスク内でコンテナ用に予約されているすべての GPU の数は、タスクが起動されたコンテナインスタンスで使用できる GPU の数を超えることはできません。詳細については、「GPU ワークロード向けの Amazon ECS タスク定義」を参照してください。

`Neuron device`

タイプ: ResourceRequirement オブジェクト

必須: いいえ

Amazon ECS コンテナエージェントがコンテナ用に予約した Neuron デバイス。サポートされている値は `ALL` のみです。これにより、インスタンス上のすべての Neuron デバイスへの排他的アクセスがコンテナに付与されます。詳細については、「AWS Neuron 機械学習ワークロードでの Amazon ECS タスク定義」を参照してください。

`Elastic Inference accelerator`

> **Note**
>
> このパラメータは、Amazon ECS マネージドインスタンスがホストするコンテナではサポートされていません。

タイプ: ResourceRequirement オブジェクト

必須: いいえ

`InferenceAccelerator` タイプでは、`value` はタスク定義で指定される `InferenceAccelerator` の `deviceName` と一致します。詳細については、「the section called “Elastic Inference アクセラレーター名 (廃止)”」を参照してください。

`essential`

タイプ: ブール値

必須: いいえ

コンテナの `essential` パラメーターが `true` としてマークされており、そのコンテナが何らかの理由で失敗するか停止するとします。その後、タスクの一部である他のすべてのコンテナが停止されます。コンテナの `essential` パラメータを `false` にマークしておくと、そのコンテナ失敗はタスク内にある残りのコンテナに影響を与えません。このパラメータを省略した場合、コンテナは必須と見なされます。

すべてのタスクには少なくとも 1 つの必須コンテナが必要です。複数のコンテナで構成されるアプリケーションがあるとします。次に、共通の目的で使用されるコンテナをコンポーネントにグループ化し、さまざまなコンポーネントを複数のタスク定義に分離します。詳細については、「Amazon ECS 用のアプリケーションの構築」を参照してください。

```
"essential": true|false
```

`entryPoint`

> **Important**
>
> 初期のバージョンの Amazon ECS コンテナエージェントは、`entryPoint` パラメータを正しく処理しません。`entryPoint` の使用時に問題が発生する場合は、コンテナエー

ジェントを更新するか、代わりに command 配列項目としてコマンドと引数を入力します。

タイプ: 文字列配列

必須: いいえ

コンテナに渡されるエントリポイント。

```
"entryPoint": ["string", ...]
```

command

タイプ: 文字列配列

必須: いいえ

コンテナに渡されるコマンド。このパラメータは create-container コマンドの Cmd にマッピングされ、 COMMAND パラメータは docker run にマッピングされます。複数の引数がある場合、各引数は配列内で、区切られた文字列により指定する必要があります。

```
"command": ["string", ...]
```

workingDirectory

タイプ: 文字列

必須: いいえ

コマンドを実行するコンテナ内の作業ディレクトリ。このパラメータは、Docker Remote API の Create a container (コンテナを作成する) セクションの WorkingDir にマップされ、--workdir オプションは docker run にマップされます。

```
"workingDirectory": "string"
```

environmentFiles

タイプ: オブジェクト配列

必須: いいえ

コンテナに渡す環境変数が含まれるファイルのリスト。このパラメータは Docker run コマンドの `--env-file` オプションにマッピングされます。

最大 10 個の環境ファイルを指定できます。ファイルには、ファイル拡張子 `.env` が必要です。環境ファイルの各行には、`VARIABLE=VALUE` 形式で環境変数が含まれています。`#` で始まる行はコメントとして扱われ、無視されます。

コンテナ定義に個別の環境変数が指定されている場合は、環境ファイルに含まれる変数よりも優先されます。同じ変数を含む複数の環境ファイルが指定されている場合、それらのファイルは上から下に処理されます。一意の変数名を使用することをお勧めします。詳細については、「個々の環境変数を Amazon ECS コンテナに渡す」を参照してください。

`value`

型: 文字列

必須: はい

環境変数ファイルを含む Amazon S3 オブジェクトの Amazon リソースネーム (ARN)。

`type`

型: 文字列

必須: はい

使用するファイルのタイプ。`s3` はサポートされる唯一の値です。

`environment`

タイプ: オブジェクト配列

必須: いいえ

コンテナに渡す環境変数。このパラメータは docker create-container コマンドの `Env` にマッピングされ、`--env` オプションは docker run コマンドにマッピングされます。

> **Important**
>
> 認証情報データなどの機密情報にプレーンテキストの環境変数を使用することはお勧めしません。

name

型: 文字列

必須: はい (environment を使用する場合)

環境変数の名前。

value

型: 文字列

必須: はい (environment を使用する場合)

環境変数の値。

```
"environment" : [
    { "name" : "string", "value" : "string" },
    { "name" : "string", "value" : "string" }
]
```

secrets

タイプ: オブジェクト配列

必須: いいえ

コンテナに公開するシークレットを表すオブジェクトです。詳細については、「Amazon ECS コンテナに機密データを渡す」を参照してください。

name

型: 文字列

必須: はい

コンテナの環境変数として設定する値。

valueFrom

型: 文字列

必須: はい

コンテナに公開するシークレット。サポートされている値は、AWS Secrets Manager シークレットの完全な Amazon リソースネーム (ARN)、または AWS Systems Manager Parameter Store 内のパラメータの完全な ARN のいずれかです。

> **Note**
>
> 起動するタスクと同じ AWS リージョン に Systems Manager パラメータストアのパラメータまたは Secrets Manager のパラメータが存在する場合は、シークレットの完全な ARN または名前のいずれかを使用できます。パラメータが別のリージョンに存在する場合は、完全な ARN を指定する必要があります。

```
"secrets": [
    {
        "name": "environment_variable_name",
        "valueFrom": "arn:aws:ssm:region:aws_account_id:parameter/parameter_name"
    }
]
```

セキュリティ

`privileged`

タイプ: ブール値

必須: いいえ

このパラメータが `true` のとき、コンテナには、ホストコンテナインスタンスに対する昇格されたアクセス権限 (`root` ユーザーと同様) が付与されます。このパラメータは docker create-container コマンドの `Privileged` にマッピングされ、`--privileged` オプションは docker run にマッピングされます。

`user`

タイプ: 文字列

必須: いいえ

コンテナ内で使用するユーザー。このパラメータは docker create-container コマンドの `User` にマッピングされ、`--user` オプションは docker run にマッピングされます。

> **Important**
>
> host ネットワークモードを使用してタスクを実行する場合、ルートユーザー (UID 0) を使用してコンテナを実行しないでください。セキュリティを強化するために、ルート以外のユーザーを使用することをお勧めします。

以下の形式を使用して、user を指定できます。UID または GID を指定する場合は、正の整数として指定する必要があります。

- user
- user:group
- uid
- uid:gid
- user:gid
- uid:group

readonlyRootFilesystem

タイプ: ブール値

必須: いいえ

このパラメータが true のとき、コンテナは読み取り専用のルートファイルシステムを付与されます。このパラメータは docker create-container コマンドの ReadonlyRootfs にマッピングされ、--read-only オプションは docker run にマッピングされます。

dockerSecurityOptions

> **Note**
>
> このパラメータは、Amazon ECS マネージドインスタンス上で実行されるタスクには対応していません。

タイプ: 文字列配列

必須: いいえ

SELinux および AppArmor のマルチレベルセキュリティシステム向けのカスタムラベルに提供する文字列のリスト。このフィールドは、Fargate を使用するタスク内のコンテナには無効です。

`ulimits`

タイプ: Ulimit オブジェクトの配列

必須: いいえ

コンテナで設定する `ulimits` のリスト。タスク定義で ulimit 値を指定すると、Docker によって設定されたデフォルト値が上書きされます。このパラメータは docker create-container コマンドの `Ulimits` にマッピングされ、`--ulimit` オプションは docker run にマッピングされます。有効な命名値は Ulimit データ型で表示されます。

Fargate でホストされる Amazon ECS タスクは、オペレーションシステムで設定されたデフォルトのリソース制限値を使用します。ただし、Fargate がオーバーライドする `nofile` リソース制限パラメータを除きます。`nofile` リソース制限は、コンテナが使用できるオープンファイルの数の制限を設定します。デフォルトの `nofile` ソフト制限は `1024` で、ハード制限は `65535` です。

このパラメータは、コンテナインスタンスで Docker Remote API バージョン 1.18 以上を使用する必要があります。コンテナインスタンスの Docker Remote API のバージョンを確認するには、コンテナインスタンスにログインし、`sudo docker version --format '{{.Server.APIVersion}}'` コマンドを実行します。

`dockerLabels`

タイプ: 文字列間のマッピング

必須: いいえ

コンテナに追加するラベルのキー/値マップ。このパラメータは docker create-container コマンドの `Labels` にマッピングされ、`--label` オプションは docker run にマッピングされます。

このパラメータは、コンテナインスタンスで Docker Remote API バージョン 1.18 以上を使用する必要があります。

```
"dockerLabels": {"string": "string"
      ...}
```

ネットワーク設定

`disableNetworking`

> **Note**
>
> このパラメータは、Amazon ECS マネージドインスタンス上で実行されるタスクには対応していません。

タイプ: ブール値

必須: いいえ

このパラメータが true のとき、ネットワークはコンテナ内でオフになります。

デフォルトは `false` です。

```
"disableNetworking": true|false
```

`links`

> **Note**
>
> このパラメータは、Amazon ECS マネージドインスタンス上で実行されるタスクには対応していません。

タイプ: 文字列配列

必須: いいえ

`link` パラメータでは、コンテナがポートマッピングを必要とせずに互いに通信することを許可します。このパラメータは、タスク定義のネットワークモードが `bridge` に設定されている場合にのみサポートされます。`name:internalName` コンストラクトは Docker リンクの `name:alias` に似ています。最大 255 文字の英字 (大文字と小文字)、数字、ハイフン、アンダースコアを使用できます。

> **Important**
>
> 同じコンテナインスタンスに配置されたコンテナ同士は、リンクやホストポートマッピングを必要とせずに、互いに通信する場合があります。コンテナインスタンスでのネットワークの分離は、セキュリティグループと VPC 設定によって制御されます。

```
"links": ["name:internalName", ...]
```

`hostname`

> **Note**
>
> このパラメータは、Amazon ECS マネージドインスタンス上で実行されるタスクには対応していません。

タイプ: 文字列

必須: いいえ

コンテナに使用するホスト名。このパラメータは docker create-container の `Hostname` にマッピングされ、`--hostname` オプションは docker run にマッピングされます。

```
"hostname": "string"
```

`dnsServers`

> **Note**
>
> このパラメータは、Amazon ECS マネージドインスタンス上で実行されるタスクには対応していません。

タイプ: 文字列配列

必須: いいえ

コンテナに提示する DNS サーバーのリスト。

```
"dnsServers": ["string", ...]
```

extraHosts

> **Note**
>
> このパラメータは、awsvpc ネットワークモードを使用するタスクではサポートされていません。

タイプ: オブジェクト配列

必須: いいえ

コンテナ上の /etc/hosts ファイルに追加する、ホスト名と IP アドレスのマッピングリスト。

このパラメータは docker create-container コマンドの ExtraHosts にマッピングされ、--add-host オプションは docker run にマッピングされます。

```
"extraHosts": [
      {
        "hostname": "string",
        "ipAddress": "string"
      }
      ...
    ]
```

hostname

タイプ: 文字列

必須: はい (extraHosts を使用する場合)

/etc/hosts エントリで使用するホスト名。

ipAddress

タイプ: 文字列

必須: はい (extraHosts を使用する場合)

/etc/hosts エントリで使用する IP アドレス。

ストレージとログ記録

`readonlyRootFilesystem`

タイプ: ブール値

必須: いいえ

このパラメータが true のとき、コンテナはそのルートファイルシステムへの読み取り専用アクセスを許可されます。このパラメータは docker create-container コマンドの `ReadonlyRootfs` にマッピングされ、`--read-only` オプションは docker run にマッピングされます。

デフォルトは `false` です。

```
"readonlyRootFilesystem": true|false
```

`mountPoints`

タイプ: オブジェクト配列

必須: いいえ

コンテナでのデータボリュームのマウントポイント。このパラメータは creat-container Docker API の `Volumes` にマッピングされ、docker run の `--volume` オプションにマッピングされます。

Windows コンテナは `$env:ProgramData` と同じドライブに全部のディレクトリをマウントできます。Windows コンテナは、別のドライブにディレクトリをマウントすることはできません。また、マウントポイントは複数のドライブにまたがることはできません。Amazon EBS ボリュームを Amazon ECS タスクに直接アタッチするには、マウントポイントを指定する必要があります。

`sourceVolume`

タイプ: 文字列

必須: はい (`mountPoints` を使用する場合)

マウントするボリュームの名前。

`containerPath`

型: 文字列

必須: はい (mountPoints を使用する場合)

ボリュームをマウントするコンテナ内のパス。

readOnly

タイプ: ブール値

必須: いいえ

この値が true の場合、コンテナはボリュームへの読み取り専用アクセスを許可されます。この値が false の場合、コンテナはボリュームに書き込むことができます。デフォルト値は false です。

Windows オペレーティングシステムを実行している EC2 インスタンスで実行されるタスクの場合、 値をデフォルト の false のままにします。

volumesFrom

タイプ: オブジェクト配列

必須: いいえ

別コンテナからマウントするデータボリューム。このパラメータは docker create-container コマンドの VolumesFrom にマッピングされ、--volumes-from オプションは docker run にマッピングされます。

sourceContainer

タイプ: 文字列

必須: はい (volumesFrom を使用する場合)

ボリュームのマウント元のコンテナの名前。

readOnly

タイプ: ブール値

必須: いいえ

この値が true の場合、コンテナはボリュームへの読み取り専用アクセスを許可されます。この値が false の場合、コンテナはボリュームに書き込むことができます。デフォルト値は false です。

```
"volumesFrom": [
```

```
            {
              "sourceContainer": "string",
              "readOnly": true|false
            }
          ]
```

`logConfiguration`

タイプ: LogConfiguration オブジェクト

必須: いいえ

コンテナに対するログ構成の仕様です。

ログ設定を使用したタスク定義の例については、「Amazon ECS のタスク定義の例」を参照してください。

このパラメータは docker create-container コマンドの `LogConfig` にマッピングされ、`--log-driver` オプションは docker run にマッピングされます。デフォルトでは、コンテナは Docker デーモンで使用されるのと同じロギングドライバーを使用します。ただし、コンテナ定義の中で、このパラメータによりログドライバーを指定することで、Docker デーモンとは異なるログドライバーをコンテナに使用させることも可能です。コンテナに異なるロギングドライバーを使用するには、コンテナインスタンス (またはリモートログ記録オプションの別のログサーバー) でログシステムを適切に設定する必要があります。

コンテナのログ設定を指定する際には、以下の点に注意してください。

- 現在、Amazon ECS では Docker デーモンに使用可能なログドライバーがいくつかサポートされています。
- このパラメータを使用するには、コンテナインスタンスで Docker Remote API のバージョン 1.18 以降が必要です。

```
"logConfiguration": {
      "logDriver": "awslogs",""splunk", "awsfirelens",
      "options": {"string": "string"
        ...},
 "secretOptions": [{
  "name": "string",
  "valueFrom": "string"
 }]
}
```

`logDriver`

型: 文字列

有効な値: `"awslogs","splunk","awsfirelens"`

必須: はい (`logConfiguration` を使用する場合)

コンテナに使用するログドライバー。デフォルトでは、上記の有効な値は Amazon ECS コンテナエージェントが通信できるログドライバーです。

サポートされているログドライバーは `awslogs`、`splunk`、`awsfirelens` です。

タスク定義で `awslogs` ログドライバーを使用してコンテナログを CloudWatch Logs に送信する方法については、「Amazon ECS ログを CloudWatch に送信する」を参照してください。

`awsfirelens` ログドライバーの使用について、詳細は「Amazon ECS ログを AWS サービスまたは AWS Partner に送信する」を参照してください。

> **Note**
>
> 上記に示されていないカスタムドライバーがある場合、GitHub で入手できる Amazon ECS コンテナエージェントプロジェクトを fork し、そのドライバーを使用するようにカスタマイズできます。含めたい変更については、プルリクエストを送信することをお勧めします。ただし、現時点では、このソフトウェアの変更されたコピーの実行はサポートされていません。

このパラメータは、コンテナインスタンスで Docker Remote API バージョン 1.18 以上を使用する必要があります。

`options`

タイプ: 文字列間のマッピング

必須: いいえ

ログドライバーに送信する設定オプションの Key-Value マップ。

指定できるオプションは、ログドライバーに応じて異なります。`awslogs` ルーターを使用して Amazon CloudWatch にログをルーティングするときに指定できるオプションには、以下が含まれます。

`awslogs-create-group`

必須: いいえ

ロググループを自動的に作成させるかどうかを指定します。このオプションを指定しない場合、デフォルトは `false` です。

> **Note**
>
> `awslogs-create-group` を使用する前に、IAM ポリシーにはアクセス許可 `logs:CreateLogGroup` が含まれている必要があります。

`awslogs-region`

必須: はい

`awslogs` ログドライバが Docker ログを送信する先の、AWS リージョン を指定します。CloudWatch Logs では、異なるリージョンのクラスターからすべてのログを 1 つのリージョンに送信するように選択できます。これにより、すべてのログを一元的な場所で確認できるようになります。他にも、リージョンごとにそれらを分離して、より細分化することが可能です。このオプションで指定するリージョンに、対象のロググループが存在することを確認してください。

`awslogs-group`

必須: はい

`awslogs` ログドライバーがログストリームを送信する先の、ロググループを指定する必要があります。

`awslogs-stream-prefix`

必須: はい

指定したプレフィックス、コンテナ名、コンテナが属する Amazon ECS タスクの ID にログストリームを関連付けるには、`awslogs-stream-prefix` オプションを使用します。このオプションでプレフィックスを指定した場合、ログストリームの形式は以下のようになります。

```
prefix-name/container-name/ecs-task-id
```

このオプションでプレフィックスを指定しない場合、ログストリームには、コンテナインスタンスの Docker デーモンによって割り当てられたコンテナ ID に基づいた名前が付けられます。Docker コンテナ ID (コンテナインスタンスでのみ使用可能) だけでそのログを送信したコンテナを追跡するのは難しいため、このオプションでプレフィックスを指定することをお勧めします。

Amazon ECS サービスでは、サービス名をプレフィックスとして使用できます。これにより、コンテナが属するサービスへのログストリームと、それを送信したコンテナの名前、そのコンテナが所属するタスクの ID を追跡できます。

Amazon ECS コンソールを使用する際に [Log] ペインにログを表示するためには、ログのストリームプレフィックスを指定する必要があります。

`awslogs-datetime-format`

必須: いいえ

このオプションは、Python `strftime` 形式で複数行起動パターンを定義します。ログメッセージは、パターンに一致する 1 行と、それに続くパターンに一致しない行で構成されます。一致する 1 行とは、ログメッセージ間の区切りです。

この形式を使用する場合のユースケースの例としては、スタックダンプなどの解析された出力があり、これを使用しなければ、複数のエントリに記録されることになります。適切なパターンにより、単一のエントリにキャプチャさせます。

詳細については、awslogs-datetime-formatを参照してください。

`awslogs-datetime-format` と `awslogs-multiline-pattern` オプションの両方を設定することはできません。

> **Note**
>
> 複数行のログ記録は、すべてのログメッセージの正規表現の解析とマッチングを実行します。これによりログ記録のパフォーマンスに悪影響が及ぶ可能性があります。

`awslogs-multiline-pattern`

必須: いいえ

このオプションでは、正規表現を使用する複数行起動パターンを定義します。ログメッセージは、パターンに一致する 1 行と、それに続くパターンに一致しない行で構成されます。一致する 1 行とは、ログメッセージ間の区切りです。

詳細については、「awslogs-multiline-pattern」を参照してください。

`awslogs-datetime-format` も設定されている場合は、このオプションは無視されます。

`awslogs-datetime-format` と `awslogs-multiline-pattern` オプションの両方を設定することはできません。

> **Note**
>
> 複数行のログ記録は、すべてのログメッセージの正規表現の解析とマッチングを実行します。これによりログ記録のパフォーマンスに悪影響が及ぶ可能性があります。

`mode`

必須: いいえ

有効な値: `non-blocking` | `blocking`

このオプションは、コンテナから `awslogs` ログドライバーへのログメッセージの配信モードを定義します。選択した配信モードは、コンテナからンテナーからのログの流れが中断されたときのアプリケーションの可用性に影響します。

`blocking` モードを使用しており、CloudWatch へのログのフローが中断されると、`stdout` ストリームと `stderr` ストリームに書き込むためのコンテナコードからの呼び出しがブロックされます。その結果、アプリケーションのロギングスレッドがブロックされます。これにより、アプリケーションが応答しなくなり、コンテナのヘルスチェックが失敗する可能性があります。

`non-blocking` モードを使用する場合、コンテナのログは代わりに `max-buffer-size` オプションで設定されたメモリ内の中間バッファに保存されます。これにより、ログを CloudWatch に送信できない場合にアプリケーションが応答しなくなるのを防ぎます。サービスの可用性を確保したいが、多少のログ損失があっても問題ない場合は、このモードを使用することをおすすめします。詳細については、「Preventing log loss with non-blocking mode in the `awslogs` container log driver」を参照してください。

`max-buffer-size`

必須: いいえ

デフォルト値: `1m`

`non-blocking` モードが使用されている場合、`max-buffer-size` ログオプションは、中間メッセージ用のストレージに使用されるバッファのサイズを制御します。アプリケーションに基づいて、必ず適切なバッファサイズを指定してください。バッファがいっぱいになると、それ以上ログを保存できなくなります。保存できないログは失われます。

`splunk` ログルーターを使用してログをルーティングするには、`splunk-token` と `splunk-url` を指定する必要があります。

ログの保存と分析のための AWS のサービス または AWS Partner Network を送信先としたログのルーティングに `awsfirelens` ログルーターを使用するときは、`log-driver-buffer-limit` オプションを設定して、イベントがログルーターコンテナに送信される前に、メモリでバッファリングされるイベントの数を制限できます。この制限は、スループットが高い場合に発生する可能性がある、Docker 内のバッファのメモリ不足による、潜在的なログ損失の問題を解決するのに役立ちます。詳細については、「the section called “高スループットのログの設定”」を参照してください。

`awsfirelens` を使用してログをルーティングするときに指定できるその他のオプションは、送信先に応じて異なります。Amazon Data Firehose にログをエクスポートするときは、`region` を使用して AWS リージョンを指定し、`delivery_stream` を使用してログストリームの名前を指定することができます。

Amazon Kinesis Data Streams にログをエクスポートするときは、`region` を使用して AWS リージョンを指定し、`stream` を使用してデータストリーム名を指定することができます。

Amazon OpenSearch Service にログをエクスポートするときは、`Name`、`Host` (プロトコルを使用しない OpenSearch Service エンドポイント)、`Port`、`Index`、`Type`、`Aws_auth`、`Aws_region`、`Suppress_Type_Name`、および `tls` といったオプションを指定できます。

Amazon S3 にログをエクスポートするときは、`bucket` オプションを使用してバケットを指定できます。また、`region`、`total_file_size`、`upload_timeout`、および `use_put_object` をオプションとして指定することもできます。

このパラメータは、コンテナインスタンスで Docker Remote API バージョン 1.19 以上を使用する必要があります。

`secretOptions`

タイプ: オブジェクト配列

必須: いいえ

ログ設定に渡すシークレットを示すオブジェクト。ログ構成で使用されるシークレットには、認証トークン、証明書、または暗号化キーを含められます。詳細については、「Amazon ECS コンテナに機密データを渡す」を参照してください。

`name`

型: 文字列

必須: はい

コンテナの環境変数として設定する値。

`valueFrom`

型: 文字列

必須: はい

コンテナのログ設定に公開するシークレット。

```
"logConfiguration": {
 "logDriver": "splunk",
 "options": {
  "splunk-url": "https://cloud.splunk.com:8080",
  "splunk-token": "...",
  "tag": "...",
  ...
 },
 "secretOptions": [{
  "name": "splunk-token",
  "valueFrom": "/ecs/logconfig/splunkcred"
 }]
}
```

`firelensConfiguration`

タイプ: FirelensConfiguration オブジェクト

必須: いいえ

コンテナの FireLens 構成。これは、コンテナログのログルーターの指定と設定に使用されます。詳細については、「Amazon ECS ログを AWS サービスまたは AWS Partner に送信する」を参照してください。

```
{
    "firelensConfiguration": {
        "type": "fluentd",
        "options": {
            "KeyName": ""
        }
    }
}
```

`options`

型: 文字列間のマッピング

必須: いいえ

ログルーターの設定時に使用するオプションの Key-Value マップ。このフィールドはオプションで、カスタム設定ファイルを指定するか、タスク、タスク定義、クラスター、コンテナインスタンスの詳細などのメタデータをログイベントに追加するために使用できます。指定した場合、使用する構文は `"options":{"enable-ecs-log-metadata":"true|false","config-file-type:"s3|file","config-file-value":"arn:aws:s3:::amzn-s3-demo-bucket/fluent.conf|filepath"}` です。詳細については、「Amazon ECS タスク定義の例: FireLens にログをルーティングする」を参照してください。

`type`

型: 文字列

必須: はい

使用するログルーター。有効な値は `fluentd` または `fluentbit` です。

リソースの要件

`resourceRequirements`

タイプ: ResourceRequirement オブジェクトの配列

必須: いいえ

コンテナに割り当てる各リソースのタイプと量。サポートされているリソースは GPU と Neuron デバイスです。

`type`

タイプ: 文字列

必須: はい

コンテナに割り当てるリソースのタイプ。サポートされる値は `GPU` と `NeuronDevice` です。

`value`

タイプ: 文字列

必須: はい

指定されたリソースタイプの値。

GPU タイプを使用する場合、値は Amazon ECS コンテナエージェントがコンテナ用に予約する物理的な GPUs の数です。完全な GPU の整数値、G6f インスタンスの小数 GPU の 10 進値 (`0.125`、`0.25`、または `0.5`)、またはコンテナインスタンス上のすべての GPU をコンテナに割り当てる `ALL` を指定できます。タスク内でコンテナ用に予約されている、すべての GPU の数は、タスクが起動されたコンテナインスタンスで使用できる GPU の数を超えることはできません。

GPU は、Fargate 上で実行するタスクでは使用できません。

`NeuronDevice` タイプを使用している場合、サポートされる値は `ALL` のみです。`ALL` を指定すると、コンテナはインスタンス上のすべての Neuron デバイスへの排他的アクセスを取得します。

Neuron デバイスは、Fargate 上で実行するタスクでは使用できません。

コンテナのタイムアウト

`startTimeout`

タイプ: 整数

必須: いいえ

値の例: 120

コンテナの依存関係解決の再試行を止めるまでの待機時間 (秒)。

例えば、タスク定義内に 2 つのコンテナを指定するとします。containerA は、COMPLETE、SUCCESS、または HEALTHY のいずれかのステータスに到達する containerB に依存関係を持ちます。startTimeout の値に containerB が指定されていて、コンテナが時間内に目標のステータスまで到達しない場合、containerA は開始しません。

> **Note**
>
> あるコンテナが依存関係における制限事項を満たさない場合、または制限を満たす前にタイムアウトした場合、Amazon ECS は、依存関係にある他のコンテナの状態も次に遷移させることはしません。

最大値は 600 秒 (10 分) です。

stopTimeout

タイプ: 整数

必須: いいえ

値の例: 120

コンテナが正常に終了しなかった場合にコンテナが強制終了されるまでの待機時間 (秒)。

このパラメータを指定しない場合には、デフォルト値の 30 秒が適用されます。最大値は 86400 秒 (24 時間) です。

コンテナの依存関係

dependsOn

型: ContainerDependency オブジェクトの配列

必須: いいえ

コンテナの起動と停止に定義されている依存関係。コンテナには複数の依存関係を含めることができます。依存関係がコンテナの起動に対して定義されている場合、コンテナの停止の場合、依存関係は逆になります。例については、コンテナの依存関係を参照してください。

> **Note**
>
> あるコンテナが依存関係における制限事項を満たさない場合、または制限を満たす前にタイムアウトした場合、Amazon ECS は、依存関係にある他のコンテナの状態も次に遷移させることはしません。

このパラメータでは、タスクまたはサービスがプラットフォームバージョン `1.3.0` 以降（Linux）または `1.0.0`（Windows）を使用している必要があります。

```
"dependsOn": [
    {
        "containerName": "string",
        "condition": "string"
    }
]
```

`containerName`

タイプ: 文字列

必須: はい

コンテナ名が指定された条件を満たしている必要があります。

`condition`

型: 文字列

必須: はい

コンテナの依存関係の条件です。使用可能な条件とその動作を以下に示します。

- `START` - この条件は、すぐに現在のリンクとボリュームの動作を工ミュレートします。この条件は、他のコンテナの開始を許可する前に、依存コンテナが開始されていることを検証します。
- `COMPLETE` - この条件は、他のコンテナの開始を許可する前に、依存コンテナの実行が完了 (終了) することを検証します。これは、スクリプトを実行して終了するだけの、重要性の低いコンテナのために便利です。この条件は、必須コンテナには設定できません。
- `SUCCESS` - この条件は `COMPLETE` と同じですが、コンテナが `zero` ステータスで終了していることも必要です。この条件は、必須コンテナには設定できません。

- HEALTHY — この条件は、他のコンテナの開始を許可する前に、依存コンテナがそのコンテナのヘルスチェックに合格したことを検証します。これには、タスク定義に設定されているヘルスチェックが依存コンテナにある必要があります。タスクの起動時にのみ、この条件が確認されます。

システムコントロール

`systemControls`

型: SystemControl オブジェクト

必須: いいえ

コンテナ内で設定する名前空間カーネルパラメータのリスト。このパラメータは docker create-container コマンドの `Sysctls` にマッピングされ、`--sysctl` オプションは docker run にマッピングされます。例えば、接続をより長く維持するように `net.ipv4.tcp_keepalive_time` 設定を構成できます。

awsvpc または host ネットワークモードも使用する単一のタスクで、複数のコンテナに対してネットワーク関連の `systemControls` パラメータを指定することは推奨されません。これを行うと、次のような欠点があります。

- `systemControls` をいずれかのコンテナ用に設定した場合、タスク内のすべてのコンテナに適用されます。単一のタスクの複数のコンテナに対して異なる `systemControls` を設定すると、最後に開始されたコンテナによって、有効になる `systemControls` が決定します。

タスク内でコンテナに使用するため IPC リソース名前空間を設定している場合、システムコントロールには以下の条件が適用されます。詳細については、「IPC モード」を参照してください。

- host IPC モードを使用するタスクの場合、IPC 名前空間の `systemControls` はサポートされていません。
- task IPC モードを使用するタスクでは、IPC 名前空間の `systemControls` 値が、タスク内のすべてのコンテナに適用されます。

```
"systemControls": [
    {
         "namespace":"string",
         "value":"string"
    }
]
```

namespace

タイプ: 文字列

必須: いいえ

value を設定する名前空間カーネルパラメータ。

有効な IPC 名前空間の値: "fs.mqueue.*" で開始する "kernel.msgmax" | "kernel.msgmnb" | "kernel.msgmni" | "kernel.sem" | "kernel.shmall" | "kernel.shmmax" | "kernel.shmmni" | "kernel.shm_rmid_forced"、および Sysctls

有効なネットワーク名前空間値: "net.*" で始まる Sysctls。Fargate では、コンテナ内に存在する名前空間 Sysctls のみが受け入れられます。

これらの値はすべて Fargate でサポートされています。

value

タイプ: 文字列

必須: いいえ

namespace で指定された名前空間カーネルパラメータの値。

インタラクティブ

interactive

タイプ: ブール値

必須: いいえ

このパラメータが true の場合、stdin または tty を割り当てる必要がある、コンテナ化されたアプリケーションをデプロイすることができます。このパラメータは docker create-container コマンドの OpenStdin にマッピングされ、--interactive オプションは docker run にマッピングされます。

デフォルトは false です。

擬似ターミナル

`pseudoTerminal`

タイプ: ブール値

必須: いいえ

このパラメータが `true` の場合、TTY が割り当てられます。このパラメータは docker create-container コマンドの `Tty` にマッピングされ、`--tty` オプションは docker run にマッピングされます。

デフォルトは `false` です。

Linux パラメータ

`linuxParameters`

型: LinuxParameters オブジェクト

必須: いいえ

Linux カーネル機能など、コンテナに適用される Linux に固有の変更です。

`capabilities`

型: KernelCapabilities オブジェクト

必須: いいえ

Docker から提供されているデフォルト構成に追加または削除される、コンテナ用の Linux 機能。

`devices`

タイプ: デバイスオブジェクト配列

必須: いいえ

コンテナに公開するすべてのホストデバイス。このパラメータは docker create-container コマンドの `Devices` にマッピングされ、`--device` オプションは docker run にマッピングされます。

`initProcessEnabled`

タイプ: ブール値

必須: いいえ

信号を転送しプロセスを利用するコンテナ内で、`init` を実行。このパラメータは docker run の `--init` オプションにマッピングされます。

`maxSwap`

> **Note**
> このパラメータは、Amazon ECS マネージドインスタンス上で実行されるタスクには対応していません。

タイプ: 整数

必須: いいえ

コンテナが使用できるスワップメモリの合計 (MiB 単位)。このパラメータは、docker run の `--memory-swap` オプションに変換されます。値はコンテナメモリの合計に `maxSwap` の値を加えた値です。

`swappiness`

> **Note**
> このパラメータは、Amazon ECS マネージドインスタンス上で実行されるタスクには対応していません。

タイプ: 整数

必須: いいえ

これにより、コンテナのメモリスワップ動作を調整できます。`swappiness` 値が `0` の場合、絶対に必要な場合を除きスワップは行われません。`swappiness` 値が `100` の場合、ページは非常に積極的にスワップされます。有効な値は、`0` と `100` の間の整数です。`swappiness` パラメータを指定しない場合、デフォルト値の `60` が使用されます。`maxSwap` の値が指定

されていない場合、このパラメータは無視されます。このパラメータは、docker run の `--memory-swappiness` オプションにマッピングされます。

`sharedMemorySize`

> **Note**
>
> このパラメータは、Amazon ECS マネージドインスタンス上で実行されるタスクには対応していません。

タイプ: 整数

必須: いいえ

`/dev/shm` ボリュームのサイズ (MiB) です。このパラメータは docker run の `--shm-size` オプションにマッピングされます。

`tmpfs`

型: Tmpfs オブジェクト配列

必須: いいえ

Tmpfs マウントのコンテナパス、マウントオプション、およびサイズ (MiB) です。このパラメータは docker run の `--tmpfs` オプションにマッピングされます。

Fargate での Amazon ECS タスク定義パラメータ

タスク定義は、タスクファミリー、AWS Identity and Access Management (IAM) タスクロール、ネットワークモード、コンテナ定義、ボリューム、および起動タイプという各部分に分かれています。ファミリとコンテナの定義は、タスク定義の必須項目です。これに対して、タスクロール、ネットワークモード、ボリューム、および起動タイプは省略することができます。

これらのパラメータを JSON ファイルで使用し、タスク定義を設定できます。

Fargate の各タスク定義パラメータについて、以下に詳述します。

ファミリー

`family`

タイプ: 文字列

必須: はい

タスク定義を登録するときに、ファミリー (複数バージョンのタスク定義の名前のようなもの) を指定する必要があります。登録したタスク定義には、リビジョン番号が与えられます。特定のファミリーに登録した最初のタスク定義には、リビジョン 1 が与えられます。その後に登録したタスク定義には、連番でリビジョン番号が与えられます。

Capacity

タスク定義の登録時、Amazon ECS がタスク定義の検証基準となる容量を指定できます。タスク定義が指定された互換性を検証しない場合、クライアント例外が返されます。

タスク定義では、以下のパラメータが使用できます。

`requiresCompatibilities`

タイプ: 文字列配列

必須: いいえ

有効な値: `FARGATE`

タスク定義が検証された容量。これにより、タスク定義で使用されているすべてのパラメータが、Fargate の要件を満たしていることの確認処理が開始されます。

タスクロール

`taskRoleArn`

タイプ: 文字列

必須: いいえ

タスク定義を登録するときに、IAM ロールのタスクロールを割り当てて、タスクのコンテナに、関連するポリシーに指定された AWS API を呼び出すためのアクセス権限を付与できます。詳細については、「Amazon ECS タスクの IAM ロール」を参照してください。

タスク実行ロール

`executionRoleArn`

タイプ: 文字列

必須: 条件による

ユーザーに代わって AWS API コールを実行するアクセス許可を Amazon ECS コンテナエージェントに付与するタスク実行ロールの Amazon リソースネーム (ARN)。

> **Note**
>
> タスク実行 IAM ロールは、タスクの要件に応じて必要です。詳細については、「Amazon ECS タスク実行IAM ロール」を参照してください。

ネットワークモード

`networkMode`

タイプ: 文字列

必須: はい

タスクのコンテナで使用する Docker ネットワークモード。Fargate でホストされている Amazon ECS タスクでは、`awsvpc` ネットワークモードが必要です。

ランタイムプラットフォーム

`operatingSystemFamily`

タイプ: 文字列

必須: 条件による

デフォルト: LINUX

Fargate でホストされる Amazon ECS タスクでは、このパラメータは必須です。

タスク定義を登録する際、オペレーティングシステムファミリを指定します。

有効な値
は、LINUX、WINDOWS_SERVER_2025_FULL、WINDOWS_SERVER_2025_CORE、WINDOWS_SERVER_2
および WINDOWS_SERVER_2019_CORE です。

サービスで使用されるすべてのタスク定義は、このパラメータに対して同じ値を設定する必要があります。

タスク定義がサービスの一部である場合、この値はサービスの platformFamily 値と一致する必要があります。

cpuArchitecture

タイプ: 文字列

必須: 条件による

デフォルト: X86_64

パラメータを null のままにすると、Fargate でホストされているタスクの開始すしたときにデフォルト値が自動的に割り当てられます。

タスク定義を登録する際は、CPU アーキテクチャを指定します。有効な値は X86_64 および ARM64 です。

サービスで使用されるすべてのタスク定義は、このパラメータに対して同じ値を設定する必要があります。

Linux タスクがある場合は、値を ARM64 に設定できます。詳細については、「the section called "64 ビット ARM ワークロードでのタスク定義"」を参照してください。

タスクサイズ

タスク定義の登録時に、そのタスクが使用する CPU とメモリの合計量を指定できます。これは、コンテナ定義レベルの cpu および memory の値とは異なります。Fargate (Linux 向けと Windows 向けの両方) でホストされるタスクの場合、これらのフィールドは必須です。また、cpu および memory の両方について、特定の値がサポートされています。

以下のパラメータをタスク定義で使用できます。

cpu

タイプ: 文字列

必須: はい

> **Note**
>
> タスクレベルの CPU およびメモリパラメータは必須であり、タスクが実行されるインスタンスタイプとサイズを判断するために使用されます。Windows タスクの場合、Windows にはコンテナのグループに集合リソース制限を簡単に適用できるネイティブメカニズムがないため、これらの値は実行時に適用されません。リソース制限を適用する場合は、Windows コンテナのコンテナレベルのリソースを使用することをお勧めします。

タスクに適用される CPU ユニットのハード制限。JSON ファイルでは、CPU 値を CPU ユニットまたは仮想 CPU (vCPU) の文字列として指定できます。例えば、CPU 値を 1 vCPU (vCPU) または 1024 (CPU ユニット) として指定できます。タスク定義が登録されると、vCPU 値は、CPU ユニットを示す整数に変換されます。

このフィールドは必須であり、次のいずれかの値を使用する必要があります。この値により memory パラメータでサポートされる値の範囲が決まります。以下の表に、タスクレベルの CPU とメモリの有効な組み合わせを示します。

CPU の値	メモリの値	AWS Fargate でサポートされるオペレーティングシステム
256 (.25 vCPU)	512 MiB、1 GB、2 GB	Linux
512 (.5 vCPU)	1 GB、2 GB、3 GB、4 GB	Linux
1,024 (1 vCPU)	2 GB、3 GB、4 GB、5 GB、6 GB、7 GB、8 GB	Linux、Windows

CPU の値	メモリの値	AWS Fargate でサポートされるオペレーティングシステム
2,048 (2 vCPU)	4 GB ～ 16 GB (1 GB のインクリメント)	Linux、Windows
4,096 (4 vCPU)	8 GB ～ 30 GB (1 GB のインクリメント)	Linux、Windows
8192 (8 vCPU) **Note** このオプションには Linux プラットフォーム 1.4.0 以降が必要です。	16 GB ~ 60 GB (4 GB のインクリメント)	Linux
16384 (16 vCPU) **Note** このオプションには Linux プラットフォーム 1.4.0 以降が必要です。	32 GB ~ 120 GB (8 GB のインクリメント)	Linux
32768 (32 vCPU) **Note** このオプションには Linux プラットフォーム 1.4.0 以降が必要です。	60 GB、120 GB、244 GB	Linux

`memory`

タイプ: 文字列

必須: はい

> **Note**
>
> タスクレベルの CPU およびメモリパラメータは必須であり、タスクが実行されるインスタンスタイプとサイズを判断するために使用されます。Windows タスクの場合、Windows にはコンテナのグループに集合リソース制限を簡単に適用できるネイティブメカニズムがないため、これらの値は実行時に適用されません。リソース制限を適用する場合は、Windows コンテナのコンテナレベルのリソースを使用することをお勧めします。

タスクに適用されるメモリのハード制限です。タスク定義のメモリの値は、メビバイト (MiB) またはギガバイト (GB) の文字列として指定できます。例えば、メモリの値を `3072` (MiB) または `3 GB` (GB) のいずれかで指定できます。タスク定義が登録されると、GB 値は、MiB を示す整数に変換されます。

このフィールドは必須であり、次のいずれかの値を使用する必要があります。この値により `cpu` パラメータでサポートされる値の範囲が決まります。

メモリ値 (単位 MiB、ほぼ同等の GB での値を付記)	CPU の値	Fargate でサポートされるオペレーティングシステム
512 (0.5 GB)、1024 (1 GB)、2048 (2 GB)	256 (.25 vCPU)	Linux
1024 (1 GB)、2048 (2 GB)、3072 (3 GB)、4096 (4 GB)	512 (.5 vCPU)	Linux
2048 (2 GB)、3072 (3 GB)、4096 (4 GB)、5120 (5 GB)、6144 (6 GB)、7168 (7 GB)、8192 (8 GB)	1,024 (1 vCPU)	Linux、Windows

メモリ値 (単位 MiB、ほぼ同等の GB での値を付記)	CPU の値	Fargate でサポートされるオペレーティングシステム
4,096 (4 GB) ~ 16,384 (16 GB) (1,024 (1 GB) 単位の増加)	2,048 (2 vCPU)	Linux、Windows
8,192 (8 GB) ~ 30,720 (30 GB) (1,024 (1 GB) 単位の増加)	4,096 (4 vCPU)	Linux、Windows
16 GB ~ 60 GB (4 GB のインクリメント) **Note** このオプションには Linux プラットフォーム `1.4.0` 以降が必要です。	8192 (8 vCPU)	Linux
32 GB ~ 120 GB (8 GB のインクリメント) **Note** このオプションには Linux プラットフォーム `1.4.0` 以降が必要です。	16384 (16vCPU)	Linux

コンテナ定義

タスク定義を登録するときは、コンテナインスタンスの Docker デーモンに渡されるコンテナ定義のリストを指定する必要があります。以下のパラメータをコンテナ定義で使用できます。

トピック

- 標準のコンテナ定義のパラメータ
- 詳細コンテナ定義パラメータ
- その他のコンテナ定義のパラメータ

標準のコンテナ定義のパラメータ

以下のタスク定義のパラメータは必須であるか、ほとんどのコンテナ定義で使用されます。

トピック

- 名前
- 画像
- メモリ
- ポートマッピング
- プライベートリポジトリの認証情報

名前

`name`

タイプ: 文字列

必須: はい

コンテナの名前。最大 255 文字の英字 (大文字と小文字)、数字、ハイフン、アンダースコアを使用できます。タスク定義で複数のコンテナをリンクしている場合、あるコンテナの `name` を別のコンテナの `links` に入力できます。これにより、コンテナ同士を接続します。

画像

`image`

タイプ: 文字列

必須: はい

コンテナの開始に使用するイメージ。この文字列は Docker デーモンに直接渡されます。デフォルトでは、Docker Hub レジストリのイメージを使用できます。*`repository-url`*`/`*`image`*`:`*`tag`*

または *repository-url*/*image*@*digest* で他のリポジトリを指定することもできます。最大 255 文字の英字 (大文字と小文字)、数字、ハイフン、アンダースコア、コロン、ピリオド、スラッシュ、シャープ記号を使用できます。このパラメータは、docker create-container コマンドの Image と Docker run コマンドの IMAGE パラメータにマッピングされます。

- 新しいタスクが開始されると、Amazon ECS コンテナエージェントは、指定されたイメージおよびタグの最新バージョンをプルしてコンテナで使用します。ただし、リポジトリイメージの後続の更新がすでに実行されているタスクに反映されることはありません。
- タスク定義のイメージパスでタグまたはダイジェストを指定しない場合、Amazon ECS コンテナエージェントは latest タグを使用して指定されたイメージを取り込みます。
- リポジトリイメージに後から加えられた更新内容が、実行中のタスクに反映されることはありません。
- プライベートレジストリのイメージがサポートされています。詳細については、「Amazon ECS での AWS 以外のコンテナイメージの使用」を参照してください。
- Amazon ECR リポジトリのイメージは、registry/repository:tag または registry/repository@digest の完全な命名規則 (例えば、*aws_account_id*.dkr.ecr.*region*.amazonaws.com/*my-web-app*:*latest* や、*aws_account_id*.dkr.ecr.*region*.amazonaws.com/*my-web-app*@*sha256:94afd1f2e64d908bc90dbca0035a5b567EXAMPLE*) を使用して指定します。
- Docker Hub の公式リポジトリのイメージでは、1 つの名前 (例: ubuntu または mongo) を使用します。
- Docker Hub の他のリポジトリのイメージは、組織名で修飾されます (例: amazon/amazon-ecs-agent)。
- 他のオンラインリポジトリのイメージは、さらにドメイン名で修飾されます (例: quay.io/assemblyline/ubuntu)。

versionConsistency

タイプ: 文字列

有効な値: enabled|disabled

必須: いいえ

コンテナ定義で指定されたコンテナイメージタグを Amazon ECS がイメージダイジェストに解決するかどうかを指定します。デフォルトでは、この動作は enabled です。コンテナの値を disabled として設定した場合、Amazon ECS はコンテナイメージタグをダイジェストに解決せ

ず、コンテナ定義で指定された元のイメージ URI をデプロイ用に使用します。コンテナイメージの解決の詳細については、「コンテナイメージの解決」を参照してください。

メモリ

`memory`

タイプ: 整数

必須: いいえ

コンテナに適用されるメモリの量 (MiB 単位)。コンテナは、ここで指定したメモリを超えようとすると、強制終了されます。タスク内のすべてのコンテナ用に予約されるメモリの合計量は、タスクの `memory` 値より小さくする必要があります (指定されている場合)。このパラメータは docker create-container コマンドの `Memory` にマッピングされ、`--memory` オプションは docker run にマッピングされます。

Docker デーモン 20.10.0 以降によって、コンテナ用として 6 MiB 以上のメモリが予約されます。従って、このコンテナに対しては 6 MiB 未満のメモリは指定しないようにします。

Docker デーモン 19.03.13-ce 以降では、コンテナ用として 4 MiB 以上のメモリが予約されます。このため、このコンテナ用には 4 MiB 未満のメモリを指定しないようにします。

> **Note**
>
> リソースの使用率を最大化することを目的に、特定のインスタンスタイプにおいて、タスクにできるだけ多くのメモリを提供する場合には、「Amazon ECS Linux コンテナインスタンスのメモリを予約する」を参照してください。

`memoryReservation`

タイプ: 整数

必須: いいえ

コンテナ用に予約するメモリのソフト制限 (MiB 単位)。システムメモリに競合がある場合、Docker はコンテナのメモリ量をこのソフト制限値内に維持しようとします。ただし、コンテナは必要に応じてより多くのメモリを使用することができます。コンテナは、`memory` パラメーターで指定されたハード制限 (該当する場合) またはコンテナインスタンス上の利用可能なすべてのメモリのいずれか早く達する方まで使用できます。このパラメータは docker create-container

コマンドの MemoryReservation にマッピングされ、--memory-reservation オプションは docker run にマッピングされます。

タスクレベルでメモリ値を指定しない場合、コンテナ定義で memory または memoryReservation の一方または両方に 0 以外の整数を指定する必要があります。両方を指定する場合、memory は memoryReservation より大きいことが必要です。memoryReservation を指定する場合、コンテナが配置されているコンテナインスタンスで使用可能なメモリリソース量から、上記の値が減算されます。それ以外の場合は、memory の値が使用されます。

例えば、コンテナが通常 128 MiB のメモリを使用しているが、短期間にメモリが 256 MiB にバーストする場合があるとします。128 MiB の memoryReservation と 300 MiB の memory ハード制限を設定できます。この設定では、コンテナが、コンテナインスタンスの残りのリソースから 128 MiB のメモリのみを予約できます。同時に、この構成により、コンテナは必要に応じてより多くのメモリ リソースを使用できるようになります。

> **Note**
>
> このパラメータは Windows コンテナではサポートされません。

Docker デーモン 20.10.0 以降によって、コンテナ用として 6 MiB 以上のメモリが予約されます。従って、このコンテナに対しては 6 MiB 未満のメモリは指定しないようにします。

Docker デーモン 19.03.13-ce 以降では、コンテナ用として 4 MiB 以上のメモリが予約されます。このため、このコンテナ用には 4 MiB 未満のメモリを指定しないようにします。

> **Note**
>
> リソースの使用率を最大化することを目的に、特定のインスタンスタイプにおいて、タスクにできるだけ多くのメモリを提供する場合には、「Amazon ECS Linux コンテナインスタンスのメモリを予約する」を参照してください。

ポートマッピング

`portMappings`

タイプ: オブジェクト配列

必須: いいえ

ポートマッピングは、コンテナのネットワークポートを外部に公開します。これは、クライアントがアプリケーションにアクセスすることを可能にします。これは、同じタスク内にあるコンテナ間の通信にも使用されます。

awsvpc ネットワークモードを使用するタスク定義では、containerPort のみを指定します。hostPort は常に無視され、コンテナポートはホスト上の番号が大きいランダムなポートに自動的にマップされます。

Windows のポートマッピングでは、NetNAT の代わりに localhost ゲートウェイを使用します。Windows のポートマッピングにはループバックが存在しないため、ホスト自体からコンテナのマッピングされたポートにアクセスはできません。

このパラメータのほとんどのフィールド (containerPort、hostPort、protocol を含む) は docker create-container コマンドの PortBindings にマッピングされ、--publish オプションは docker run にマッピングされます。タスク定義でネットワークモードを host に設定している場合、ホストポートは未定義とするか、ポートマッピングのコンテナポートと一致させる必要があります。

> **Note**
>
> タスクが RUNNING ステータスに達すると、手動および自動で割り当てられたホストとコンテナポートが次の場所で表示されます:
>
> - コンソール: 指定されたタスクのコンテナ詳細の [Network Bindings] セクション。
> - AWS CLI: networkBindings コマンド出力の describe-tasks セクション。
> - API: DescribeTasks レスポンス。
> - メタデータ: タスクメタデータのエンドポイント。

appProtocol

タイプ: 文字列

必須: いいえ

ポートマッピングに使用されるアプリケーションプロトコル。このパラメータは Service Connect にのみ適用されます。アプリケーションが使用するプロトコルと一貫性を持つよう

に、このパラメータを設定することをお勧めします。このパラメータを設定すると、Amazon ECS はサーバー コネクト プロキシにプロトコル固有の接続処理を追加します。このパラメータを設定すると、Amazon ECS は Amazon ECS コンソールと CloudWatch にプロトコル固有のテレメトリを追加します。

このパラメータ値を設定しないと、TCP が使用されます。ただし、Amazon ECS では、TCP 用のプロトコル固有のテレメトリーは追加されません。

詳細については、「the section called “ Service Connect”」を参照してください。

有効なプロトコル値: `"http" | "http2" | "grpc"`

`containerPort`

タイプ: 整数

必須: はい (`portMappings` を使用する場合)

ユーザーが指定したホストポートまたは自動的に割り当てられたホストポートにバインドされるコンテナポートの番号。

awsvpc ネットワークモードを使用するタスクの場合は、`containerPort` を使用して公開ポートを指定します。

Fargate 上の Windows コンテナの場合、`containerPort` にポート 3150 は使用できません。このポートは予約済みです。

`containerPortRange`

タイプ: 文字列

必須: いいえ

動的にマッピングされたホストポート範囲にバインドされるコンテナのポート番号の範囲。

このパラメータは、`register-task-definition` API を使用してのみ設定できます。このオプションは、`portMappings` パラメータで使用できます。詳細については、「AWS Command Line Interface リファレンス」の「register-task-definition」を参照してください。

`containerPortRange` を指定するときは、以下のルールが適用されます。

- `awsvpc` ネットワークモードを使用する必要があります。

- このパラメータは、Linux と Windows の両オペレーティングシステムで使用できます。
- コンテナインスタンスには、少なくともコンテナエージェントのバージョン 1.67.0 と `ecs-init` パッケージのバージョン 1.67.0-1 が必要です。
- 各コンテナにつき、最大 100 のポートレンジを指定できます。
- `hostPortRange` は指定しません。`hostPortRange` の値は次のように設定されます。
  - `awsvpc` ネットワークモードのタスク内のコンテナでは、`hostPort` は `containerPort` と同じ値に設定されます。これは静的マッピング戦略です。
- `containerPortRange` の有効な値は 1 ~ 65535 です。
- 1 つのポートは、コンテナごとに 1 つのポートマッピングにのみ含められます。
- 重複するポート範囲は指定できません。
- 範囲内の最初のポートは、最後のポートよりも小さくなければなりません。
- Docker では、ポートの数が多い場合は、Docker デーモン設定ファイルの Docker-proxy をオフにすることをお勧めします。

  詳細については、GitHub の Issue #11185 を参照してください。

  Docker デーモン設定ファイルの docker-proxy をオフにする方法については、Amazon ECS 開発者ガイドの「Docker デーモン」を参照してください。

DescribeTasks を呼び出すと、コンテナポートにバインドされているホストポートである `hostPortRange` を表示できます。

ポート範囲は、EventBridge に送信される Amazon ECS タスクイベントに含まれません。詳細については、「the section called “EventBridge を使用して Amazon ECS エラーへの対応を自動化する”」を参照してください。

`hostPortRange`

型: 文字列

必須: いいえ

ネットワークバインディングで使用されるホストのポート番号範囲。これは Docker によって割り当てられ、Amazon ECS エージェントによって配信されます。

`hostPort`

タイプ: 整数

必須: いいえ

コンテナ用に予約するコンテナインスタンスのポート番号。

hostPort は、空白のままにするか、containerPort と同じ値にできます。

Docker バージョン 1.6.0 以降のデフォルトの一時ポート範囲は、インスタンスの /proc/sys/net/ipv4/ip_local_port_range にリストされています。このカーネルパラメータが使用できない場合、49153-65535 から始まるデフォルトのエフェメラルポート範囲が使用されます。エフェメラルポート範囲では、ホストポートを指定しないでください。これらの範囲は自動割り当て用に予約済みです。一般的に、32768 より小さい番号のポートは一時ポート範囲に含まれません。

デフォルトの予約済みポートは、SSH 用の 22、Docker ポートの 2375 および 2376、Amazon ECS コンテナエージェントポートの 51678-51680 です。実行中のタスクに対して以前にユーザーが指定したホストポートも、タスクの実行中に予約されます。タスクが停止すると、ホストポートはリリースされます。現在の予約済みポートは、describe-container-instances 出力の remainingResources に表示されます。コンテナインスタンスには、デフォルトの予約済みポートを含めて、一度に最大 100 個の予約済みポートを割り当てられます。自動的に割り当てられるポートは、この 100 個の予約済みポートクォータにはカウントされません。

name

タイプ: 文字列

必須: いいえ。サービスに Service Connect および VPC Lattice を設定する場合は必要です。

ポートマッピングに使用される名前。このパラメータは Service Connect および VPC Lattice にのみ適用されます。このパラメータは、サービスの Service Connect および VPC Lattice 設定で使用する名前です。

詳細については、「Service Connect を使用して Amazon ECS サービスを短縮名で接続する」を参照してください。

次の例では、Service Connect および VPC Lattice の 2 つの必須フィールドが使用されています。

```
"portMappings": [
```

```
    {
        "name": string,
        "containerPort": integer
    }
]
```

`protocol`

タイプ: 文字列

必須: いいえ

ポートマッピングに使用されるプロトコル。有効な値は `tcp` および `udp` です (大文字と小文字を区別します)。デフォルトは `tcp` です。Amazon ECS は、その他の指定された値を `tcp` として扱います。

> **Important**
>
> `tcp` は Service Connect でのみサポートされます。このフィールドが設定されていない場合は、`tcp` が暗示されることに注意してください。

ホストポートを指定する際は、以下の構文を使用します。

```
"portMappings": [
    {
        "containerPort": integer,
        "hostPort": integer
    }
    ...
]
```

自動割り当てのホストポートが必要な場合は、以下の構文を使用します。

```
"portMappings": [
    {
        "containerPort": integer
    }
    ...
```

```
]
```

プライベートリポジトリの認証情報

`repositoryCredentials`

タイプ: RepositoryCredentials オブジェクト

必須: いいえ

プライベートレジストリ認証用のリポジトリ認証情報。

詳細については、「Amazon ECS での AWS 以外のコンテナイメージの使用」を参照してください。

`credentialsParameter`

タイプ: 文字列

必須: はい (`repositoryCredentials` を使用する場合)

プライベートリポジトリの認証情報が含まれているシークレットの Amazon リソースネーム (ARN)。

詳細については、「Amazon ECS での AWS 以外のコンテナイメージの使用」を参照してください。

> **Note**
>
> Amazon ECS API、AWS CLI、または AWS SDK を使用する場合、起動するタスクと同じリージョンにシークレットが存在する場合は、シークレットの完全な ARN または名前のどちらも使用できます。AWS マネジメントコンソール を使用する場合は、シークレットの完全な ARN を指定する必要があります。

必要なパラメータが含まれるタスク定義のスニペットを、以下に示します。

```
"containerDefinitions": [
    {
        "image": "private-repo/private-image",
```

```
        "repositoryCredentials": {
            "credentialsParameter":
 "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name"
        }
    }
]
```

詳細コンテナ定義パラメータ

以下のコンテナ定義用の詳細パラメータは、Amazon ECS コンテナインスタンスでのコンテナの起動に使用する Docker run コマンドの拡張機能を追加します。

トピック

- 再起動ポリシー
- ヘルスチェック
- 環境
- ネットワーク設定
- ストレージとログ記録
- セキュリティ
- リソースの制限
- Docker のラベル

再起動ポリシー

`restartPolicy`

コンテナ再起動ポリシーと関連する設定パラメータ。コンテナの再起動ポリシーを設定すると、Amazon ECS はタスクを置き換えることなくコンテナを再起動できます。詳細については、「コンテナ再起動ポリシーを使用して Amazon ECS タスク内の個々のコンテナを再起動する」を参照してください。

`enabled`

タイプ: ブール値

必須: はい

コンテナに対して再起動ポリシーを有効にするかどうかを指定します。

`ignoredExitCodes`

タイプ: 整数配列

必須: いいえ

Amazon ECS が無視し、再起動を試みない終了コードのリスト。最大 50 個のコンテナ終了コードを指定できます。デフォルトでは、Amazon ECS はいずれの終了コードも無視しません。

`restartAttemptPeriod`

タイプ: 整数

必須: いいえ

再起動を試みる前にコンテナが実行する必要がある時間 (秒単位) 。コンテナを再起動できるのは、`restartAttemptPeriod` 秒に 1 回のみです。コンテナがこの期間実行できずに早く終了した場合、コンテナは再起動されません。最小の `restartAttemptPeriod` は 60 秒、最大の `restartAttemptPeriod` は 1,800 秒を指定できます。デフォルトでは、コンテナは再起動の前に 300 秒間実行する必要があります。

ヘルスチェック

`healthCheck`

コンテナに対するヘルスチェックのコマンドと、コンテナのための関連する設定パラメータです。詳細については、「コンテナのヘルスチェックを使用して Amazon ECS タスク状態を判定する」を参照してください。

`command`

正常状態かどうかを決定するために、コンテナが実行するコマンドが格納された文字列配列。この文字列配列の先頭には、コマンド引数を直接実行するための `CMD`、またはコンテナのデフォルトシェルでコマンドを実行するための `CMD-SHELL` を付加できます。これらのいずれも指定しない場合は `CMD` が使用されます。

AWS マネジメントコンソール にタスク定義を登録する場合は、カンマで区切ったコマンドリストを使用してください。これらのコマンドは、タスク定義が作成された後に文字列に変換されます。ヘルスチェックに対する入力の例を次に示します。

```
CMD-SHELL, curl -f http://localhost/ || exit 1
```

AWS マネジメントコンソール JSON パネル、AWS CLI、または API を使用してタスク定義を登録するときは、コマンドのリストを角かっこで囲みます。ヘルスチェックに対する入力の例を次に示します。

```
[ "CMD-SHELL", "curl -f http://localhost/ || exit 1" ]
```

stderr が出力されない終了コード 0 は成功を示し、0 以外の終了コードは失敗を示します。

`interval`

各ヘルスチェック間の時間間隔 (秒単位)。5 ～ 300 秒を指定できます。デフォルトの値は 30 秒です。

`timeout`

成功まで待機しているヘルスチェックが、失敗したと見なされるまでの期間 (秒単位) です。2 ～ 60 秒を指定できます。デフォルト値は 5 秒です。

`retries`

コンテナが異常と見なされるまでに、失敗したヘルスチェックを再試行する回数です。1～10 回を指定できます。デフォルト値は 3 回の再試行です。

`startPeriod`

失敗したヘルスチェックの再試行が最大回数に達する前に、コンテナにブートストラップための時間を提供するオプションの猶予期間です。0 ~ 300 秒の値を指定できます。デフォルトでは、`startPeriod` は無効となっています。

ヘルスチェックが `startPeriod` 内で成功した場合、コンテナは正常であるとみなされ、その後の失敗は最大再試行回数にカウントされます。

環境

`cpu`

タイプ: 整数

必須: いいえ

Amazon ECS コンテナエージェントがコンテナ用に予約した cpu ユニットの数。Linux では、このパラメータは「Create a container」セクションの CpuShares にマッピングされます。

Fargate を使用するタスクでは、このフィールドは省略可能です。タスク内のすべてのコンテナのために予約されている CPU の合計量は、タスクレベルの cpu 値以上にはできません。

Linux コンテナは、割り当てられた CPU ユニットと同じ比率を使用して、割り当てられていない CPU ユニットをコンテナインスタンス上の他コンテナと共有します。例えば、そのコンテナ用に 512 個の CPU ユニットが指定された単一コアインスタンスタイプで、唯一のコンテナタスクを実行するとします。さらに、そのタスクはコンテナインスタンスで実行される唯一のタスクです。この場合のコンテナは、1,024 個の CPU ユニット配分を任意のタイミングで使用できます。一方、そのコンテナインスタンスで同じタスクのコピーを別途起動したと仮定します。各タスクには、必要に応じて少なくとも 512 個の CPU ユニットが保証されます。同様に、その他のコンテナが残りの CPU を使用していない場合、各コンテナは CPU 使用率を高められます。ただし、両方のタスクが常に 100% アクティブである場合、使用できるのは 512 CPU ユニットに制限されます。

Linux コンテナインスタンスでは、コンテナインスタンス上の Docker デーモンは、CPU 値を使用して、実行中のコンテナに対する相対 CPU 配分比率を計算します。Linux カーネルが許可する最小の有効な CPU 共有値は 2 で、Linux カーネルが許可する最大の有効な CPU 共有値は 262,144 です。ただし、CPU のパラメータは必須ではなく、コンテナ定義では 2 未満の CPU 値および 262144 より大きな CPU 値を使用できます。CPU 値が 2 未満 (null を含む) および 262144 より大きな値である場合、動作は Amazon ECS コンテナエージェントのバージョンによって異なります。

Windows コンテナインスタンスでは、CPU クォータは絶対クォータとして適用されます。Windows コンテナは、タスク定義で定義されている指定された量の CPU にのみアクセスできます。null またはゼロの CPU 値が 0 として Docker に渡されます。次に Windows はこの値を 1 つの CPU の 1% として解釈します。

その他の例については、「Amazon ECS で CPU およびメモリリソースの管理方法」を参照してください。

gpu

このパラメータは、Fargate でホストされているコンテナではサポートされていません。

タイプ: ResourceRequirement オブジェクト

必須: いいえ

Amazon ECS コンテナエージェントがコンテナ用に予約した物理 GPUs の数。タスク内でコンテナ用に予約されているすべての GPU の数は、タスクが起動されたコンテナインスタンスで使用できる GPU の数を超えることはできません。詳細については、「GPU ワークロード向けの Amazon ECS タスク定義」を参照してください。

`Elastic Inference accelerator`

このパラメータは、Fargate でホストされているコンテナではサポートされていません。

タイプ: ResourceRequirement オブジェクト

必須: いいえ

`InferenceAccelerator` タイプでは、`value` はタスク定義で指定される `InferenceAccelerator` の `deviceName` と一致します。詳細については、「the section called “Elastic Inference アクセラレーター名 (廃止)”」を参照してください。

`essential`

タイプ: ブール値

必須: いいえ

コンテナの `essential` パラメーターが `true` としてマークされており、そのコンテナが何らかの理由で失敗するか停止するとします。その後、タスクの一部である他のすべてのコンテナが停止されます。コンテナの `essential` パラメータを `false` にマークしておくと、そのコンテナ失敗はタスク内にある残りのコンテナに影響を与えません。このパラメータを省略した場合、コンテナは必須と見なされます。

すべてのタスクには少なくとも 1 つの必須コンテナが必要です。複数のコンテナで構成されるアプリケーションがあるとします。次に、共通の目的で使用されるコンテナをコンポーネントにグループ化し、さまざまなコンポーネントを複数のタスク定義に分離します。詳細については、「Amazon ECS 用のアプリケーションの構築」を参照してください。

```
"essential": true|false
```

`entryPoint`

> **Important**
>
> 初期のバージョンの Amazon ECS コンテナエージェントは、`entryPoint` パラメータを正しく処理しません。`entryPoint` の使用時に問題が発生する場合は、コンテナエー

ジェントを更新するか、代わりに command 配列項目としてコマンドと引数を入力します。

タイプ: 文字列配列

必須: いいえ

コンテナに渡されるエントリポイント。

```
"entryPoint": ["string", ...]
```

command

タイプ: 文字列配列

必須: いいえ

コンテナに渡されるコマンド。このパラメータは create-container コマンドの Cmd にマッピングされ、 COMMAND パラメータは docker run にマッピングされます。複数の引数がある場合、各引数は配列内で、区切られた文字列により指定する必要があります。

```
"command": ["string", ...]
```

workingDirectory

タイプ: 文字列

必須: いいえ

コマンドを実行するコンテナ内の作業ディレクトリ。このパラメータは、Docker Remote API の コンテナの作成 セクションにある WorkingDir にマッピングされ、--workdir オプションは docker run にマッピングされます。

```
"workingDirectory": "string"
```

environmentFiles

これは Fargate の Windows コンテナでは使用できません

タイプ: オブジェクト配列

必須: いいえ

コンテナに渡す環境変数が含まれるファイルのリスト。このパラメータは Docker run コマンドの `--env-file` オプションにマッピングされます。

最大 10 個の環境ファイルを指定できます。ファイルには、ファイル拡張子 `.env` が必要です。環境ファイルの各行には、`VARIABLE=VALUE` 形式で環境変数が含まれています。`#` で始まる行はコメントとして扱われ、無視されます。

コンテナ定義に個別の環境変数が指定されている場合は、環境ファイルに含まれる変数よりも優先されます。同じ変数を含む複数の環境ファイルが指定されている場合、それらのファイルは上から下に処理されます。一意の変数名を使用することをお勧めします。詳細については、「個々の環境変数を Amazon ECS コンテナに渡す」を参照してください。

`value`

型: 文字列

必須: はい

環境変数ファイルを含む Amazon S3 オブジェクトの Amazon リソースネーム (ARN)。

`type`

型: 文字列

必須: はい

使用するファイルのタイプ。`s3` はサポートされる唯一の値です。

`environment`

タイプ: オブジェクト配列

必須: いいえ

コンテナに渡す環境変数。このパラメータは docker create-container コマンドの `Env` にマッピングされ、`--env` オプションは docker run コマンドにマッピングされます。

> **Important**
>
> 認証情報データなどの機密情報にプレーンテキストの環境変数を使用することはお勧めしません。

name

型: 文字列

必須: はい (environment を使用する場合)

環境変数の名前。

value

型: 文字列

必須: はい (environment を使用する場合)

環境変数の値。

```
"environment" : [
    { "name" : "string", "value" : "string" },
    { "name" : "string", "value" : "string" }
]
```

secrets

タイプ: オブジェクト配列

必須: いいえ

コンテナに公開するシークレットを表すオブジェクトです。詳細については、「Amazon ECS コンテナに機密データを渡す」を参照してください。

name

型: 文字列

必須: はい

コンテナの環境変数として設定する値。

valueFrom

型: 文字列

必須: はい

コンテナに公開するシークレット。サポートされている値は、AWS Secrets Manager シークレットの完全な Amazon リソースネーム (ARN)、または AWS Systems Manager Parameter Store 内のパラメータの完全な ARN のいずれかです。

> **Note**
>
> 起動するタスクと同じ AWS リージョン に Systems Manager パラメータストアのパラメータまたは Secrets Manager のパラメータが存在する場合は、シークレットの完全な ARN または名前のいずれかを使用できます。パラメータが別のリージョンに存在する場合は、完全な ARN を指定する必要があります。

```
"secrets": [
    {
        "name": "environment_variable_name",
        "valueFrom": "arn:aws:ssm:region:aws_account_id:parameter/parameter_name"
    }
]
```

ネットワーク設定

`disableNetworking`

このパラメータは、Fargate で実行されるタスクではサポートされていません。

タイプ: ブール値

必須: いいえ

このパラメータが true のとき、ネットワークはコンテナ内でオフになります。

デフォルトは `false` です。

```
"disableNetworking": true|false
```

`links`

このパラメータは、`awsvpc` ネットワークモードを使用するタスクではサポートされません。

タイプ: 文字列配列

必須: いいえ

link パラメータでは、コンテナがポートマッピングを必要とせずに互いに通信することを許可します。このパラメータは、タスク定義のネットワークモードが bridge に設定されている場合にのみサポートされます。name:internalName コンストラクトは Docker リンクの name:alias に似ています。最大 255 文字の英字 (大文字と小文字)、数字、ハイフン、アンダースコアを使用できます。

> **Important**
> 同じコンテナインスタンスに配置されたコンテナ同士は、リンクやホストポートマッピングを必要とせずに、互いに通信する場合があります。コンテナインスタンスでのネットワークの分離は、セキュリティグループと VPC 設定によって制御されます。

```
"links": ["name:internalName", ...]
```

hostname

タイプ: 文字列

必須: いいえ

コンテナに使用するホスト名。このパラメータは docker create-container の Hostname にマッピングされ、--hostname オプションは docker run にマッピングされます。

> **Note**
> awsvpc ネットワークモードを使用している場合、hostname パラメータはサポートされません。

```
"hostname": "string"
```

dnsServers

このパラメータは、Fargate で実行されるタスクではサポートされません。

タイプ: 文字列配列

必須: いいえ

コンテナに提示する DNS サーバーのリスト。

```
"dnsServers": ["string", ...]
```

extraHosts

このパラメータは、awsvpc ネットワークモードを使用するタスクではサポートされていません。

タイプ: オブジェクト配列

必須: いいえ

コンテナ上の /etc/hosts ファイルに追加する、ホスト名と IP アドレスのマッピングリスト。

このパラメータは docker create-container コマンドの ExtraHosts にマッピングされ、--add-host オプションは docker run にマッピングされます。

```
"extraHosts": [
      {
        "hostname": "string",
        "ipAddress": "string"
      }
      ...
    ]
```

hostname

タイプ: 文字列

必須: はい (extraHosts を使用する場合)

/etc/hosts エントリで使用するホスト名。

ipAddress

タイプ: 文字列

必須: はい (extraHosts を使用する場合)

/etc/hosts エントリで使用する IP アドレス。

ストレージとログ記録

`readonlyRootFilesystem`

タイプ: ブール値

必須: いいえ

このパラメータが true のとき、コンテナはそのルートファイルシステムへの読み取り専用アクセスを許可されます。このパラメータは docker create-container コマンドの `ReadonlyRootfs` にマッピングされ、`--read-only` オプションは docker run にマッピングされます。

> **Note**
>
> このパラメータは Windows コンテナではサポートされません。

デフォルトは `false` です。

```
"readonlyRootFilesystem": true|false
```

`mountPoints`

タイプ: オブジェクト配列

必須: いいえ

コンテナでのデータボリュームのマウントポイント。このパラメータは creat-container Docker API の `Volumes` にマッピングされ、docker run の `--volume` オプションにマッピングされます。

Windows コンテナは `$env:ProgramData` と同じドライブに全部のディレクトリをマウントできます。Windows コンテナは、別のドライブにディレクトリをマウントすることはできません。また、マウントポイントは複数のドライブにまたがることはできません。Amazon EBS ボリュームを Amazon ECS タスクに直接アタッチするには、マウントポイントを指定する必要があります。

`sourceVolume`

タイプ: 文字列

必須: はい (`mountPoints` を使用する場合)

マウントするボリュームの名前。

containerPath

型: 文字列

必須: はい (mountPoints を使用する場合)

ボリュームをマウントするコンテナ内のパス。

readOnly

タイプ: ブール値

必須: いいえ

この値が true の場合、コンテナはボリュームへの読み取り専用アクセスを許可されます。この値が false の場合、コンテナはボリュームに書き込むことができます。デフォルト値は false です。

Windows オペレーティングシステムを実行している EC2 インスタンスで実行されるタスクの場合、 値をデフォルト の false のままにします。

volumesFrom

タイプ: オブジェクト配列

必須: いいえ

別コンテナからマウントするデータボリューム。このパラメータは docker create-container コマンドの VolumesFrom にマッピングされ、--volumes-from オプションは docker run にマッピングされます。

sourceContainer

タイプ: 文字列

必須: はい (volumesFrom を使用する場合)

ボリュームのマウント元のコンテナの名前。

readOnly

タイプ: ブール値

必須: いいえ

この値が `true` の場合、コンテナはボリュームへの読み取り専用アクセスを許可されます。この値が `false` の場合、コンテナはボリュームに書き込むことができます。デフォルト値は `false` です。

```
"volumesFrom": [
                {
                  "sourceContainer": "string",
                  "readOnly": true|false
                }
              ]
```

`logConfiguration`

タイプ: LogConfiguration オブジェクト

必須: いいえ

コンテナに対するログ構成の仕様です。

ログ設定を使用したタスク定義の例については、「Amazon ECS のタスク定義の例」を参照してください。

このパラメータは docker create-container コマンドの `LogConfig` にマッピングされ、`--log-driver` オプションは docker run にマッピングされます。デフォルトでは、コンテナは Docker デーモンで使用されるのと同じロギングドライバーを使用します。ただし、コンテナ定義の中で、このパラメータによりログドライバーを指定することで、Docker デーモンとは異なるログドライバーをコンテナに使用させることも可能です。コンテナに異なるロギングドライバーを使用するには、コンテナインスタンス (またはリモートログ記録オプションの別のログサーバー) でログシステムを適切に設定する必要があります。

コンテナのログ設定を指定する際には、以下の点に注意してください。

- 現在、Amazon ECS では Docker デーモンに使用可能なログドライバーがいくつかサポートされています。
- このパラメータを使用するには、コンテナインスタンスで Docker Remote API のバージョン 1.18 以降が必要です。
- 追加のソフトウェアは、タスクの外部にインストールする必要があります。例えば、Fluentd 出力アグリゲータであるか、Gelf ログの送信先として Logstash を実行しているリモートホストです。

```
"logConfiguration": {
      "logDriver": "awslogs","fluentd","gelf","json-
file","journald","splunk","syslog","awsfirelens",
      "options": {"string": "string"
        ...},
 "secretOptions": [{
  "name": "string",
  "valueFrom": "string"
 }]
}
```

`logDriver`

型: 文字列

有効な値: `"awslogs","fluentd","gelf","json-file","journald","splunk","syslog","awsfirelens"`

必須: はい (`logConfiguration` を使用する場合)

コンテナに使用するログドライバー。デフォルトでは、上記の有効な値は Amazon ECS コンテナエージェントが通信できるログドライバーです。

サポートされているログドライバーは `awslogs`、`splunk`、`awsfirelens` です。

タスク定義で `awslogs` ログドライバーを使用してコンテナログを CloudWatch Logs に送信する方法については、「Amazon ECS ログを CloudWatch に送信する」を参照してください。

`awsfirelens` ログドライバーの使用の詳細については、「カスタムログのルーティング」を参照してください。

> **Note**
>
> 上記に示されていないカスタムドライバーがある場合、GitHub で入手できる Amazon ECS コンテナエージェントプロジェクトを fork し、そのドライバーを使用するようにカスタマイズできます。含めたい変更については、プルリクエストを送信することをお勧めします。ただし、現時点では、このソフトウェアの変更されたコピーの実行はサポートされていません。

このパラメータは、コンテナインスタンスで Docker Remote API バージョン 1.18 以上を使用する必要があります。

`options`

タイプ: 文字列間のマッピング

必須: いいえ

ログドライバーに送信する設定オプションの Key-Value マップ。

指定できるオプションは、ログドライバーに応じて異なります。`awslogs` ルーターを使用して Amazon CloudWatch にログをルーティングするときに指定できるオプションには、以下が含まれます。

`awslogs-create-group`

必須: いいえ

ロググループを自動的に作成させるかどうかを指定します。このオプションを指定しない場合、デフォルトは `false` です。

> **Note**
>
> `awslogs-create-group` を使用する前に、IAM ポリシーにはアクセス許可 `logs:CreateLogGroup` が含まれている必要があります。

`awslogs-region`

必須: はい

`awslogs` ログドライバが Docker ログを送信する先の、AWS リージョン を指定します。CloudWatch Logs では、異なるリージョンのクラスターからすべてのログを 1 つのリージョンに送信するように選択できます。これにより、すべてのログを一元的な場所で確認できるようになります。他にも、リージョンごとにそれらを分離して、より細分化することが可能です。このオプションで指定するリージョンに、対象のロググループが存在することを確認してください。

`awslogs-group`

必須: はい

awslogs ログドライバーがログストリームを送信する先の、ロググループを指定する必要があります。

`awslogs-stream-prefix`

必須: はい

指定したプレフィックス、コンテナ名、コンテナが属する Amazon ECS タスクの ID にログストリームを関連付けるには、`awslogs-stream-prefix` オプションを使用します。このオプションでプレフィックスを指定した場合、ログストリームの形式は以下のようになります。

```
prefix-name/container-name/ecs-task-id
```

このオプションでプレフィックスを指定しない場合、ログストリームには、コンテナインスタンスの Docker デーモンによって割り当てられたコンテナ ID に基づいた名前が付けられます。Docker コンテナ ID (コンテナインスタンスでのみ使用可能) だけでそのログを送信したコンテナを追跡するのは難しいため、このオプションでプレフィックスを指定することをお勧めします。

Amazon ECS サービスでは、サービス名をプレフィックスとして使用できます。これにより、コンテナが属するサービスへのログストリームと、それを送信したコンテナの名前、そのコンテナが所属するタスクの ID を追跡できます。

Amazon ECS コンソールを使用する際に [Log] ペインにログを表示するためには、ログのストリームプレフィックスを指定する必要があります。

`awslogs-datetime-format`

必須: いいえ

このオプションは、Python `strftime` 形式で複数行起動パターンを定義します。ログメッセージは、パターンに一致する 1 行と、それに続くパターンに一致しない行で構成されます。一致する 1 行とは、ログメッセージ間の区切りです。

この形式を使用する場合のユースケースの例としては、スタックダンプなどの解析された出力があり、これを使用しなければ、複数のエントリに記録されることになります。適切なパターンにより、単一のエントリにキャプチャさせます。

詳細については、awslogs-datetime-formatを参照してください。

awslogs-datetime-format と awslogs-multiline-pattern オプションの両方を設定することはできません。

> **Note**
>
> 複数行のログ記録は、すべてのログメッセージの正規表現の解析とマッチングを実行します。これによりログ記録のパフォーマンスに悪影響が及ぶ可能性があります。

awslogs-multiline-pattern

必須: いいえ

このオプションでは、正規表現を使用する複数行起動パターンを定義します。ログメッセージは、パターンに一致する 1 行と、それに続くパターンに一致しない行で構成されます。一致する 1 行とは、ログメッセージ間の区切りです。

詳細については、「awslogs-multiline-pattern」を参照してください。

awslogs-datetime-format も設定されている場合は、このオプションは無視されます。

awslogs-datetime-format と awslogs-multiline-pattern オプションの両方を設定することはできません。

> **Note**
>
> 複数行のログ記録は、すべてのログメッセージの正規表現の解析とマッチングを実行します。これによりログ記録のパフォーマンスに悪影響が及ぶ可能性があります。

以下のオプションは、サポートされているすべてのログドライバーに適用されます。

mode

必須: いいえ

有効な値: non-blocking | blocking

このオプションは、コンテナから logDriver で指定したログドライバーへのログメッセージの配信モードを定義します。選択した配信モードは、コンテナからンテナーからのログの流れが中断されたときのアプリケーションの可用性に影響します。

blocking モードを使用していて、ログのフローが中断された場合、stdout ストリームと stderr ストリームに書き込むためのコンテナコードからの呼び出しがブロックされます。その結果、アプリケーションのロギングスレッドがブロックされます。これにより、アプリケーションが応答しなくなり、コンテナのヘルスチェックが失敗する可能性があります。

non-blocking モードを使用する場合、コンテナのログは代わりに max-buffer-size オプションで設定されたメモリ内の中間バッファに保存されます。これにより、ログを送信できない場合にアプリケーションが応答しなくなるのを防ぐことができます。サービスの可用性を確保したいが、多少のログ損失があっても問題ない場合は、このモードを使用することをおすすめします。詳細については、「Preventing log loss with non-blocking mode in the awslogs container log driver」を参照してください。

defaultLogDriverMode アカウント設定を使用して、特定の AWS リージョンにあるすべてのコンテナのデフォルトの mode を設定できます。logConfiguration で mode オプションを指定しない場合、またはアカウント設定を構成しない場合、Amazon ECS はデフォルトで non-blocking モードになります。アカウント設定の詳細については、「デフォルトのログドライバーモード」を参照してください。

non-blocking モードが使用されている場合、max-buffer-size ログオプションは、中間メッセージ用のストレージに使用されるバッファのサイズを制御します。アプリケーションに基づいて、必ず適切なバッファサイズを指定してください。タスクレベルで割り当てられるメモリの合計量は、ログドライバーのメモリバッファに加えて、すべてのコンテナに割り当てられたメモリ量よりも大きくなければなりません。

> **Note**
>
> 2025 年 6 月 25 日、Amazon ECS はデフォルトのログドライバーモードが blocking から non-blocking に変更され、ログ記録よりもタスクの可用性が優先されるようになりました。この変更以降も blocking モードを引き続き使用するには、次のいずれかを実行してください。
>
> - コンテナ定義の logConfiguration の mode オプションを blocking に設定します。
> - defaultLogDriverMode アカウント設定を blocking に設定します。

`max-buffer-size`

必須: いいえ

デフォルト値: `10m`

non-blocking モードが使用されている場合、`max-buffer-size` ログオプションは、中間メッセージ用のストレージに使用されるバッファのサイズを制御します。アプリケーションに基づいて、必ず適切なバッファサイズを指定してください。バッファがいっぱいになると、それ以上ログを保存できなくなります。保存できないログは失われます。

`splunk` ログルーターを使用してログをルーティングするには、`splunk-token` と `splunk-url` を指定する必要があります。

ログの保存と分析のための AWS のサービスまたは AWS Partner Networkを送信先としたログのルーティングに `awsfirelens` ログルーターを使用するときは、`log-driver-buffer-limit` オプションを設定して、イベントがログルーターコンテナに送信される前に、メモリでバッファリングされるログのライン数を制限できます。この制限は、スループットが高い場合に発生する可能性がある、Docker 内のバッファのメモリ不足による、潜在的なログ損失の問題を解決するのに役立ちます。詳細については、「the section called “高スループットのログの設定”」を参照してください。

`awsfirelens` を使用してログをルーティングするときに指定できるその他のオプションは、送信先に応じて異なります。Amazon Data Firehose にログをエクスポートするときは、`region` を使用して AWS リージョンを指定し、`delivery_stream` を使用してログストリームの名前を指定することができます。

Amazon Kinesis Data Streams にログをエクスポートするときは、`region` を使用して AWS リージョンを指定し、`stream` を使用してデータストリーム名を指定することができます。

Amazon OpenSearch Service にログをエクスポートするときは、`Name`、`Host` (プロトコルを使用しない OpenSearch Service エンドポイント)、`Port`、`Index`、`Type`、`Aws_auth`、`Aws_region`、`Suppress_Type_Name`、および `tls` といったオプションを指定できます。

Amazon S3 にログをエクスポートするときは、`bucket` オプションを使用してバケットを指定できます。また、`region`、`total_file_size`、`upload_timeout`、および `use_put_object` をオプションとして指定することもできます。

このパラメータは、コンテナインスタンスで Docker Remote API バージョン 1.19 以上を使用する必要があります。

`secretOptions`

タイプ: オブジェクト配列

必須: いいえ

ログ設定に渡すシークレットを示すオブジェクト。ログ構成で使用されるシークレットには、認証トークン、証明書、または暗号化キーを含められます。詳細については、「Amazon ECS コンテナに機密データを渡す」を参照してください。

`name`

型: 文字列

必須: はい

コンテナの環境変数として設定する値。

`valueFrom`

型: 文字列

必須: はい

コンテナのログ設定に公開するシークレット。

```
"logConfiguration": {
 "logDriver": "splunk",
 "options": {
  "splunk-url": "https://cloud.splunk.com:8080",
  "splunk-token": "...",
  "tag": "...",
  ...
 },
 "secretOptions": [{
  "name": "splunk-token",
  "valueFrom": "/ecs/logconfig/splunkcred"
 }]
}
```

`firelensConfiguration`

タイプ: FirelensConfiguration オブジェクト

必須: いいえ

コンテナの FireLens 構成。これは、コンテナログのログルーターの指定と設定に使用されます。詳細については、「Amazon ECS ログを AWS サービスまたは AWS Partner に送信する」を参照してください。

```
{
    "firelensConfiguration": {
        "type": "fluentd",
        "options": {
            "KeyName": ""
        }
    }
}
```

options

型: 文字列間のマッピング

必須: いいえ

ログルーターの設定時に使用するオプションの Key-Value マップ。このフィールドはオプションで、カスタム設定ファイルを指定するか、タスク、タスク定義、クラスター、コンテナインスタンスの詳細などのメタデータをログイベントに追加するために使用できます。指定した場合、使用する構文は `"options":{"enable-ecs-log-metadata":"true|false","config-file-type:"s3|file","config-file-value":"arn:aws:s3:::amzn-s3-demo-bucket/fluent.conf|filepath"}` です。詳細については、「Amazon ECS タスク定義の例: FireLens にログをルーティングする」を参照してください。

type

型: 文字列

必須: はい

使用するログルーター。有効な値は `fluentd` または `fluentbit` です。

セキュリティ

コンテナセキュリティの詳細については、「Amazon ECS タスクおよびコンテナのセキュリティのベストプラクティス」を参照してください。

`credentialSpecs`

タイプ: 文字列配列

必須: いいえ

Active Directory 認証用のコンテナを設定する認証情報仕様 (`CredSpec`) ファイルへの SSM または Amazon S3 の ARN のリスト。`dockerSecurityOptions` の代わりに、このパラメータを使用することをお勧めします。ARN の最大数は 1 です。

各 ARN には 2 つの形式があります。

credentialspecdomainless:MyARN

Secrets Manager にシークレットの追加セクションを持つ `CredSpec` を提供するために `credentialspecdomainless:MyARN` を使用します。シークレットでドメインへのログイン認証情報を指定します。

任意のコンテナインスタンスで実行される各タスクは、異なるドメインに参加できます。

この形式は、コンテナインスタンスをドメインに結合しなくても使用できます。

credentialspec:MyARN

単一ドメインに対して `CredSpec` を提供するために `credentialspec:MyARN` を使用します。

このタスク定義を使用するタスクを開始する前に、コンテナインスタンスをドメインに参加させる必要があります。

どちらの形式でも、SSM または Amazon S3 で `MyARN` を ARN に置き換えます。

`credspec` は、ユーザー名、パスワード、接続先のドメインを含むシークレットの ARN を Secrets Manager に提供する必要があります。セキュリティ向上のため、インスタンスはドメインレス認証のドメインに参加しません。インスタンス上の他のアプリケーションは、ドメインレス認証情報を使用できません。このパラメータを使用すると、タスクが別のドメインに参加する必要がある場合でも、同じインスタンスでタスクを実行できます。詳細については、「Windows コンテナでの gMSAs の使用」および「Linux コンテナ向け gMSAs を使用する」を参照してください。

`user`

タイプ: 文字列

必須: いいえ

コンテナ内で使用するユーザー。このパラメータは docker create-container コマンドの `User` にマッピングされ、`--user` オプションは docker run にマッピングされます。

> **Important**
>
> host ネットワークモードを使用してタスクを実行する場合は、ルートユーザー (UID 0) を使用してコンテナを実行しないでください。セキュリティのベストプラクティスとしては、常にルート以外のユーザーを使用します。

以下の形式を使用して、`user` を指定できます。UID または GID を指定する場合は、正の整数として指定する必要があります。

- `user`
- `user:group`
- `uid`
- `uid:gid`
- `user:gid`
- `uid:group`

> **Note**
>
> このパラメータは Windows コンテナではサポートされません。

```
"user": "string"
```

リソースの制限

`ulimits`

タイプ: オブジェクト配列

必須: いいえ

コンテナに定義する ulimit 値の一覧。この値は、オペレーティングシステムのデフォルトのリソースクォータ設定を上書きします。このパラメータは docker create-container コマンドの Ulimits にマッピングされ、--ulimit オプションは docker run にマッピングされます。

Fargate でホストされる Amazon ECS タスクは、オペレーションシステムで設定されたデフォルトのリソース制限値を使用します。ただし、nofile リソース制限パラメータを除きます。nofile リソース制限は、コンテナが使用できるオープンファイルの数の制限を設定します。Fargate では、デフォルトの nofile ソフト制限は　65535 で、ハード制限は 65535 です。両方の制限の値を設定できます (最大 1048576)。詳細については、「タスクリソースの制限」を参照してください。

このパラメータは、コンテナインスタンスで Docker Remote API バージョン 1.18 以上を使用する必要があります。

> **Note**
>
> このパラメータは Windows コンテナではサポートされません。

```
"ulimits": [
      {
        "name":
 "core"|"cpu"|"data"|"fsize"|"locks"|"memlock"|"msgqueue"|"nice"|"nofile"|"nproc"|"rss"|"rtp
        "softLimit": integer,
        "hardLimit": integer
      }
      ...
    ]
```

name

型: 文字列

有効な値: "core" | "cpu" | "data" | "fsize" | "locks" | "memlock" | "msgqueue" | "nice" | "nofile" | "nproc" | "rss" | "rtprio" | "rttime" | "sigpending" | "stack"

必須: はい (ulimits を使用する場合)

ulimit の type。

`hardLimit`

タイプ: 整数

必須: はい (`ulimits` を使用する場合)

`ulimit` タイプのハード制限。値は、`ulimit` の `type` に応じて、バイト、秒、またはカウントとして指定できます。

`softLimit`

タイプ: 整数

必須: はい (`ulimits` を使用する場合)

`ulimit` タイプのソフト制限。値は、`ulimit` の `type` に応じて、バイト、秒、またはカウントとして指定できます。

Docker のラベル

`dockerLabels`

タイプ: 文字列間のマッピング

必須: いいえ

コンテナに追加するラベルのキー/値マップ。このパラメータは docker create-container コマンドの `Labels` にマッピングされ、`--label` オプションは docker run にマッピングされます。

このパラメータは、コンテナインスタンスで Docker Remote API バージョン 1.18 以上を使用する必要があります。

```
"dockerLabels": {"string": "string"
      ...}
```

その他のコンテナ定義のパラメータ

以下のコンテナ定義パラメータは、[Configure via JSON] (JSON による設定) オプションを使用して、Amazon ECS コンソールでタスク定義を登録する際に使用できます。詳細については、「コンソールを使用した Amazon ECS タスク定義の作成」を参照してください。

トピック

- Linux パラメータ
- コンテナの依存関係
- コンテナのタイムアウト
- システムコントロール
- インタラクティブ
- 擬似ターミナル

Linux パラメータ

`linuxParameters`

型: LinuxParameters オブジェクト

必須: いいえ

KernelCapabilities など、コンテナに適用される Linux 固有のオプション。

> **Note**
>
> このパラメータは Windows コンテナではサポートされません。

```
"linuxParameters": {
      "capabilities": {
        "add": ["string", ...],
        "drop": ["string", ...]
        }
      }
```

`capabilities`

型: KernelCapabilities オブジェクト

必須: いいえ

Docker によって提供されているデフォルト設定から削除されたコンテナ用の Linux 機能。これらの Linux 機能の詳細については、Linux マニュアルページの「機能 (7)」を参照してください。

add

タイプ: 文字列配列

有効な値: "SYS_PTRACE"

必須：いいえ

Docker によって提供されているデフォルト設定のコンテナに追加する Linux 機能。このパラメータは docker create-container コマンドの CapAdd にマッピングされ、--cap-add オプションは docker run にマッピングされます。

drop

タイプ: 文字列配列

有効な値: "ALL" | "AUDIT_CONTROL" | "AUDIT_WRITE" | "BLOCK_SUSPEND" | "CHOWN" | "DAC_OVERRIDE" | "DAC_READ_SEARCH" | "FOWNER" | "FSETID" | "IPC_LOCK" | "IPC_OWNER" | "KILL" | "LEASE" | "LINUX_IMMUTABLE" | "MAC_ADMIN" | "MAC_OVERRIDE" | "MKNOD" | "NET_ADMIN" | "NET_BIND_SERVICE" | "NET_BROADCAST" | "NET_RAW" | "SETFCAP" | "SETGID" | "SETPCAP" | "SETUID" | "SYS_ADMIN" | "SYS_BOOT" | "SYS_CHROOT" | "SYS_MODULE" | "SYS_NICE" | "SYS_PACCT" | "SYS_PTRACE" | "SYS_RAWIO" | "SYS_RESOURCE" | "SYS_TIME" | "SYS_TTY_CONFIG" | "SYSLOG" | "WAKE_ALARM"

必須：いいえ

Docker によって提供されているデフォルト設定のコンテナから削除する Linux 機能。このパラメータは docker create-container コマンドの CapDrop にマッピングされ、--cap-drop オプションは docker run にマッピングされます。

devices

コンテナに公開するすべてのホストデバイス。このパラメータは docker create-container コマンドの Devices にマッピングされ、--device オプションは Docker run にマッピングされます。

> **Note**
>
> Fargate 起動タイプを使用する場合、devices パラメータはサポートされません。

タイプ: デバイスオブジェクト配列

必須: いいえ

`hostPath`

ホストコンテナインスタンス上のデバイスのパス。

型: 文字列

必須: はい

`containerPath`

ホストデバイスを公開する先のコンテナ内のパス。

タイプ: 文字列

必須: いいえ

`permissions`

デバイス用のコンテナに提供する明示的な許可。デフォルトでは、コンテナにはデバイスの `read`、`write`、および `mknod` のアクセス許可があります。

タイプ: 文字列の配列

有効な値: `read` | `write` | `mknod`

`initProcessEnabled`

信号を転送しプロセスを利用するコンテナ内で、`init` を実行。このパラメータは docker run の `--init` オプションにマッピングされます。

このパラメータは、コンテナインスタンスで Docker Remote API バージョン 1.25 以上を使用する必要があります。

`maxSwap`

このパラメータは、Fargate で実行されるタスクではサポートされません。

コンテナが使用できるスワップメモリの合計 (MiB 単位)。このパラメータは、docker run の `--memory-swap` オプションに変換されます。値はコンテナメモリの合計に `maxSwap` の値を加えた値です。

maxSwap の 0 値を指定した場合、コンテナはスワップを使用しません。許容値は、0 または任意の正の整数です。maxSwap パラメータを省略すると、コンテナは実行中のコンテナインスタンスのスワップ設定を使用します。maxSwap パラメータを使用するには、swappiness 値を設定する必要があります。

sharedMemorySize

/dev/shm ボリュームのサイズ値 (MiB) です。このパラメータは docker run の --shm-size オプションにマッピングされます。

> **Note**
>
> Fargate を使用するタスクを使用している場合、sharedMemorySize パラメータはサポートされません。

タイプ: 整数

tmpfs

tmpfs マウントのコンテナパス、マウントオプション、および最大サイズ (MiB) です。このパラメータは docker run の --tmpfs オプションにマッピングされます。

型: Tmpfs オブジェクト配列

必須: いいえ

containerPath

tmpfs ボリュームがマウントされる絶対ファイルパスです。

タイプ: 文字列

必須: はい

mountOptions

tmpfs ボリュームのマウントオプションのリストです。

型: 文字列の配列

必須: いいえ

有効な値: `"defaults" | "ro" | "rw" | "suid" | "nosuid" | "dev" | "nodev" | "exec" | "noexec" | "sync" | "async" | "dirsync" | "remount" | "mand" | "nomand" | "atime" | "noatime" | "diratime" | "nodiratime" | "bind" | "rbind" | "unbindable" | "runbindable" | "private" | "rprivate" | "shared" | "rshared" | "slave" | "rslave" | "relatime" | "norelatime" | "strictatime" | "nostrictatime" | "mode" | "uid" | "gid" | "nr_inodes" | "nr_blocks" | "mpol"`

`size`

tmpfs ボリュームの最大サイズ (MiB) です。

タイプ: 整数

必須: はい

コンテナの依存関係

`dependsOn`

型: ContainerDependency オブジェクトの配列

必須: いいえ

コンテナの起動と停止に定義されている依存関係。コンテナには複数の依存関係を含めることができます。依存関係がコンテナの起動に対して定義されている場合、コンテナの停止の場合、依存関係は逆になります。例については、コンテナの依存関係を参照してください。

> **Note**
>
> あるコンテナが依存関係における制限事項を満たさない場合、または制限を満たす前にタイムアウトした場合、Amazon ECS は、依存関係にある他のコンテナの状態も次に遷移させることはしません。

このパラメータでは、タスクまたはサービスがプラットフォームバージョン `1.3.0` 以降 (Linux) または `1.0.0` (Windows) を使用している必要があります。

```
"dependsOn": [
```

```
    {
        "containerName": "string",
        "condition": "string"
    }
]
```

containerName

タイプ: 文字列

必須: はい

コンテナ名が指定された条件を満たしている必要があります。

condition

型: 文字列

必須: はい

コンテナの依存関係の条件です。使用可能な条件とその動作を以下に示します。

- START - この条件は、すぐに現在のリンクとボリュームの動作を工ミュレートします。この条件は、他のコンテナの開始を許可する前に、依存コンテナが開始されていることを検証します。
- COMPLETE - この条件は、他のコンテナの開始を許可する前に、依存コンテナの実行が完了 (終了) することを検証します。これは、スクリプトを実行して終了するだけの、重要性の低いコンテナのために便利です。この条件は、必須コンテナには設定できません。
- SUCCESS - この条件は COMPLETE と同じですが、コンテナが zero ステータスで終了していることも必要です。この条件は、必須コンテナには設定できません。
- HEALTHY — この条件は、他のコンテナの開始を許可する前に、依存コンテナがそのコンテナのヘルスチェックに合格したことを検証します。これには、タスク定義に設定されているヘルスチェックが依存コンテナにある必要があります。タスクの起動時にのみ、この条件が確認されます。

コンテナのタイムアウト

startTimeout

タイプ: 整数

必須: いいえ

値の例: 120

コンテナの依存関係解決の再試行を止めるまでの待機時間 (秒)。

例えば、タスク定義内に 2 つのコンテナを指定するとします。containerA は、COMPLETE、SUCCESS、または HEALTHY のいずれかのステータスに到達する containerB に依存関係を持ちます。startTimeout の値に containerB が指定されていて、コンテナが時間内に目標のステータスまで到達しない場合、containerA は開始しません。

> **Note**
>
> あるコンテナが依存関係における制限事項を満たさない場合、または制限を満たす前にタイムアウトした場合、Amazon ECS は、依存関係にある他のコンテナの状態も次に遷移させることはしません。

このパラメータでは、タスクまたはサービスがプラットフォームバージョン 1.3.0 以降 (Linux) を使用している必要があります。最大値は 120 秒です。

stopTimeout

タイプ: 整数

必須: いいえ

値の例: 120

コンテナが正常に終了しなかった場合にコンテナが強制終了されるまでの待機時間 (秒)。

このパラメータでは、タスクまたはサービスがプラットフォームバージョン 1.3.0 以降 (Linux) を使用している必要があります。このパラメータを指定しない場合には、デフォルト値の 30 秒が適用されます。最大値は 120 秒です。

システムコントロール

systemControls

型: SystemControl オブジェクト

必須: いいえ

コンテナ内で設定する名前空間カーネルパラメータのリスト。このパラメータは docker create-container コマンドの Sysctls にマッピングされ、--sysctl オプションは docker run にマッピングされます。例えば、接続をより長く維持するように net.ipv4.tcp_keepalive_time 設定を構成できます。

awsvpc または host ネットワークモードも使用する単一のタスクで、複数のコンテナに対してネットワーク関連の systemControls パラメータを指定することは推奨されません。これを行うと、次のような欠点があります。

- systemControls をいずれかのコンテナ用に設定した場合、タスク内のすべてのコンテナに適用されます。単一のタスクの複数のコンテナに対して異なる systemControls を設定すると、最後に開始されたコンテナによって、有効になる systemControls が決定します。

タスク内でコンテナに使用するため IPC リソース名前空間を設定している場合、システムコントロールには以下の条件が適用されます。詳細については、「IPC モード」を参照してください。

- host IPC モードを使用するタスクの場合、IPC 名前空間の systemControls はサポートされていません。
- task IPC モードを使用するタスクでは、IPC 名前空間の systemControls 値が、タスク内のすべてのコンテナに適用されます。

> **Note**
>
> このパラメータは Windows コンテナではサポートされません。

> **Note**
>
> このパラメータは、プラットフォームバージョン 1.4.0 以降 (Linux) を使用する場合に、AWS Fargate でホストされたタスクでのみサポートされます。このパラメータは Fargate 上の Windows コンテナではサポートされません。

```
"systemControls": [
    {
         "namespace":"string",
         "value":"string"
```

```
    }
]
```

namespace

タイプ: 文字列

必須: いいえ

value を設定する名前空間カーネルパラメータ。

有効な IPC 名前空間の値: "fs.mqueue.*" で開始する "kernel.msgmax" | "kernel.msgmnb" | "kernel.msgmni" | "kernel.sem" | "kernel.shmall" | "kernel.shmmax" | "kernel.shmmni" | "kernel.shm_rmid_forced"、および Sysctls

有効なネットワーク名前空間値: "net.*" で始まる Sysctls。Fargate では、コンテナ内に存在する名前空間 Sysctls のみが受け入れられます。

これらの値はすべて Fargate でサポートされています。

value

タイプ: 文字列

必須: いいえ

namespace で指定された名前空間カーネルパラメータの値。

インタラクティブ

interactive

タイプ: ブール値

必須: いいえ

このパラメータが true の場合、stdin または tty を割り当てる必要がある、コンテナ化されたアプリケーションをデプロイすることができます。このパラメータは docker create-container コマンドの OpenStdin にマッピングされ、--interactive オプションは docker run にマッピングされます。

デフォルトは `false` です。

擬似ターミナル

`pseudoTerminal`

タイプ: ブール値

必須: いいえ

このパラメータが `true` の場合、TTY が割り当てられます。このパラメータは docker create-container コマンドの `Tty` にマッピングされ、`--tty` オプションは docker run にマッピングされます。

デフォルトは `false` です。

Elastic Inference アクセラレーター名 (廃止)

タスク定義用の Elastic Inference アクセラレーターのリソース要件。

> **Note**
>
> Amazon Elastic Inference (EI) は利用できなくなりました。

以下のパラメータをタスク定義で使用できます。

`deviceName`

タイプ: 文字列

必須: はい

Elastic Inference アクセラレーターのデバイス名。`deviceName` は、コンテナ定義でも参照されます。`Elastic Inference accelerator` を参照してください。

`deviceType`

タイプ: 文字列

必須: はい

使用する Elastic Inference アクセラレーター。

プロキシ設定

`proxyConfiguration`

タイプ: ProxyConfiguration オブジェクト

必須: いいえ

App Mesh プロキシ設定の詳細。

> **Note**
>
> このパラメータは Windows コンテナではサポートされません。

```
"proxyConfiguration": {
    "type": "APPMESH",
    "containerName": "string",
    "properties": [
        {
           "name": "string",
           "value": "string"
        }
    ]
}
```

`type`

型: 文字列

重要な値: APPMESH

必須: いいえ

プロキシのタイプ。APPMESH はサポートされる唯一の値です。

`containerName`

型: 文字列

必須: はい

App Mesh プロキシとして機能するコンテナの名前です。

`properties`

タイプ: パラメータオブジェクトの配列

必須: いいえ

Container Network Interface(CNI) プラグインを提供するネットワーク構成パラメータのセットで、キーと値のペアとして指定されます。

- `IgnoredUID` - (必須) コンテナ定義の `user` パラメータで定義されるプロキシコンテナのユーザー ID (UID)。これは、プロキシがそれ自体のトラフィックを無視するようにするために使用されます。`IgnoredGID` を指定した場合は、このフィールドは空にできます。
- `IgnoredGID` - (必須) コンテナ定義の `user` パラメータで定義されるプロキシコンテナのグループ ID (GID)。これは、プロキシがそれ自体のトラフィックを無視するようにするために使用されます。`IgnoredUID` を指定した場合は、このフィールドは空にできます。
- `AppPorts` - (必須) アプリケーションが使用するポートのリスト。これらのポートへのネットワークトラフィックは `ProxyIngressPort` および `ProxyEgressPort` に転送されます。
- `ProxyIngressPort` - (必須) `AppPorts` への着信トラフィックが誘導されるポートを指定します。
- `ProxyEgressPort` - (必須) `AppPorts` からの発信トラフィックが誘導されるポートを指定します。
- `EgressIgnoredPorts` – (必須) 指定されたこれらのポートに向かうアウトバウンドトラフィックは無視され、`ProxyEgressPort` にリダイレクトされません。空のリストを指定できます。
- `EgressIgnoredIPs` – (必須) 指定されたこれらの IP アドレスに向かうアウトバウンドトラフィックは無視され、`ProxyEgressPort` にリダイレクトされません。空のリストを指定できます。

`name`

型: 文字列

必須: いいえ

キーと値のペアの名前。

`value`

型: 文字列

必須: いいえ

キーと値のペアの値。

ボリューム

タスク定義を登録する際、コンテナインスタンスの Docker デーモンに渡されるボリュームのリストを任意で指定することができます。すると、同じコンテナインスタンス上の他のコンテナで使用できるようになります。

使用できるデータボリュームの種類は以下のとおりです。

- Amazon EBS ボリューム – データ集約型のコンテナ化されたワークロード向けに、費用対効果が高く、耐久性があり、高性能なブロックストレージを提供します。スタンドアロンタスクを実行するとき、またはサービスを作成または更新するときに、Amazon ECS タスクごとに 1 つの Amazon EBS ボリュームをアタッチできます。Amazon EBS ボリュームは、Fargate でホストされている Linux タスクでサポートされています。詳細については、「Amazon ECS での Amazon EBS ボリュームの使用」を参照してください。
- Amazon EFS ボリューム - Amazon ECS タスクで使用するためのシンプルかつスケーラブルで、永続的なファイルストレージを提供します。Amazon EFSでは、ストレージ容量は伸縮性があります。この容量は、ファイルの追加や削除に伴い自動的に拡大および縮小されます。アプリケーションは、必要なときに必要なストレージを確保できます。Amazon EFS ボリュームは、Fargate でホストされるタスクでサポートされます。詳細については、「Amazon ECS での Amazon EFS ボリュームの使用」を参照してください。
- FSx for Windows File Server ボリューム – 完全マネージド型の Microsoft Windows ファイルサーバーを提供します。これらのファイルサーバは、Windows ファイルシステムによってバックアップされています。Amazon ECS と共に FSx for Windows File Server を使用する場合、永続的、分散型、共有型、および静的なファイルストレージを使用して、Windows タスクをプロビジョニングすることが可能です。詳細については、「Amazon ECS での FSx for Windows File Server ボリュームの使用」を参照してください。

  このオプションは Fargate の Windows コンテナではサポートされません。
- バインドマウント – ホストマシン上のファイルやディレクトリがコンテナにマウントされます。バインドマウントホストボリュームは、タスク実行時にサポートされます。バインドマウントのホ

ストボリュームを使用するには、タスク定義で host およびオプションの sourcePath 値を使用します。

詳細については、「Amazon ECS タスクのストレージオプション」を参照してください。

以下のパラメータをコンテナ定義で使用できます。

name

タイプ: 文字列

必須: いいえ

ボリュームの名前。最大 255 文字の英字 (大文字と小文字の区別あり)、数字、ハイフン (-)、アンダースコア (_) を使用できます。この名前は、コンテナ定義 mountPoints オブジェクトの sourceVolume パラメータで参照されます。

host

必須: いいえ

host パラメーターは、バインドマウントのライフサイクルを、タスクではなくホスト Amazon EC2 インスタンスと、それが格納されている場所に関連付けるために使用されます。host パラメーターが空の場合、Docker デーモンはデータボリュームのホストパスを割り当てますが、関連付けられたコンテナの実行が停止した後にデータが保持されるとは限りません。

Windows コンテナは $env:ProgramData と同じドライブに全部のディレクトリをマウントできます。

> **Note**
>
> sourcePath パラメータは、Amazon EC2 インスタンスまたは Amazon ECS マネージドインスタンスでホストされているタスクを使用する場合にのみサポートされます。

sourcePath

タイプ: 文字列

必須: いいえ

host パラメータを使用する場合は、sourcePath を指定して、コンテナに表示されるホスト Amazon EC2 インスタンスのパスを宣言します。このパラメータが空の場合は、Docker デーモンによってホストパスが割り当てられます。host パラメータに sourcePath の場所が含まれている場合、データボリュームは手動で削除するまでホスト Amazon EC2 インスタンスの指定された場所に保持されます。sourcePath の値がホスト Amazon EC2 インスタンスに存在しない場合は、Docker デーモンによって作成されます。その場所が存在する場合は、ソースパスフォルダの内容がエクスポートされます。

configuredAtLaunch

タイプ: ブール値

必須: いいえ

起動時にボリュームを設定可能にするかどうかを指定します。true に設定すると、スタンドアロンタスクを実行するとき、またはサービスを作成または更新するときにボリュームを設定できます。true に設定すると、タスク定義内で別のボリューム設定を提供することはできません。タスクにアタッチする Amazon EBS ボリュームを設定するには、このパラメータを true に設定する必要があります。configuredAtLaunch を true に設定し、ボリューム設定を起動フェーズに先送りすることで、ボリュームタイプや特定のボリューム設定に限定されないタスク定義を作成できます。こうすることで、タスク定義をさまざまな実行環境で再利用できるようになります。詳細については、「Amazon EBS ボリューム」を参照してください。

dockerVolumeConfiguration

タイプ: DockerVolumeConfiguration オブジェクト

必須: いいえ

このパラメータは、Docker ボリュームを使用する場合に指定します。Docker ボリュームは、EC2 インスタンスでタスクを実行する場合にのみサポートされます。Windows コンテナでは、local ドライバーの使用のみがサポートされます。バインドマウントを使用するには、代わりに host を指定します。

scope

型: 文字列

有効な値: task | shared

必須: いいえ

Docker ボリュームのスコープ。これにより、ボリュームのライフサイクルが決定されます。Docker ボリュームの範囲が `task` の場合は、タスクが開始すると自動的にプロビジョンされ、タスクが停止すると破棄されます。Docker ボリュームの範囲が `shared` の場合は、タスクの停止後も保持されます。

`autoprovision`

タイプ: ブール値

デフォルト値: `false`

必須: いいえ

この値が `true` の場合、既に存在していない場合は Docker ボリュームが作成されます。このフィールドは、`scope` が `shared` の場合にのみ使用されます。`scope` が `task` の場合、このパラメータは省略する必要があります。

`driver`

タイプ: 文字列

必須: いいえ

使用する Docker ボリュームドライバー。この名前はタスク配置に使用されるため、ドライバー値は Docker で提供されているドライバー名と一致する必要があります。ドライバーが Docker プラグイン CLI を使用してインストールされた場合は、`docker plugin ls` を使用してコンテナインスタンスからドライバー名を取得します。ドライバーが別の方法でインストール済みである場合は、Docker プラグイン検出を使用してドライバー名を取得します。

`driverOpts`

タイプ: 文字列

必須: いいえ

パススルーする Docker ドライバー固有のオプションのマップ。このパラメータは、Docker の「Create a volume」セクションの `DriverOpts` にマッピングされます。

`labels`

タイプ: 文字列

必須: いいえ

Docker ボリュームに追加するカスタムメタデータ。

`efsVolumeConfiguration`

タイプ: EFSVolumeConfiguration オブジェクト

必須: いいえ

このパラメータは、Amazon EFS ボリュームを使用する場合に指定します。

`fileSystemId`

型: 文字列

必須: はい

使用する Amazon EFS ファイルシステムの ID。

`rootDirectory`

型: 文字列

必須: いいえ

ホスト内にルートディレクトリとしてマウントする Amazon EFS ファイルシステム内のディレクトリ。このパラメータを省略すると、Amazon EFS ボリュームのルートが使用されます。/ を指定すると、このパラメータを省略した場合と同じ結果になります。

> **Important**
>
> `authorizationConfig` で EFS アクセスポイントが指定されている場合は、ルートディレクトリパラメータを省略するか / に設定して、EFS アクセスポイントにパスを設定する必要があります。

`transitEncryption`

タイプ: 文字列

有効な値: `ENABLED` | `DISABLED`

必須: いいえ

Amazon ECS ホストと Amazon EFS サーバー間で、転送中の Amazon EFS データの暗号化を有効にするかどうかを指定します。Amazon EFS IAM 認可を使用する場合は、転送中の暗号化を有効にする必要があります。このパラメータを省略すると、`DISABLED` のデフォル

ト値が使用されます。詳細については、Amazon Elastic ファイルシステムユーザーガイドの「転送中データの暗号化」を参照してください。

`transitEncryptionPort`

タイプ: 整数

必須: いいえ

Amazon ECS ホストと Amazon EFS サーバーとの間で、暗号化されたデータを送信するときに使用するポート。転送暗号化ポートを指定しない場合、タスクでは Amazon EFS マウントヘルパーが使用するポート選択戦略が使用されます。詳細については、[Amazon Elastic File System User Guide] (Amazon Elastic File System ユーザーガイド) の[EFS Mount Helper] (EFS マウントヘルパー) を参照してください。

`authorizationConfig`

タイプ: EFSAuthorizationConfig オブジェクト

必須: いいえ

Amazon EFS ファイルシステムに対する認可構成の詳細。

`accessPointId`

型: 文字列

必須: いいえ

使用するアクセスポイント ID。アクセスポイントが指定されている場合は、`efsVolumeConfiguration` のルートディレクトリ値を省略するか `/` に設定して、EFS アクセスポイントにパスを設定する必要があります。アクセスポイントを使用する場合は、`EFSVolumeConfiguration` で転送中の暗号化を有効にする必要があります。詳細については、Amazon Elastic ファイルシステムユーザーガイドの Amazon EFS アクセスポイントの使用を参照してください。

`iam`

型: 文字列

有効な値: `ENABLED` | `DISABLED`

必須: いいえ

タスク定義で定義した Amazon ECS タスクの IAM ロールを、Amazon EFS ファイルシステムのマウント時に使用するかどうかを指定します。使用する場合

は、EFSVolumeConfiguration で転送中の暗号化を有効にする必要があります。このパラメータを省略すると、DISABLED のデフォルト値が使用されます。詳細については、「タスク用の IAM ロール」を参照してください。

FSxWindowsFileServerVolumeConfiguration

タイプ: FSxWindowsFileServerVolumeConfiguration オブジェクト

必須: はい

このパラメータは、タスクストレージに Amazon FSx for Windows File Server ファイルシステムを使用する場合に指定します。

fileSystemId

タイプ: 文字列

必須: はい

使用する FSx for Windows File Server ファイルシステムID。

rootDirectory

型: 文字列

必須: はい

ホスト内にルートディレクトリとしてマウントする FSx for Windows File Server ファイルシステム内のディレクトリ。

authorizationConfig

credentialsParameter

型: 文字列

必須: はい

認可の認証情報オプション。

options:

- AWS Secrets Manager シークレットの Amazon リソースネーム (ARN)。
- AWS Systems Manager パラメータの ARN。

domain

タイプ: 文字列

必須: はい

AWS Directory Service for Microsoft Active Directory (AWS Managed Microsoft AD) ディレクトリまたはセルフホスト型 EC2 Active Directory によってホストされる完全修飾ドメイン名。

タグ

タスク定義の登録する際、タスク定義に適用されるメタデータタグをオプションで指定できます。タグは、タスク定義を分類して組織化するのに役立ちます。各タグはキーとオプションの値で構成されます。両方を定義します。詳細については、「Amazon ECS リソースにタグ付けする」を参照してください。

> **Important**
>
> タグには、個人が特定可能な情報や、機密情報あるいは秘匿性の高い情報は追加しないでください。タグは、多くの AWS のサービス (請求など) からアクセスできます。タグは、プライベートデータや機密データに使用することを意図していません。

タグオブジェクトでは、次のパラメータを使用できます。

`key`

タイプ: 文字列

必須: いいえ

タグを構成するキーと値のペアの一部。キーは、より具体的なタグ値のカテゴリのように動作する、一般的なラベルです。

`value`

タイプ: 文字列

必須: いいえ

タグを構成するキーと値のペアのオプションの一部。値はタグカテゴリ (キー) の記述子として機能します。

その他のタスク定義パラメータ

以下のタスク定義パラメータは、Amazon ECS コンソールから、[Configure via JSON (JSON による設定)] オプションを使用してタスク定義を登録する際に使用します。詳細については、「コンソールを使用した Amazon ECS タスク定義の作成」を参照してください。

トピック

- エフェメラルストレージ
- IPC モード
- PID モード
- フォールトインジェクション

エフェメラルストレージ

`ephemeralStorage`

タイプ: EphemeralStorage オブジェクト

必須: いいえ

タスクに割り当てるエフェメラルストレージの容量(GB 単位)。このパラメーターは、AWS Fargate でホストされるタスクにおいて、利用可能なエフェメラルストレージの総量をデフォルトの容量を超えて拡張する際に使用します。詳細については、「the section called “バインドマウント”」を参照してください。

> **Note**
>
> このパラメータは、プラットフォームバージョン `1.4.0` 以降 (Linux) または `1.0.0` 以降 (Windows) でのみサポートされます。

IPC モード

`ipcMode`

このパラメータは、Fargate で実行されるタスクではサポートされません。

タイプ: 文字列

必須: いいえ

タスクのコンテナで使用する IPC リソースの名前空間。有効な値は host、task または none です。host が指定されている場合、同一のコンテナインスタンス上にある (host IPC モードを指定した) タスク内のすべてのコンテナは、ホスト Amazon EC2 インスタンスと同じ IPC リソースを共有します。task が指定されている場合、指定されたタスク内のすべてのコンテナは同じ IPC リソースを共有します。none が指定されている場合、タスクのコンテナ内の IPC リソースはプライベートです。タスク内またはコンテナインスタンスの他のコンテナと共有されることはありません。値を指定しない場合、IPC リソース名前空間の共有はコンテナインスタンスの Docker デーモンの設定によって異なります。

host IPC モードを使用する場合は、意図せず IPC 名前空間が公開されるリスクが高いことに注意してください。

タスク内のコンテナに、systemControls を使用して名前空間のカーネルパラメータを設定している場合は、以下の点が IPC リソース名前空間に適用されます。

- host IPC モードを使用するタスクの場合、systemControls に関連する IPC 名前空間はサポートされません。
- task IPC モードを使用するタスクでは、IPC 名前空間に関連する systemControls が、タスク内のすべてのコンテナに適用されます。

> **Note**
>
> このパラメータは、Windows コンテナ、または Fargate 起動タイプを使用するタスクではサポートされていません。

PID モード

pidMode

タイプ: 文字列

有効な値: host | task

必須: いいえ

タスクのコンテナで使用するプロセス名前空間。有効な値は host または task です。Linuxコンテナでは、有効な値は task のみです。例えば、サイドカーのモニタリングでは、pidMode が

同じタスクで実行されている他のコンテナに関する情報にアクセスする必要となる場合があります。

`task` が指定されている場合、指定したタスク内のすべてのコンテナは同じプロセス名前空間を共有します。

値が指定されていない場合、デフォルトは各コンテナのプライベート名前空間です。

> **Note**
>
> このパラメータは、プラットフォームバージョン `1.4.0` 以降 (Linux) を使用する場合に、AWS Fargate でホストされたタスクでのみサポートされます。このパラメータは Fargate 上の Windows コンテナではサポートされません。

フォールトインジェクション

`enableFaultInjection`

タイプ: ブール値

有効な値: `true` | `false`

必須: いいえ

このパラメータが `true` に設定されている場合、タスクのペイロードで、Amazon ECS と Fargate はタスクのコンテナからのフォールトインジェクションリクエストを受け入れます。デフォルトでは、このパラメータは `false` に設定されます。

Amazon EC2 における Amazon ECS タスク定義パラメータ

タスク定義は、タスクファミリー、AWS Identity and Access Management (IAM) タスクロール、ネットワークモード、コンテナ定義、ボリューム、タスク配置の制約、容量に分類されます。ファミリとコンテナの定義は、タスク定義の必須項目です。これに対して、タスクロール、ネットワークモード、ボリューム、タスク配置の制約、起動タイプは省略することができます。

これらのパラメータを JSON ファイルで使用し、タスク定義を設定できます。

Amazon EC2 の各タスク定義パラメータについて、以下に詳述します。

ファミリー

`family`

タイプ: 文字列

必須: はい

タスク定義を登録するときに、ファミリー (複数バージョンのタスク定義の名前のようなもの) を指定する必要があります。登録したタスク定義には、リビジョン番号が与えられます。特定のファミリーに登録した最初のタスク定義には、リビジョン 1 が与えられます。その後に登録したタスク定義には、連番でリビジョン番号が与えられます。

Capacity

タスク定義の登録時、Amazon ECS がタスク定義の検証基準となる容量を指定できます。タスク定義が指定された互換性を検証しない場合、クライアント例外が返されます。

タスク定義では、以下のパラメータが使用できます。

`requiresCompatibilities`

タイプ: 文字列配列

必須: いいえ

有効な値: `EC2`

タスク定義が検証された容量。これにより、タスク定義で使用されているすべてのパラメータが、Amazon EC2 の要件を満たしていることの確認処理が開始されます。

タスクロール

`taskRoleArn`

タイプ: 文字列

必須: いいえ

タスク定義を登録するときに、IAM ロールのタスクロールを割り当てて、タスクのコンテナに、関連するポリシーに指定された AWS API を呼び出すためのアクセス権限を付与できます。詳細については、「Amazon ECS タスクの IAM ロール」を参照してください。

Amazon ECS に最適化された Windows Server AMI を起動する場合、Windows タスク向けの IAM ロールでは、`-EnableTaskIAMRole` オプションが設定されている必要があります。また、コンテナでは、この機能を利用するために一部の構成コードを実行する必要があります。詳細については、「Amazon EC2 Windows インスタンスの追加設定」を参照してください。

タスク実行ロール

`executionRoleArn`

タイプ: 文字列

必須: 条件による

ユーザーに代わって AWS API コールを実行するアクセス許可を Amazon ECS コンテナエージェントに付与するタスク実行ロールの Amazon リソースネーム (ARN)。

> **Note**
>
> タスク実行 IAM ロールは、タスクの要件に応じて必要です。詳細については、「Amazon ECS タスク実行IAM ロール」を参照してください。

ネットワークモード

`networkMode`

タイプ: 文字列

必須: いいえ

タスクのコンテナで使用する Docker ネットワークモード。Amazon EC2 Linux インスタンスでホストされている Amazon ECS タスクの場合、有効な値は `none`、`bridge`、`awsvpc`、`host` です。ネットワークモードが指定されていない場合、デフォルトのネットワークモードは `bridge` です。Amazon EC2 Windows インスタンスでホストされる Amazon ECS タスクの場合、有効な値は `default` と `awsvpc` です。ネットワークモードが指定されていない場合、`default` ネットワークモードが使用されます。

ネットワークモードに none を設定した場合、タスクのコンテナは外部への接続を持たないため、コンテナ定義でポートマッピングを指定することはできません。

ネットワークモードが bridge の場合、タスクはそのタスクをホストする各 Amazon EC2 インスタンス内で実行される、Linux 上の Docker の組み込み仮想ネットワークを使用します。Linux の組み込み仮想ネットワークでは、bridge Docker ネットワークドライバーが使用されます。

ネットワークモードが host の場合、タスクはそのタスクをホストする Amazon EC2 インスタンスの ENI にコンテナポートを直接マッピングすることで Docker の組み込み仮想ネットワークをバイパスする、ホストのネットワークを使用します。ダイナミックポートマッピングは、このネットワークモードでは使用できません。このモードを使用するタスク定義内のコンテナには、特定の hostPort 番号を指定する必要があります。ホストのポート番号は、複数のタスクで使用できません。その結果として、1 つの Amazon EC2 インスタンスで同じタスク定義のタスクを複数実行することはできません。

> **Important**
>
> host ネットワークモードをによりタスクを実行する場合、より良いセキュリティーのために、ルートユーザー (UID 0) を使用してコンテナを実行しないでください。セキュリティのベストプラクティスとしては、常にルート以外のユーザーを使用します。

ネットワークモードが awsvpc の場合は、タスクに Elastic Network Interface が割り当てられるため、タスク定義を使用したサービスの作成時またはタスクの実行時に NetworkConfiguration を指定する必要があります。詳細については、「EC2 の Amazon ECS タスクネットワークオプション」を参照してください。

ネットワークモードが default の場合、タスクはそのタスクをホストする各 Amazon EC2 インスタンス内で実行される、Windows 上の Docker の組み込み仮想ネットワークを使用します。Windows の組み込み仮想ネットワークは nat Docker ネットワークドライバーを使用します。

host および awsvpc のネットワークモードは、Amazon EC2 のネットワークスタックを使用するので、コンテナのネットワークパフォーマンスが最大限発揮されます。host および awsvpc ネットワークモードにおいて公開されるコンテナポートは、対応するホストポート (host ネットワークモードの場合)、またはアタッチされた Elastic Network Interface ポート (awsvpc ネットワークモードの場合) に直接マッピングされます。このため、動的ホストポートマッピングは使用できません。

使用可能なネットワークモードは、基板となる EC2 インスタンスのオペレーティングシステムによって異なります。Linux の場合は、どのネットワークモードも使用できます。Windows の場合、default およびawsvpc モードを使用できます。

ランタイムプラットフォーム

operatingSystemFamily

タイプ: 文字列

必須: 条件による

デフォルト: LINUX

タスク定義を登録する際、オペレーティングシステムファミリを指定します。

有効な値
は、LINUX、WINDOWS_SERVER_2025_FULL、WINDOWS_SERVER_2025_CORE、WINDOWS_SERVER_2
および
WINDOWS_SERVER_2019_CORE、WINDOWS_SERVER_2016_FULL、WINDOWS_SERVER_2004_CORE、
および WINDOWS_SERVER_20H2_CORE です。

サービスで使用されるすべてのタスク定義は、このパラメータに対して同じ値を設定する必要があります。

タスク定義がサービスの一部である場合、この値はサービスの platformFamily 値と一致する必要があります。

cpuArchitecture

タイプ: 文字列

必須: 条件による

タスク定義を登録する際は、CPU アーキテクチャを指定できます。有効な値は X86_64 および ARM64 です。値を指定しない場合、Amazon ECS はキャパシティプロバイダーの設定に基づいて、使用可能な CPU アーキテクチャにタスクを配置しようとします。タスクが特定の CPU アーキテクチャに配置されるようにするには、タスク定義で cpuArchitecture の値を指定します。

サービスで使用されるすべてのタスク定義は、このパラメータに対して同じ値を設定する必要があります。

Linux タスクがある場合は、値を ARM64 に設定できます。詳細については、「the section called "64 ビット ARM ワークロードでのタスク定義"」を参照してください。

タスクサイズ

タスク定義の登録時に、そのタスクが使用する CPU とメモリの合計量を指定できます。これは、コンテナ定義レベルの cpu および memory の値とは異なります。Amazon EC2 インスタンスでホストされるタスクの場合、これらのフィールドは省略可能です。

> **Note**
>
> タスクレベル CPU およびメモリのパラメータは Windows コンテナでは無視されます。Windows コンテナではコンテナレベルリソースを指定することをお勧めします。

`cpu`

タイプ: 文字列

必須: 条件による

> **Note**
>
> このパラメータは Windows コンテナではサポートされません。

タスクに適用される CPU ユニットのハード制限。JSON ファイルでは、CPU 値を CPU ユニットまたは仮想 CPU (vCPU) の文字列として指定できます。例えば、CPU 値を 1 vCPU (vCPU) または 1024 (CPU ユニット) として指定できます。タスク定義が登録されると、vCPU 値は、CPU ユニットを示す整数に変換されます。

このフィールドはオプションです。クラスターに、要求された CPU ユニットが利用できる登録済みのコンテナインスタンスがない場合、タスクは失敗します。サポートされている値は、0.125 vCPU から 192 vCPU までです。

`memory`

タイプ: 文字列

必須: 条件による

> **Note**
>
> このパラメータは Windows コンテナではサポートされません。

タスクに適用されるメモリのハード制限です。タスク定義のメモリの値は、メビバイト (MiB) またはギガバイト (GB) の文字列として指定できます。例えば、メモリの値を 3072 (MiB) または 3 GB (GB) のいずれかで指定できます。タスク定義が登録されると、GB 値は、MiB を示す整数に変換されます。

このフィールドはオプションです。任意の値を使用できます。タスクレベルでメモリ値が指定されている場合、コンテナレベルのメモリ値の設定は省略可能です。クラスターに、要求されたメモリが利用できる登録済みのコンテナインスタンスがない場合、タスクは失敗します。特定のインスタンスタイプについて、タスクにできるだけ多くのメモリを提供することで、リソースの使用率を最大化することができます。詳細については、「Amazon ECS Linux コンテナインスタンスのメモリを予約する」を参照してください。

コンテナ定義

タスク定義を登録するときは、コンテナインスタンスの Docker デーモンに渡されるコンテナ定義のリストを指定する必要があります。以下のパラメータをコンテナ定義で使用できます。

トピック

- 標準のコンテナ定義のパラメータ
- 詳細コンテナ定義パラメータ
- その他のコンテナ定義のパラメータ

標準のコンテナ定義のパラメータ

以下のタスク定義のパラメータは必須であるか、ほとんどのコンテナ定義で使用されます。

トピック

- 名前
- 画像
- メモリ

- ポートマッピング
- プライベートリポジトリの認証情報

名前

name

タイプ: 文字列

必須: はい

コンテナの名前。最大 255 文字の英字 (大文字と小文字)、数字、ハイフン、アンダースコアを使用できます。タスク定義で複数のコンテナをリンクしている場合、あるコンテナの name を別のコンテナの links に入力できます。これにより、コンテナ同士を接続します。

画像

image

タイプ: 文字列

必須: はい

コンテナの開始に使用するイメージ。この文字列は Docker デーモンに直接渡されます。デフォルトでは、Docker Hub レジストリのイメージを使用できます。*repository-url*/*image*:*tag* または *repository-url*/*image*@*digest* で他のリポジトリを指定することもできます。最大 255 文字の英字 (大文字と小文字)、数字、ハイフン、アンダースコア、コロン、ピリオド、スラッシュ、シャープ記号を使用できます。このパラメータは、docker create-container コマンドの Image と Docker run コマンドの IMAGE パラメータにマッピングされます。

- 新しいタスクが開始されると、Amazon ECS コンテナエージェントは、指定されたイメージおよびタグの最新バージョンをプルしてコンテナで使用します。ただし、リポジトリイメージの後続の更新がすでに実行されているタスクに反映されることはありません。
- タスク定義のイメージパスでタグまたはダイジェストを指定しない場合、Amazon ECS コンテナエージェントは latest タグを使用して指定されたイメージを取り込みます。
- リポジトリイメージに後から加えられた更新内容が、実行中のタスクに反映されることはありません。
- プライベートレジストリのイメージがサポートされています。詳細については、「Amazon ECS での AWS 以外のコンテナイメージの使用」を参照してください。

- Amazon ECR リポジトリのイメージは、`registry/repository:tag` または `registry/repository@digest` の完全な命名規則 (例えば、*`aws_account_id`*`.dkr.ecr.`*`region`*`.amazonaws.com/`*`my-web-app`*`:`*`latest`* や、*`aws_account_id`*`.dkr.ecr.`*`region`*`.amazonaws.com/`*`my-web-app`*`@`*`sha256:94afd1f2e64d908bc90dbca0035a5b567EXAMPLE`*) を使用して指定します。
- Docker Hub の公式リポジトリのイメージでは、1 つの名前 (例: `ubuntu` または `mongo`) を使用します。
- Docker Hub の他のリポジトリのイメージは、組織名で修飾されます (例: `amazon/amazon-ecs-agent`)。
- 他のオンラインリポジトリのイメージは、さらにドメイン名で修飾されます (例: `quay.io/assemblyline/ubuntu`)。

`versionConsistency`

タイプ: 文字列

有効な値: `enabled|disabled`

必須: いいえ

コンテナ定義で指定されたコンテナイメージタグを Amazon ECS がイメージダイジェストに解決するかどうかを指定します。デフォルトでは、この動作は `enabled` です。コンテナの値を `disabled` として設定した場合、Amazon ECS はコンテナイメージタグをダイジェストに解決せず、コンテナ定義で指定された元のイメージ URI をデプロイ用に使用します。コンテナイメージの解決の詳細については、「コンテナイメージの解決」を参照してください。

メモリ

`memory`

タイプ: 整数

必須: いいえ

コンテナに適用されるメモリの量 (MiB 単位)。コンテナは、ここで指定したメモリを超えようとすると、強制終了されます。タスク内のすべてのコンテナ用に予約されるメモリの合計量は、タスクの `memory` 値より小さくする必要があります (指定されている場合)。このパラメータは docker create-container コマンドの `Memory` にマッピングされ、`--memory` オプションは docker run にマッピングされます。

タスクレベルのメモリ値またはコンテナレベルのメモリ値を指定する必要があります。memory コンテナレベルと memoryReservation 値の両方を指定する場合、memory 値は memoryReservation 値より大きくする必要があります。memoryReservation を指定する場合、コンテナが配置されているコンテナインスタンスで使用可能なメモリリソース量から、上記の値が減算されます。それ以外の場合は、memory の値が使用されます。

Docker デーモン 20.10.0 以降によって、コンテナ用として 6 MiB 以上のメモリが予約されます。従って、このコンテナに対しては 6 MiB 未満のメモリは指定しないようにします。

Docker デーモン 19.03.13-ce 以降では、コンテナ用として 4 MiB 以上のメモリが予約されます。このため、このコンテナ用には 4 MiB 未満のメモリを指定しないようにします。

> **Note**
>
> リソースの使用率を最大化することを目的に、特定のインスタンスタイプにおいて、タスクにできるだけ多くのメモリを提供する場合には、「Amazon ECS Linux コンテナインスタンスのメモリを予約する」を参照してください。

`memoryReservation`

タイプ: 整数

必須: いいえ

コンテナ用に予約するメモリのソフト制限 (MiB 単位)。システムメモリに競合がある場合、Docker はコンテナのメモリ量をこのソフト制限値内に維持しようとします。ただし、コンテナは必要に応じてより多くのメモリを使用することができます。コンテナは、memory パラメーターで指定されたハード制限 (該当する場合) またはコンテナインスタンス上の利用可能なすべてのメモリのいずれか早く達する方まで使用できます。このパラメータは docker create-container コマンドの MemoryReservation にマッピングされ、--memory-reservation オプションは docker run にマッピングされます。

タスクレベルでメモリ値を指定しない場合、コンテナ定義で memory または memoryReservation の一方または両方に 0 以外の整数を指定する必要があります。両方を指定する場合、memory は memoryReservation より大きいことが必要です。memoryReservation を指定する場合、コンテナが配置されているコンテナインスタンスで使用可能なメモリリソース量から、上記の値が減算されます。それ以外の場合は、memory の値が使用されます。

例えば、コンテナが通常 128 MiB のメモリを使用しているが、短期間にメモリが 256 MiB にバーストする場合があるとします。128 MiB の `memoryReservation` と 300 MiB の `memory` ハード制限を設定できます。この設定では、コンテナが、コンテナインスタンスの残りのリソースから 128 MiB のメモリのみを予約できます。同時に、この構成により、コンテナは必要に応じてより多くのメモリ リソースを使用できるようになります。

> **Note**
>
> このパラメータは Windows コンテナではサポートされません。

Docker デーモン 20.10.0 以降によって、コンテナ用として 6 MiB 以上のメモリが予約されます。従って、このコンテナに対しては 6 MiB 未満のメモリは指定しないようにします。

Docker デーモン 19.03.13-ce 以降では、コンテナ用として 4 MiB 以上のメモリが予約されます。このため、このコンテナ用には 4 MiB 未満のメモリを指定しないようにします。

> **Note**
>
> リソースの使用率を最大化することを目的に、特定のインスタンスタイプにおいて、タスクにできるだけ多くのメモリを提供する場合には、「Amazon ECS Linux コンテナインスタンスのメモリを予約する」を参照してください。

ポートマッピング

`portMappings`

タイプ: オブジェクト配列

必須: いいえ

ポートマッピングは、コンテナのネットワークポートを外部に公開します。これは、クライアントがアプリケーションにアクセスすることを可能にします。これは、同じタスク内にあるコンテナ間の通信にも使用されます。

`awsvpc` ネットワークモードを使用するタスク定義では、`containerPort` のみを指定します。`hostPort` は常に無視され、コンテナポートはホスト上の番号が大きいランダムなポートに自動的にマップされます。

Windows のポートマッピングでは、NetNAT の代わりに localhost ゲートウェイを使用します。Windows のポートマッピングにはループバックが存在しないため、ホスト自体からコンテナのマッピングされたポートにアクセスはできません。

このパラメータのほとんどのフィールド (containerPort、hostPort、protocol を含む) は docker create-container コマンドの PortBindings にマッピングされ、--publish オプションは docker run にマッピングされます。タスク定義でネットワークモードを host に設定している場合、ホストポートは未定義とするか、ポートマッピングのコンテナポートと一致させる必要があります。

> **Note**
>
> タスクが RUNNING ステータスに達すると、手動および自動で割り当てられたホストとコンテナポートが次の場所で表示されます:
>
> - コンソール: 指定されたタスクのコンテナ詳細の [Network Bindings] セクション。
> - AWS CLI: networkBindings コマンド出力の describe-tasks セクション。
> - API: DescribeTasks レスポンス。
> - メタデータ: タスクメタデータのエンドポイント。

`appProtocol`

タイプ: 文字列

必須: いいえ

ポートマッピングに使用されるアプリケーションプロトコル。このパラメータは Service Connect にのみ適用されます。アプリケーションが使用するプロトコルと一貫性を持つように、このパラメータを設定することをお勧めします。このパラメータを設定すると、Amazon ECS はサーバー コネクト プロキシにプロトコル固有の接続処理を追加します。このパラメータを設定すると、Amazon ECS は Amazon ECS コンソールと CloudWatch にプロトコル固有のテレメトリを追加します。

このパラメータ値を設定しないと、TCP が使用されます。ただし、Amazon ECS では、TCP 用のプロトコル固有のテレメトリーは追加されません。

詳細については、「the section called “ Service Connect”」を参照してください。

有効なプロトコル値: `"http" | "http2" | "grpc"`

containerPort

タイプ: 整数

必須: はい (portMappings を使用する場合)

ユーザーが指定したホストポートまたは自動的に割り当てられたホストポートにバインドされるコンテナポートの番号。

awsvpc ネットワークモードを使用するタスクの場合は、containerPort を使用して公開ポートを指定します。

EC2 キャパシティプロバイダーのタスクでコンテナを使用しており、ホストポートではなくコンテナポートを指定するとします。その後、コンテナはエフェメラルポート範囲内のホストポートを自動で受け取ります。詳細については、「hostPort」を参照してください。この方法で自動的に割り当てられるれるポートマッピングは、コンテナインスタンスの予約済みポート数 100 個のクォータに対してはカウントされません。

containerPortRange

タイプ: 文字列

必須: いいえ

動的にマッピングされたホストポート範囲にバインドされるコンテナのポート番号の範囲。

このパラメータは、register-task-definition API を使用してのみ設定できます。このオプションは、portMappings パラメータで使用できます。詳細については、「AWS Command Line Interface リファレンス」の「register-task-definition」を参照してください。

containerPortRange を指定するときは、以下のルールが適用されます。

- bridge ネットワークモードまたは awsvpc ネットワークモードのいずれかを使用する必要があります。
- このパラメータは、Linux と Windows の両オペレーティングシステムで使用できます。
- コンテナインスタンスには、少なくともコンテナエージェントのバージョン 1.67.0 と ecs-init パッケージのバージョン 1.67.0-1 が必要です。
- 各コンテナにつき、最大 100 のポートレンジを指定できます。
- hostPortRange は指定しません。hostPortRange の値は次のように設定されます。
  - awsvpc ネットワークモードのタスク内のコンテナでは、hostPort は containerPort と同じ値に設定されます。これは静的マッピング戦略です。

- bridge ネットワークモードのタスク内のコンテナの場合、Amazon ECS エージェントはデフォルトの一時範囲から開いているホストポートを見つけ、それを Docker に渡してコンテナポートにバインドします。
- containerPortRange の有効な値は 1 ~ 65535 です。
- 1 つのポートは、コンテナごとに 1 つのポートマッピングにのみ含められます。
- 重複するポート範囲は指定できません。
- 範囲内の最初のポートは、最後のポートよりも小さくなければなりません。
- Docker では、ポートの数が多い場合は、Docker デーモン設定ファイルの Docker-proxy をオフにすることをお勧めします。

  詳細については、GitHub の Issue #11185 を参照してください。

  Docker デーモン設定ファイルの docker-proxy をオフにする方法については、Amazon ECS 開発者ガイドの「Docker デーモン」を参照してください。

DescribeTasks を呼び出すと、コンテナポートにバインドされているホストポートである hostPortRange を表示できます。

ポート範囲は、EventBridge に送信される Amazon ECS タスクイベントに含まれません。詳細については、「the section called “EventBridge を使用して Amazon ECS エラーへの対応を自動化する”」を参照してください。

hostPortRange

型: 文字列

必須: いいえ

ネットワークバインディングで使用されるホストのポート番号範囲。これは Docker によって割り当てられ、Amazon ECS エージェントによって配信されます。

hostPort

タイプ: 整数

必須: いいえ

コンテナ用に予約するコンテナインスタンスのポート番号。

コンテナポートマッピングに、予約されていないホストポートを指定できます。これは、静的ホストポートマッピングと呼ばれます。または、containerPort を指定するときに hostPort を省略 (または 0 に設定) できます。コンテナは、コンテナインスタンスのオペ

レーティングシステムと Docker バージョンに応じたエフェメラルポート範囲のポートを自動で受け取ります。これは、動的ホストポートマッピングと呼ばれます。

Docker バージョン 1.6.0 以降のデフォルトの一時ポート範囲は、インスタンスの `/proc/sys/net/ipv4/ip_local_port_range` にリストされています。このカーネルパラメータが使用できない場合、`49153-65535` から始まるデフォルトのエフェメラルポート範囲が使用されます。エフェメラルポート範囲では、ホストポートを指定しないでください。これらの範囲は自動割り当て用に予約済みです。一般的に、32768 より小さい番号のポートは一時ポート範囲に含まれません。ECS コンテナエージェント設定内の `ECS_DYNAMIC_HOST_PORT_RANGE` 設定を使用して、動的に割り当てられたホストポートのカスタム範囲を指定できます。これは、エフェメラルポート範囲からのポートも使用するアウトバウンド接続など、コンテナインスタンス上の他のプロセスとのポート競合が原因でタスクが開始されない場合に役立ちます。詳細については、「Amazon ECS コンテナエージェントの設定」を参照してください。

デフォルトの予約済みポートは、SSH 用の 22、Docker ポートの 2375 および 2376、Amazon ECS コンテナエージェントポートの `51678-51680` です。実行中のタスクに対して以前にユーザーが指定したホストポートも、タスクの実行中に予約されます。タスクが停止すると、ホストポートはリリースされます。現在の予約済みポートは、describe-container-instances 出力の `remainingResources` に表示されます。コンテナインスタンスには、デフォルトの予約済みポートを含めて、一度に最大 100 個の予約済みポートを割り当てられます。自動的に割り当てられるポートは、この 100 個の予約済みポートクォータにはカウントされません。

`name`

タイプ: 文字列

必須: いいえ。サービスに Service Connect および VPC Lattice を設定する場合は必要です。

ポートマッピングに使用される名前。このパラメータは Service Connect および VPC Lattice にのみ適用されます。このパラメータは、サービスの Service Connect および VPC Lattice 設定で使用する名前です。

詳細については、「Service Connect を使用して Amazon ECS サービスを短縮名で接続する」を参照してください。

次の例では、Service Connect および VPC Lattice の 2 つの必須フィールドが使用されています。

```
"portMappings": [
    {
        "name": string,
        "containerPort": integer
    }
]
```

protocol

タイプ: 文字列

必須: いいえ

ポートマッピングに使用されるプロトコル。有効な値は、tcp および udp です。デフォルト: tcp。

> **Important**
>
> tcp は Service Connect でのみサポートされます。このフィールドが設定されていない場合は、tcp が暗示されることに注意してください。

> **Important**
>
> UDP サポートは、バージョン 1.2.0 以降の Amazon ECS コンテナエージェント (amzn-ami-2015.03.c-amazon-ecs-optimized AMI など)、またはバージョン 1.3.0 以降に更新したコンテナエージェントで起動されたコンテナインスタンスでのみ使用できます。コンテナエージェントを最新バージョンに更新するには、「Amazon ECS コンテナエージェントをアップデートする」を参照してください。

ホストポートを指定する際は、以下の構文を使用します。

```
"portMappings": [
    {
        "containerPort": integer,
        "hostPort": integer
    }
    ...
]
```

自動割り当てのホストポートが必要な場合は、以下の構文を使用します。

```
"portMappings": [
    {
        "containerPort": integer
    }
    ...
]
```

プライベートリポジトリの認証情報

`repositoryCredentials`

タイプ: RepositoryCredentials オブジェクト

必須: いいえ

プライベートレジストリ認証用のリポジトリ認証情報。

詳細については、「Amazon ECS での AWS 以外のコンテナイメージの使用」を参照してください。

`credentialsParameter`

タイプ: 文字列

必須: はい (`repositoryCredentials` を使用する場合)

プライベートリポジトリの認証情報が含まれているシークレットの Amazon リソースネーム (ARN)。

詳細については、「Amazon ECS での AWS 以外のコンテナイメージの使用」を参照してください。

> **Note**
>
> Amazon ECS API、AWS CLI、または AWS SDK を使用する場合、起動するタスクと同じリージョンにシークレットが存在する場合は、シークレットの完全な ARN または名前のどちらも使用できます。AWS マネジメントコンソール を使用する場合は、シークレットの完全な ARN を指定する必要があります。

必要なパラメータが含まれるタスク定義のスニペットを、以下に示します。

```
"containerDefinitions": [
    {
        "image": "private-repo/private-image",
        "repositoryCredentials": {
            "credentialsParameter":
 "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name"
        }
    }
]
```

詳細コンテナ定義パラメータ

以下のコンテナ定義用の詳細パラメータは、Amazon ECS コンテナインスタンスでのコンテナの起動に使用する Docker run コマンドの拡張機能を追加します。

トピック

- 再起動ポリシー
- ヘルスチェック
- 環境
- ネットワーク設定
- ストレージとログ記録
- セキュリティ
- リソースの制限
- Docker のラベル

再起動ポリシー

`restartPolicy`

コンテナ再起動ポリシーと関連する設定パラメータ。コンテナの再起動ポリシーを設定すると、Amazon ECS はタスクを置き換えることなくコンテナを再起動できます。詳細については、「コンテナ再起動ポリシーを使用して Amazon ECS タスク内の個々のコンテナを再起動する」を参照してください。

`enabled`

タイプ: ブール値

必須: はい

コンテナに対して再起動ポリシーを有効にするかどうかを指定します。

`ignoredExitCodes`

タイプ: 整数配列

必須: いいえ

Amazon ECS が無視し、再起動を試みない終了コードのリスト。最大 50 個のコンテナ終了コードを指定できます。デフォルトでは、Amazon ECS はいずれの終了コードも無視しません。

`restartAttemptPeriod`

タイプ: 整数

必須: いいえ

再起動を試みる前にコンテナが実行する必要がある時間 (秒単位) 。コンテナを再起動できるのは、`restartAttemptPeriod` 秒に 1 回のみです。コンテナがこの期間実行できずに早く終了した場合、コンテナは再起動されません。最小の `restartAttemptPeriod` は 60 秒、最大の `restartAttemptPeriod` は 1,800 秒を指定できます。デフォルトでは、コンテナは再起動の前に 300 秒間実行する必要があります。

ヘルスチェック

`healthCheck`

コンテナに対するヘルスチェックのコマンドと、コンテナのための関連する設定パラメータです。詳細については、「コンテナのヘルスチェックを使用して Amazon ECS タスク状態を判定する」を参照してください。

`command`

正常状態かどうかを決定するために、コンテナが実行するコマンドが格納された文字列配列。この文字列配列の先頭には、コマンド引数を直接実行するための `CMD`、またはコンテナのデフォルトシェルでコマンドを実行するための `CMD-SHELL` を付加できます。これらのいずれも指定しない場合は `CMD` が使用されます。

AWS マネジメントコンソール にタスク定義を登録する場合は、カンマで区切ったコマンドリストを使用してください。これらのコマンドは、タスク定義が作成された後に文字列に変換されます。ヘルスチェックに対する入力の例を次に示します。

```
CMD-SHELL, curl -f http://localhost/ || exit 1
```

AWS マネジメントコンソール JSON パネル、AWS CLI、または API を使用してタスク定義を登録するときは、コマンドのリストを角かっこで囲みます。ヘルスチェックに対する入力の例を次に示します。

```
[ "CMD-SHELL", "curl -f http://localhost/ || exit 1" ]
```

stderr が出力されない終了コード 0 は成功を示し、0 以外の終了コードは失敗を示します。

`interval`

各ヘルスチェック間の時間間隔 (秒単位)。5 〜 300 秒を指定できます。デフォルトの値は 30 秒です。

`timeout`

成功まで待機しているヘルスチェックが、失敗したと見なされるまでの期間 (秒単位) です。2 〜 60 秒を指定できます。デフォルト値は 5 秒です。

`retries`

コンテナが異常と見なされるまでに、失敗したヘルスチェックを再試行する回数です。1〜10 回を指定できます。デフォルト値は 3 回の再試行です。

`startPeriod`

失敗したヘルスチェックの再試行が最大回数に達する前に、コンテナにブートストラップための時間を提供するオプションの猶予期間です。0 〜 300 秒を指定できます。デフォルトでは、`startPeriod` は無効となっています。

ヘルスチェックが `startPeriod` 内で成功した場合、コンテナは正常であるとみなされ、その後の失敗は最大再試行回数にカウントされます。

環境

`cpu`

タイプ: 整数

必須: いいえ

Amazon ECS コンテナエージェントがコンテナ用に予約した cpu ユニットの数。Linux では、このパラメータは「Create a container」セクションの CpuShares にマッピングされます。

> **Note**
>
> Amazon EC2 インスタンスタイプごとに使用可能な CPU ユニットの数を決定できます。これを行うには、Amazon EC2 インスタンスの詳細ページでそのインスタンスタイプにリストされている vCPU の数を 1,024 倍します。

Linux コンテナは、割り当てられた CPU ユニットと同じ比率を使用して、割り当てられていない CPU ユニットをコンテナインスタンス上の他コンテナと共有します。例えば、そのコンテナ用に 512 個の CPU ユニットが指定された単一コアインスタンスタイプで、唯一のコンテナタスクを実行するとします。さらに、そのタスクはコンテナインスタンスで実行される唯一のタスクです。この場合のコンテナは、1,024 個の CPU ユニット配分を任意のタイミングで使用できます。一方、そのコンテナインスタンスで同じタスクのコピーを別途起動したと仮定します。各タスクには、必要に応じて少なくとも 512 個の CPU ユニットが保証されます。同様に、その他のコンテナが残りの CPU を使用していない場合、各コンテナは CPU 使用率を高められます。ただし、両方のタスクが常に 100% アクティブである場合、使用できるのは 512 CPU ユニットに制限されます。

Linux コンテナインスタンスでは、コンテナインスタンス上の Docker デーモンは、CPU 値を使用して、実行中のコンテナに対する相対 CPU 配分比率を計算します。Linux カーネルが許可する最小の有効な CPU 共有値は 2 で、Linux カーネルが許可する最大の有効な CPU 共有値は 262,144 です。ただし、CPU のパラメータは必須ではなく、コンテナ定義では 2 未満の CPU 値および 262144 より大きな CPU 値を使用できます。CPU 値が 2 未満 (null を含む) および 262144 より大きな値である場合、動作は Amazon ECS コンテナエージェントのバージョンによって異なります。

- Agent versions <= 1.1.0: Null およびゼロの CPU 値は Docker に 0 として渡され、Docker は 1,024 個の CPU 配分に変換します。1 の CPU 値は、Docker に対しては 1 として渡され、Linux カーネルではそれが 2 個の CPU 配分に変換されます。
- エージェントバージョン >= 1.2.0: Null、0、および 1 の CPU 値は、Docker に 2 個の CPU 配分として渡されます。
- エージェントバージョン >= 1.84.0: 256 vCPU を超える CPU 値は Docker に 256 として渡されます。これは 262144 CPU 共有に相当します。

Windows コンテナインスタンスでは、CPU クォータは絶対クォータとして適用されます。Windows コンテナは、タスク定義で定義されている指定された量の CPU にのみアクセスできます。null またはゼロの CPU 値が `0` として Docker に渡されます。次に Windows はこの値を 1 つの CPU の 1% として解釈します。

その他の例については、「Amazon ECS で CPU およびメモリリソースの管理方法」を参照してください。

`gpu`

タイプ: ResourceRequirement オブジェクト

必須: いいえ

Amazon ECS コンテナエージェントがコンテナ用に予約した物理 GPUs の数。整数値、フラクショナル GPU の小数値、または `ALL` を指定できます。`ALL` を指定すると、コンテナインスタンス上のすべての GPU がコンテナに割り当てられます。10 進値 (`0.125`、`0.25`、`0.5` など) を指定すると、Amazon ECS はリクエストされた GPU 容量を提供する小数 GPU インスタンス (G6f) にタスクを配置します。タスク内でコンテナ用に予約されているすべての GPU の数は、タスクが起動されたコンテナインスタンスで使用できる GPU の数を超えることはできません。詳細については、「GPU ワークロード向けの Amazon ECS タスク定義」を参照してください。

> **Note**
> このパラメータは Windows コンテナではサポートされません。

`Elastic Inference accelerator`

タイプ: ResourceRequirement オブジェクト

必須: いいえ

`InferenceAccelerator` タイプでは、`value` はタスク定義で指定される `InferenceAccelerator` の `deviceName` と一致します。詳細については、「the section called “Elastic Inference アクセラレーター名 (廃止)”」を参照してください。

> **Note**
> このパラメータは Windows コンテナではサポートされません。

`essential`

タイプ: ブール値

必須: いいえ

コンテナの `essential` パラメーターが `true` としてマークされており、そのコンテナが何らかの理由で失敗するか停止するとします。その後、タスクの一部である他のすべてのコンテナが停止されます。コンテナの `essential` パラメータを `false` にマークしておくと、そのコンテナ失敗はタスク内にある残りのコンテナに影響を与えません。このパラメータを省略した場合、コンテナは必須と見なされます。

すべてのタスクには少なくとも 1 つの必須コンテナが必要です。複数のコンテナで構成されるアプリケーションがあるとします。次に、共通の目的で使用されるコンテナをコンポーネントにグループ化し、さまざまなコンポーネントを複数のタスク定義に分離します。詳細については、「Amazon ECS 用のアプリケーションの構築」を参照してください。

```
"essential": true|false
```

`entryPoint`

> **Important**
>
> 初期のバージョンの Amazon ECS コンテナエージェントは、`entryPoint` パラメータを正しく処理しません。`entryPoint` の使用時に問題が発生する場合は、コンテナエージェントを更新するか、代わりに `command` 配列項目としてコマンドと引数を入力します。

タイプ: 文字列配列

必須: いいえ

コンテナに渡されるエントリポイント。

```
"entryPoint": ["string", ...]
```

`command`

タイプ: 文字列配列

必須: いいえ

コンテナに渡されるコマンド。このパラメータは create-container コマンドの Cmd にマッピングされ、 COMMAND パラメータは docker run にマッピングされます。複数の引数がある場合、各引数は配列内で、区切られた文字列により指定する必要があります。

```
"command": ["string", ...]
```

workingDirectory

タイプ: 文字列

必須: いいえ

コマンドを実行するコンテナ内の作業ディレクトリ。このパラメータは、Docker Remote API の Create a container (コンテナを作成する) セクションの WorkingDir にマップされ、--workdir オプションは docker run にマップされます。

```
"workingDirectory": "string"
```

environmentFiles

タイプ: オブジェクト配列

必須: いいえ

コンテナに渡す環境変数が含まれるファイルのリスト。このパラメータは Docker run コマンドの --env-file オプションにマッピングされます。

FIPS が有効化されている場合、ピリオド (.) が使用されているバケット名 (amzn-s3-demo-bucket1.name.example など) はサポートされません。バケット名にピリオド (.) を使用すると、エージェントが Amazon S3 から環境変数ファイルを取得できなくなるため、タスクが開始されません。

これは Windows コンテナには使用できません。

最大 10 個の環境ファイルを指定できます。ファイルには、ファイル拡張子 .env が必要です。環境ファイルの各行には、VARIABLE=VALUE 形式で環境変数が含まれています。# で始まる行はコメントとして扱われ、無視されます。

コンテナ定義に個別の環境変数が指定されている場合は、環境ファイルに含まれる変数よりも優先されます。同じ変数を含む複数の環境ファイルが指定されている場合、それらのファイルは上から下に処理されます。一意の変数名を使用することをお勧めします。詳細については、「個々の環境変数を Amazon ECS コンテナに渡す」を参照してください。

`value`

型: 文字列

必須: はい

環境変数ファイルを含む Amazon S3 オブジェクトの Amazon リソースネーム (ARN)。

`type`

型: 文字列

必須: はい

使用するファイルのタイプ。`s3` はサポートされる唯一の値です。

`environment`

タイプ: オブジェクト配列

必須: いいえ

コンテナに渡す環境変数。このパラメータは docker create-container コマンドの `Env` にマッピングされ、`--env` オプションは docker run コマンドにマッピングされます。

> **Important**
> 認証情報データなどの機密情報にプレーンテキストの環境変数を使用することはお勧めしません。

`name`

型: 文字列

必須: はい (`environment` を使用する場合)

環境変数の名前。

`value`

型: 文字列

必須: はい (`environment` を使用する場合)

環境変数の値。

```
"environment" : [
    { "name" : "string", "value" : "string" },
    { "name" : "string", "value" : "string" }
]
```

`secrets`

タイプ: オブジェクト配列

必須: いいえ

コンテナに公開するシークレットを表すオブジェクトです。詳細については、「Amazon ECS コンテナに機密データを渡す」を参照してください。

`name`

型: 文字列

必須: はい

コンテナの環境変数として設定する値。

`valueFrom`

型: 文字列

必須: はい

コンテナに公開するシークレット。サポートされている値は、AWS Secrets Manager シークレットの完全な Amazon リソースネーム (ARN)、または AWS Systems Manager Parameter Store 内のパラメータの完全な ARN のいずれかです。

> **Note**
>
> 起動するタスクと同じ AWS リージョン に Systems Manager パラメータストアのパラメータまたは Secrets Manager のパラメータが存在する場合は、シークレットの完

全な ARN または名前のいずれかを使用できます。パラメータが別のリージョンに存在する場合は、完全な ARN を指定する必要があります。

```
"secrets": [
    {
        "name": "environment_variable_name",
        "valueFrom": "arn:aws:ssm:region:aws_account_id:parameter/parameter_name"
    }
]
```

ネットワーク設定

`disableNetworking`

タイプ: ブール値

必須: いいえ

このパラメータが true のとき、ネットワークはコンテナ内でオフになります。

> **Note**
>
> awsvpc ネットワークモードを使用するタスクもしくは Windows コンテナでは、このパラメータはサポートされません。

デフォルトは `false` です。

```
"disableNetworking": true|false
```

`links`

タイプ: 文字列配列

必須: いいえ

`link` パラメータでは、コンテナがポートマッピングを必要とせずに互いに通信することを許可します。このパラメータは、タスク定義のネットワークモードが `bridge` に設定されている場合にのみサポートされます。`name:internalName` コンストラクトは Docker リンク

の name:alias に似ています。最大 255 文字の英字 (大文字と小文字)、数字、ハイフン、アンダースコアを使用できます。

> **Note**
>
> awsvpc ネットワークモードを使用するタスクもしくは Windows コンテナでは、このパラメータはサポートされません。

> **Important**
>
> 同じコンテナインスタンスに配置されたコンテナ同士は、リンクやホストポートマッピングを必要とせずに、互いに通信する場合があります。コンテナインスタンスでのネットワークの分離は、セキュリティグループと VPC 設定によって制御されます。

```
"links": ["name:internalName", ...]
```

hostname

タイプ: 文字列

必須: いいえ

コンテナに使用するホスト名。このパラメータは docker create-container の Hostname にマッピングされ、--hostname オプションは docker run にマッピングされます。

> **Note**
>
> awsvpc ネットワークモードを使用している場合、hostname パラメータはサポートされません。

```
"hostname": "string"
```

dnsServers

タイプ: 文字列配列

必須: いいえ

コンテナに提示する DNS サーバーのリスト。

> **Note**
>
> awsvpc ネットワークモードを使用するタスクもしくは Windows コンテナでは、このパラメータはサポートされません。

```
"dnsServers": ["string", ...]
```

`dnsSearchDomains`

タイプ: 文字列配列

必須: いいえ

パターン: ^[a-zA-Z0-9-.]{0,253}[a-zA-Z0-9]$

コンテナに提示する DNS 検索ドメインのリスト。このパラメータは docker create-container コマンドの `DnsSearch` にマッピングされ、`--dns-search` オプションは docker run にマッピングされます。

> **Note**
>
> このパラメータは Windows コンテナまたは、`awsvpc` ネットワークモードを使用するタスクではサポートされていません。

```
"dnsSearchDomains": ["string", ...]
```

`extraHosts`

タイプ: オブジェクト配列

必須: いいえ

コンテナ上の `/etc/hosts` ファイルに追加する、ホスト名と IP アドレスのマッピングリスト。

このパラメータは docker create-container コマンドの `ExtraHosts` にマッピングされ、`--add-host` オプションは docker run にマッピングされます。

> **Note**
>
> このパラメータは Windows コンテナまたは、awsvpc ネットワークモードを使用するタスクではサポートされていません。

```
"extraHosts": [
      {
        "hostname": "string",
        "ipAddress": "string"
      }
      ...
    ]
```

`hostname`

タイプ: 文字列

必須: はい (`extraHosts` を使用する場合)

`/etc/hosts` エントリで使用するホスト名。

`ipAddress`

タイプ: 文字列

必須: はい (`extraHosts` を使用する場合)

`/etc/hosts` エントリで使用する IP アドレス。

ストレージとログ記録

`readonlyRootFilesystem`

タイプ: ブール値

必須: いいえ

このパラメータが true のとき、コンテナはそのルートファイルシステムへの読み取り専用アクセスを許可されます。このパラメータは docker create-container コマンドの `ReadonlyRootfs` にマッピングされ、`--read-only` オプションは docker run にマッピングされます。

> **Note**
>
> このパラメータは Windows コンテナではサポートされません。

デフォルトは `false` です。

```
"readonlyRootFilesystem": true|false
```

`mountPoints`

タイプ: オブジェクト配列

必須: いいえ

コンテナでのデータボリュームのマウントポイント。このパラメータは creat-container Docker API の `Volumes` にマッピングされ、docker run の `--volume` オプションにマッピングされます。

Windows コンテナは `$env:ProgramData` と同じドライブに全部のディレクトリをマウントできます。Windows コンテナは、別のドライブにディレクトリをマウントすることはできません。また、マウントポイントは複数のドライブにまたがることはできません。Amazon EBS ボリュームを Amazon ECS タスクに直接アタッチするには、マウントポイントを指定する必要があります。

`sourceVolume`

タイプ: 文字列

必須: はい (`mountPoints` を使用する場合)

マウントするボリュームの名前。

`containerPath`

型: 文字列

必須: はい (`mountPoints` を使用する場合)

ボリュームをマウントするコンテナ内のパス。

`readOnly`

タイプ: ブール値

必須: いいえ

この値が true の場合、コンテナはボリュームへの読み取り専用アクセスを許可されます。この値が false の場合、コンテナはボリュームに書き込むことができます。デフォルト値は false です。

Windows オペレーティングシステムを実行しているタスクの場合、 値をデフォルト の false のままにします。

volumesFrom

タイプ: オブジェクト配列

必須: いいえ

別コンテナからマウントするデータボリューム。このパラメータは docker create-container コマンドの VolumesFrom にマッピングされ、--volumes-from オプションは docker run にマッピングされます。

sourceContainer

タイプ: 文字列

必須: はい (volumesFrom を使用する場合)

ボリュームのマウント元のコンテナの名前。

readOnly

タイプ: ブール値

必須: いいえ

この値が true の場合、コンテナはボリュームへの読み取り専用アクセスを許可されます。この値が false の場合、コンテナはボリュームに書き込むことができます。デフォルト値は false です。

```
"volumesFrom": [
                {
                  "sourceContainer": "string",
                  "readOnly": true|false
                }
              ]
```

`logConfiguration`

タイプ: LogConfiguration オブジェクト

必須: いいえ

コンテナに対するログ構成の仕様です。

ログ設定を使用したタスク定義の例については、「Amazon ECS のタスク定義の例」を参照してください。

このパラメータは docker create-container コマンドの `LogConfig` にマッピングされ、`--log-driver` オプションは docker run にマッピングされます。デフォルトでは、コンテナは Docker デーモンで使用されるのと同じロギングドライバーを使用します。ただし、コンテナ定義の中で、このパラメータによりログドライバーを指定することで、Docker デーモンとは異なるログドライバーをコンテナに使用させることも可能です。コンテナに異なるロギングドライバーを使用するには、コンテナインスタンス (またはリモートログ記録オプションの別のログサーバー) でログシステムを適切に設定する必要があります。

コンテナのログ設定を指定する際には、以下の点に注意してください。

- 現在、Amazon ECS では Docker デーモンに使用可能なログドライバーがいくつかサポートされています。
- このパラメータを使用するには、コンテナインスタンスで Docker Remote API のバージョン 1.18 以降が必要です。
- コンテナインスタンスで実行される Amazon ECS コンテナエージェントは、そのインスタンスに配置されたコンテナがこれらのログ設定オプションを使用できるようにする前に、そのインスタンスで使用可能なログドライバーを `ECS_AVAILABLE_LOGGING_DRIVERS` 環境変数に登録する必要があります。詳細については、「Amazon ECS コンテナエージェントの設定」を参照してください。

```
"logConfiguration": {
      "logDriver": "awslogs","fluentd","gelf","json-
file","journald","splunk","syslog","awsfirelens",
      "options": {"string": "string"
        ...},
 "secretOptions": [{
  "name": "string",
  "valueFrom": "string"
 }]
```

```
}
```

`logDriver`

型: 文字列

有効な値: `"awslogs","fluentd","gelf","json-file","journald","splunk","syslog","awsfirelens"`

必須: はい (`logConfiguration` を使用する場合)

コンテナに使用するログドライバー。デフォルトでは、上記の有効な値は Amazon ECS コンテナエージェントが通信できるログドライバーです。

サポートされているログドライバーは `awslogs`、`fluentd`、`gelf`、`json-file`、`journald`、`syslog`、`splunk`、`awsfirelens` です。

タスク定義で `awslogs` ログドライバーを使用してコンテナログを CloudWatch Logs に送信する方法については、「Amazon ECS ログを CloudWatch に送信する」を参照してください。

`awsfirelens` ログドライバーの使用の詳細については、「カスタムログのルーティング」を参照してください。

> **Note**
>
> 上記に示されていないカスタムドライバーがある場合、GitHub で入手できる Amazon ECS コンテナエージェントプロジェクトを fork し、そのドライバーを使用するようにカスタマイズできます。含めたい変更については、プルリクエストを送信することをお勧めします。ただし、現時点では、このソフトウェアの変更されたコピーの実行はサポートされていません。

このパラメータは、コンテナインスタンスで Docker Remote API バージョン 1.18 以上を使用する必要があります。

`options`

タイプ: 文字列間のマッピング

必須: いいえ

ログドライバーに送信する設定オプションの Key-Value マップ。

指定できるオプションは、ログドライバーに応じて異なります。awslogs ルーターを使用して Amazon CloudWatch にログをルーティングするときに指定できるオプションには、以下が含まれます。

`awslogs-create-group`

必須: いいえ

ロググループを自動的に作成させるかどうかを指定します。このオプションを指定しない場合、デフォルトは `false` です。

> **Note**
>
> `awslogs-create-group` を使用する前に、IAM ポリシーにはアクセス許可 `logs:CreateLogGroup` が含まれている必要があります。

`awslogs-region`

必須: はい

awslogs ログドライバが Docker ログを送信する先の、AWS リージョン を指定します。CloudWatch Logs では、異なるリージョンのクラスターからすべてのログを 1 つのリージョンに送信するように選択できます。これにより、すべてのログを一元的な場所で確認できるようになります。他にも、リージョンごとにそれらを分離して、より細分化することが可能です。このオプションで指定するリージョンに、対象のロググループが存在することを確認してください。

`awslogs-group`

必須: はい

awslogs ログドライバーがログストリームを送信する先の、ロググループを指定する必要があります。

`awslogs-stream-prefix`

必須：オプション

指定したプレフィックス、コンテナ名、コンテナが属する Amazon ECS タスクの ID にログストリームを関連付けるには、`awslogs-stream-prefix` オプションを使用します。

このオプションでプレフィックスを指定した場合、ログストリームの形式は以下のようになります。

```
prefix-name/container-name/ecs-task-id
```

このオプションでプレフィックスを指定しない場合、ログストリームには、コンテナインスタンスの Docker デーモンによって割り当てられたコンテナ ID に基づいた名前が付けられます。Docker コンテナ ID (コンテナインスタンスでのみ使用可能) だけでそのログを送信したコンテナを追跡するのは難しいため、このオプションでプレフィックスを指定することをお勧めします。

Amazon ECS サービスでは、サービス名をプレフィックスとして使用できます。これにより、コンテナが属するサービスへのログストリームと、それを送信したコンテナの名前、そのコンテナが所属するタスクの ID を追跡できます。

Amazon ECS コンソールを使用する際に [Log] ペインにログを表示するためには、ログのストリームプレフィックスを指定する必要があります。

`awslogs-datetime-format`

必須: いいえ

このオプションは、Python `strftime` 形式で複数行起動パターンを定義します。ログメッセージは、パターンに一致する 1 行と、それに続くパターンに一致しない行で構成されます。一致する 1 行とは、ログメッセージ間の区切りです。

この形式を使用する場合のユースケースの例としては、スタックダンプなどの解析された出力があり、これを使用しなければ、複数のエントリに記録されることになります。適切なパターンにより、単一のエントリにキャプチャさせます。

詳細については、awslogs-datetime-formatを参照してください。

`awslogs-datetime-format` と `awslogs-multiline-pattern` オプションの両方を設定することはできません。

> **Note**
>
> 複数行のログ記録は、すべてのログメッセージの正規表現の解析とマッチングを実行します。これによりログ記録のパフォーマンスに悪影響が及ぶ可能性があります。

`awslogs-multiline-pattern`

必須: いいえ

このオプションでは、正規表現を使用する複数行起動パターンを定義します。ログメッセージは、パターンに一致する 1 行と、それに続くパターンに一致しない行で構成されます。一致する 1 行とは、ログメッセージ間の区切りです。

詳細については、「awslogs-multiline-pattern」を参照してください。

`awslogs-datetime-format` も設定されている場合は、このオプションは無視されます。

`awslogs-datetime-format` と `awslogs-multiline-pattern` オプションの両方を設定することはできません。

> **Note**
>
> 複数行のログ記録は、すべてのログメッセージの正規表現の解析とマッチングを実行します。これによりログ記録のパフォーマンスに悪影響が及ぶ可能性があります。

以下のオプションは、サポートされているすべてのログドライバーに適用されます。

`mode`

必須: いいえ

有効な値: `non-blocking` | `blocking`

このオプションは、コンテナから `logDriver` で指定したログドライバーへのログメッセージの配信モードを定義します。選択した配信モードは、コンテナからンテナーからのログの流れが中断されたときのアプリケーションの可用性に影響します。

`blocking` モードを使用していて、ログのフローが中断された場合、`stdout` ストリームと `stderr` ストリームに書き込むためのコンテナコードからの呼び出しがブロックされます。その結果、アプリケーションのロギングスレッドがブロックされます。これにより、アプリケーションが応答しなくなり、コンテナのヘルスチェックが失敗する可能性があります。

non-blocking モードを使用する場合、コンテナのログは代わりに max-buffer-size オプションで設定されたメモリ内の中間バッファに保存されます。これにより、ログを送信できない場合にアプリケーションが応答しなくなるのを防ぐことができます。サービスの可用性を確保したいが、多少のログ損失があっても問題ない場合は、このモードを使用することをおすすめします。詳細については、「Preventing log loss with non-blocking mode in the awslogs container log driver」を参照してください。

defaultLogDriverMode アカウント設定を使用して、特定の AWS リージョンにあるすべてのコンテナのデフォルトの mode を設定できます。logConfiguration で mode オプションを指定しない場合、またはアカウント設定を構成しない場合、Amazon ECS はデフォルトで non-blocking モードになります。アカウント設定の詳細については、「デフォルトのログドライバーモード」を参照してください。

non-blocking モードが使用されている場合、max-buffer-size ログオプションは、中間メッセージ用のストレージに使用されるバッファのサイズを制御します。アプリケーションに基づいて、必ず適切なバッファサイズを指定してください。タスクレベルで割り当てられるメモリの合計量は、ログドライバーのメモリバッファに加えて、すべてのコンテナに割り当てられたメモリ量よりも大きくなければなりません。

> **Note**
>
> 2025 年 6 月 25 日、Amazon ECS はデフォルトのログドライバーモードが blocking から non-blocking に変更され、ログ記録よりもタスクの可用性が優先されるようになりました。この変更以降も blocking モードを引き続き使用するには、次のいずれかを実行してください。
>
> - コンテナ定義の logConfiguration の mode オプションを blocking に設定します。
> - defaultLogDriverMode アカウント設定を blocking に設定します。

`max-buffer-size`

必須: いいえ

デフォルト値: `10 m`

non-blocking モードが使用されている場合、max-buffer-size ログオプションは、中間メッセージ用のストレージに使用されるバッファのサイズを制御します。アプリケー

ションに基づいて、必ず適切なバッファサイズを指定してください。バッファがいっぱいになると、それ以上ログを保存できなくなります。保存できないログは失われます。

splunk ログルーターを使用してログをルーティングするには、splunk-token と splunk-url を指定する必要があります。

ログの保存と分析のための AWS のサービスまたは AWS Partner Networkを送信先としたログのルーティングに awsfirelens ログルーターを使用するときは、log-driver-buffer-limit オプションを設定して、イベントがログルーターコンテナに送信される前に、メモリでバッファリングされるログのライン数を制限できます。この制限は、スループットが高い場合に発生する可能性がある、Docker 内のバッファのメモリ不足による、潜在的なログ損失の問題を解決するのに役立ちます。詳細については、「the section called “高スループットのログの設定”」を参照してください。

awsfirelens を使用してログをルーティングするときに指定できるその他のオプションは、送信先に応じて異なります。Amazon Data Firehose にログをエクスポートするときは、region を使用して AWS リージョンを指定し、delivery_stream を使用してログストリームの名前を指定することができます。

Amazon Kinesis Data Streams にログをエクスポートするときは、region を使用して AWS リージョンを指定し、stream を使用してデータストリーム名を指定することができます。

Amazon OpenSearch Service にログをエクスポートするときは、Name、Host (プロトコルを使用しない OpenSearch Service エンドポイント)、Port、Index、Type、Aws_auth、Aws_region、Suppress_Type_Name、および tls といったオプションを指定できます。

Amazon S3 にログをエクスポートするときは、bucket オプションを使用してバケットを指定できます。また、region、total_file_size、upload_timeout、および use_put_object をオプションとして指定することもできます。

このパラメータは、コンテナインスタンスで Docker Remote API バージョン 1.19 以上を使用する必要があります。

secretOptions

タイプ: オブジェクト配列

必須: いいえ

ログ設定に渡すシークレットを示すオブジェクト。ログ構成で使用されるシークレットには、認証トークン、証明書、または暗号化キーを含められます。詳細については、「Amazon ECS コンテナに機密データを渡す」を参照してください。

`name`

型: 文字列

必須: はい

コンテナの環境変数として設定する値。

`valueFrom`

型: 文字列

必須: はい

コンテナのログ設定に公開するシークレット。

```
"logConfiguration": {
 "logDriver": "splunk",
 "options": {
  "splunk-url": "https://cloud.splunk.com:8080",
  "splunk-token": "...",
  "tag": "...",
  ...
 },
 "secretOptions": [{
  "name": "splunk-token",
  "valueFrom": "/ecs/logconfig/splunkcred"
 }]
}
```

`firelensConfiguration`

タイプ: FirelensConfiguration オブジェクト

必須: いいえ

コンテナの FireLens 構成。これは、コンテナログのログルーターの指定と設定に使用されます。詳細については、「Amazon ECS ログを AWS サービスまたは AWS Partner に送信する」を参照してください。

```
{
    "firelensConfiguration": {
        "type": "fluentd",
        "options": {
            "KeyName": ""
        }
    }
}
```

options

型: 文字列間のマッピング

必須: いいえ

ログルーターの設定時に使用するオプションの Key-Value マップ。このフィールドはオプションで、カスタム設定ファイルを指定するか、タスク、タスク定義、クラスター、コンテナインスタンスの詳細などのメタデータを口グイベントに追加するために使用できます。指定した場合、使用する構文は "options":{"enable-ecs-log-metadata":"true|false","config-file-type:"s3|file","config-file-value":"arn:aws:s3:::*amzn-s3-demo-bucket*/fluent.conf|filepath"} です。詳細については、「Amazon ECS タスク定義の例: FireLens に口グをルーティングする」を参照してください。

type

型: 文字列

必須: はい

使用する口グルーター。有効な値は fluentd または fluentbit です。

セキュリティ

コンテナセキュリティの詳細については、「Amazon ECS タスクおよびコンテナのセキュリティのベストプラクティス」を参照してください。

credentialSpecs

タイプ: 文字列配列

必須: いいえ

Active Directory 認証用のコンテナを設定する認証情報仕様 (CredSpec) ファイルへの SSM または Amazon S3 の ARN のリスト。dockerSecurityOptions の代わりに、このパラメータを使用することをお勧めします。ARN の最大数は 1 です。

各 ARN には 2 つの形式があります。

credentialspecdomainless:MyARN

Secrets Manager にシークレットの追加セクションを持つ CredSpec を提供するために credentialspecdomainless:MyARN を使用します。シークレットでドメインへのログイン認証情報を指定します。

任意のコンテナインスタンスで実行される各タスクは、異なるドメインに参加できます。

この形式は、コンテナインスタンスをドメインに結合しなくても使用できます。

credentialspec:MyARN

単一ドメインに対して CredSpec を提供するために credentialspec:MyARN を使用します。

このタスク定義を使用するタスクを開始する前に、コンテナインスタンスをドメインに参加させる必要があります。

どちらの形式でも、SSM または Amazon S3 で MyARN を ARN に置き換えます。

credspec は、ユーザー名、パスワード、接続先のドメインを含むシークレットの ARN を Secrets Manager に提供する必要があります。セキュリティ向上のため、インスタンスはドメインレス認証のドメインに参加しません。インスタンス上の他のアプリケーションは、ドメインレス認証情報を使用できません。このパラメータを使用すると、タスクが別のドメインに参加する必要がある場合でも、同じインスタンスでタスクを実行できます。詳細については、「Windows コンテナでの gMSAs の使用」および「Linux コンテナ向け gMSAs を使用する」を参照してください。

privileged

タイプ: ブール値

必須: いいえ

このパラメータが true のとき、コンテナには、ホストコンテナインスタンスに対する昇格されたアクセス権限 (root ユーザーと同様) が付与されます。privileged でコンテナを実行することはお勧めしません。ほとんどの場合、privileged を使用する代わりに特定のパラメータを使用することで、必要な権限を正確に指定できます。

このパラメータは docker create-container コマンドの `Privileged` にマッピングされ、`--privileged` オプションは docker run にマッピングされます。

> **Note**
>
> このパラメータは、Windows コンテナ、または Fargate 起動タイプを使用するタスクではサポートされていません。

デフォルトは `false` です。

```
"privileged": true|false
```

`user`

タイプ: 文字列

必須: いいえ

コンテナ内で使用するユーザー。このパラメータは docker create-container コマンドの `User` にマッピングされ、`--user` オプションは docker run にマッピングされます。

> **Important**
>
> `host` ネットワークモードを使用してタスクを実行する場合は、ルートユーザー (UID 0) を使用してコンテナを実行しないでください。セキュリティのベストプラクティスとしては、常にルート以外のユーザーを使用します。

以下の形式を使用して、`user` を指定できます。UID または GID を指定する場合は、正の整数として指定する必要があります。

- `user`
- `user:group`
- `uid`
- `uid:gid`
- `user:gid`
- `uid:group`

> **Note**
>
> このパラメータは Windows コンテナではサポートされません。

```
"user": "string"
```

`dockerSecurityOptions`

タイプ: 文字列配列

有効な値: "no-new-privileges" | "apparmor:PROFILE" | "label:*value*" | "credentialspec:*CredentialSpecFilePath*"

必須: いいえ

複数のセキュリティシステムのカスタム設定を指定する文字列のリスト。

Linux タスクの場合、このパラメータを使用して、SELinux および AppArmor マルチレベルセキュリティシステムのカスタムラベルを参照できます。

このパラメータを使用して、Active Directory 認証用のコンテナを設定する認証情報仕様ファイルを参照できます。詳細については、「Amazon ECS の EC2 Windows コンテナで gMSA を使用する方法について説明します。」および「Amazon EC2 の Linux コンテナで gMSA を使用する」を参照してください。

このパラメータは docker create-container コマンドの `SecurityOpt` にマッピングされ、`--security-opt` オプションは docker run にマッピングされます。

```
"dockerSecurityOptions": ["string", ...]
```

> **Note**
>
> コンテナインスタンスで実行される Amazon ECS コンテナエージェントは、`ECS_SELINUX_CAPABLE=true` または `ECS_APPARMOR_CAPABLE=true` 環境変数を登録する必要があります。これにより、そのインスタンスに配置されたコンテナが、これらのセキュリティオプションを使用できるようになります。詳細については、「Amazon ECS コンテナエージェントの設定」を参照してください。

リソースの制限

`ulimits`

タイプ: オブジェクト配列

必須: いいえ

コンテナに定義する `ulimit` 値の一覧。この値は、オペレーティングシステムのデフォルトのリソースクォータ設定を上書きします。このパラメータは docker create-container コマンドの `Ulimits` にマッピングされ、`--ulimit` オプションは docker run にマッピングされます。

このパラメータは、コンテナインスタンスで Docker Remote API バージョン 1.18 以上を使用する必要があります。

> **Note**
>
> このパラメータは Windows コンテナではサポートされません。

```
"ulimits": [
      {
        "name":
 "core"|"cpu"|"data"|"fsize"|"locks"|"memlock"|"msgqueue"|"nice"|"nofile"|"nproc"|"rss"|"rtp
        "softLimit": integer,
        "hardLimit": integer
      }
      ...
    ]
```

`name`

型: 文字列

有効な値: `"core" | "cpu" | "data" | "fsize" | "locks" | "memlock" | "msgqueue" | "nice" | "nofile" | "nproc" | "rss" | "rtprio" | "rttime" | "sigpending" | "stack"`

必須: はい (`ulimits` を使用する場合)

`ulimit` の `type`。

`hardLimit`

タイプ: 整数

必須: はい (`ulimits` を使用する場合)

`ulimit` タイプのハード制限。値は、`ulimit` の `type` に応じて、バイト、秒、またはカウントとして指定できます。

`softLimit`

タイプ: 整数

必須: はい (`ulimits` を使用する場合)

`ulimit` タイプのソフト制限。値は、`ulimit` の `type` に応じて、バイト、秒、またはカウントとして指定できます。

Docker のラベル

`dockerLabels`

タイプ: 文字列間のマッピング

必須: いいえ

コンテナに追加するラベルのキー/値マップ。このパラメータは docker create-container コマンドの `Labels` にマッピングされ、`--label` オプションは docker run にマッピングされます。

このパラメータは、コンテナインスタンスで Docker Remote API バージョン 1.18 以上を使用する必要があります。

```
"dockerLabels": {"string": "string"
      ...}
```

その他のコンテナ定義のパラメータ

以下のコンテナ定義パラメータは、[Configure via JSON] (JSON による設定) オプションを使用して、Amazon ECS コンソールでタスク定義を登録する際に使用できます。詳細については、「コンソールを使用した Amazon ECS タスク定義の作成」を参照してください。

トピック

- Linux パラメータ
- コンテナの依存関係
- コンテナのタイムアウト
- システムコントロール
- インタラクティブ
- 擬似ターミナル

Linux パラメータ

`linuxParameters`

型: LinuxParameters オブジェクト

必須: いいえ

KernelCapabilities など、コンテナに適用される Linux 固有のオプション。

> **Note**
>
> このパラメータは Windows コンテナではサポートされません。

```
"linuxParameters": {
      "capabilities": {
        "add": ["string", ...],
        "drop": ["string", ...]
        }
      }
```

`capabilities`

型: KernelCapabilities オブジェクト

必須: いいえ

Docker によって提供されているデフォルト設定に対して追加または削除する、コンテナ用の Linux 機能。これらの Linux 機能の詳細については、Linux マニュアルページの「機能 (7)」を参照してください。

add

タイプ: 文字列配列

有効な値: "ALL" | "AUDIT_CONTROL" | "AUDIT_READ" | "AUDIT_WRITE" | "BLOCK_SUSPEND" | "CHOWN" | "DAC_OVERRIDE" | "DAC_READ_SEARCH" | "FOWNER" | "FSETID" | "IPC_LOCK" | "IPC_OWNER" | "KILL" | "LEASE" | "LINUX_IMMUTABLE" | "MAC_ADMIN" | "MAC_OVERRIDE" | "MKNOD" | "NET_ADMIN" | "NET_BIND_SERVICE" | "NET_BROADCAST" | "NET_RAW" | "SETFCAP" | "SETGID" | "SETPCAP" | "SETUID" | "SYS_ADMIN" | "SYS_BOOT" | "SYS_CHROOT" | "SYS_MODULE" | "SYS_NICE" | "SYS_PACCT" | "SYS_PTRACE" | "SYS_RAWIO" | "SYS_RESOURCE" | "SYS_TIME" | "SYS_TTY_CONFIG" | "SYSLOG" | "WAKE_ALARM"

必須：いいえ

Docker によって提供されているデフォルト設定に追加する、コンテナ用の Linux 機能。このパラメータは docker create-container コマンドの CapAdd にマッピングされ、--cap-add オプションは docker run にマッピングされます。

add

タイプ: 文字列配列

有効な値: "SYS_PTRACE"

必須：いいえ

Docker によって提供されているデフォルト設定のコンテナに追加する Linux 機能。このパラメータは docker create-container コマンドの CapAdd にマッピングされ、--cap-add オプションは docker run にマッピングされます。

drop

タイプ: 文字列配列

有効な値: "ALL" | "AUDIT_CONTROL" | "AUDIT_WRITE" | "BLOCK_SUSPEND" | "CHOWN" | "DAC_OVERRIDE" | "DAC_READ_SEARCH" | "FOWNER" | "FSETID" | "IPC_LOCK" | "IPC_OWNER" | "KILL" | "LEASE" | "LINUX_IMMUTABLE" | "MAC_ADMIN" | "MAC_OVERRIDE" | "MKNOD" | "NET_ADMIN" | "NET_BIND_SERVICE" | "NET_BROADCAST" | "NET_RAW"

| "SETFCAP" | "SETGID" | "SETPCAP" | "SETUID" | "SYS_ADMIN" | "SYS_BOOT" | "SYS_CHROOT" | "SYS_MODULE" | "SYS_NICE" | "SYS_PACCT" | "SYS_PTRACE" | "SYS_RAWIO" | "SYS_RESOURCE" | "SYS_TIME" | "SYS_TTY_CONFIG" | "SYSLOG" | "WAKE_ALARM"

必須：いいえ

Docker によって提供されているデフォルト設定のコンテナから削除する Linux 機能。このパラメータは docker create-container コマンドの `CapDrop` にマッピングされ、`--cap-drop` オプションは docker run にマッピングされます。

`devices`

コンテナに公開するすべてのホストデバイス。このパラメータは docker create-container コマンドの `Devices` にマッピングされ、`--device` オプションは Docker run にマッピングされます。

タイプ: デバイスオブジェクト配列

必須: いいえ

`hostPath`

ホストコンテナインスタンス上のデバイスのパス。

型: 文字列

必須: はい

`containerPath`

ホストデバイスを公開する先のコンテナ内のパス。

タイプ: 文字列

必須: いいえ

`permissions`

デバイス用のコンテナに提供する明示的な許可。デフォルトでは、コンテナにはデバイスの `read`、`write`、および `mknod` のアクセス許可があります。

タイプ: 文字列の配列

有効な値: `read` | `write` | `mknod`

`initProcessEnabled`

信号を転送しプロセスを利用するコンテナ内で、`init` を実行。このパラメータは docker run の `--init` オプションにマッピングされます。

このパラメータは、コンテナインスタンスで Docker Remote API バージョン 1.25 以上を使用する必要があります。

`maxSwap`

コンテナが使用できるスワップメモリの合計 (MiB 単位)。このパラメータは、docker run の `--memory-swap` オプションに変換されます。値はコンテナメモリの合計に `maxSwap` の値を加えた値です。

`maxSwap` の `0` 値を指定した場合、コンテナはスワップを使用しません。許容値は、`0` または任意の正の整数です。`maxSwap` パラメータを省略すると、コンテナは実行中のコンテナインスタンスのスワップ設定を使用します。`maxSwap` パラメータを使用するには、`swappiness` 値を設定する必要があります。

`sharedMemorySize`

`/dev/shm` ボリュームのサイズ値 (MiB) です。このパラメータは docker run の `--shm-size` オプションにマッピングされます。

タイプ: 整数

`swappiness`

このパラメータにより、コンテナのメモリスワップ動作を調整できます。`swappiness` の値が `0` であると、必要な場合を除きスワップは発生しません。`swappiness` の値が `100` であると、ページのスワップが頻繁に行われます。使用できる値は、`0` と `100` の間の整数です。値を指定しない場合、デフォルト値の `60` が適用されます。また、`maxSwap` の値が指定されていない場合、このパラメータは無視されます。このパラメータは docker run の `--memory-swappiness` オプションにマッピングされます。

> **Note**
>
> Amazon Linux 2023 でタスクを使用している場合、`swappiness` パラメータはサポートされていません。

`tmpfs`

tmpfs マウントのコンテナパス、マウントオプション、および最大サイズ (MiB) です。このパラメータは docker run の `--tmpfs` オプションにマッピングされます。

型: Tmpfs オブジェクト配列

必須: いいえ

`containerPath`

tmpfs ボリュームがマウントされる絶対ファイルパスです。

タイプ: 文字列

必須: はい

`mountOptions`

tmpfs ボリュームのマウントオプションのリストです。

型: 文字列の配列

必須: いいえ

有効な値: `"defaults" | "ro" | "rw" | "suid" | "nosuid" | "dev" | "nodev" | "exec" | "noexec" | "sync" | "async" | "dirsync" | "remount" | "mand" | "nomand" | "atime" | "noatime" | "diratime" | "nodiratime" | "bind" | "rbind" | "unbindable" | "runbindable" | "private" | "rprivate" | "shared" | "rshared" | "slave" | "rslave" | "relatime" | "norelatime" | "strictatime" | "nostrictatime" | "mode" | "uid" | "gid" | "nr_inodes" | "nr_blocks" | "mpol"`

`size`

tmpfs ボリュームの最大サイズ (MiB) です。

タイプ: 整数

必須: はい

コンテナの依存関係

dependsOn

型: ContainerDependency オブジェクトの配列

必須: いいえ

コンテナの起動と停止に定義されている依存関係。コンテナには複数の依存関係を含めることができます。依存関係がコンテナの起動に対して定義されている場合、コンテナの停止の場合、依存関係は逆になります。例については、コンテナの依存関係を参照してください。

> **Note**
>
> あるコンテナが依存関係における制限事項を満たさない場合、または制限を満たす前にタイムアウトした場合、Amazon ECS は、依存関係にある他のコンテナの状態も次に遷移させることはしません。

コンテナの依存関係を有効にするには、インスタンスにバージョン 1.26.0 以上のコンテナエージェントが必要です。ただし、最新のコンテナエージェントのバージョンを使用することをお勧めします。エージェントのバージョンの確認と最新バージョンへの更新については、「Amazon ECS コンテナエージェントをアップデートする」を参照してください。Amazon ECS に最適化された Amazon Linux AMI を使用している場合、インスタンスでは、ecs-init パッケージの 1.26.0-1 バージョン以降が必要です。バージョン 20190301 以降から起動されるコンテナインスタンスには、必要なバージョンのコンテナエージェントや ecs-init が含まれています。詳細については、「Amazon ECS に最適化された Linux AMI」を参照してください。

```
"dependsOn": [
    {
        "containerName": "string",
        "condition": "string"
    }
]
```

containerName

型: 文字列

必須: はい

コンテナ名が指定された条件を満たしている必要があります。

condition

型: 文字列

必須: はい

コンテナの依存関係の条件です。使用可能な条件とその動作を以下に示します。

- START - この条件は、すぐに現在のリンクとボリュームの動作を工ミュレートします。この条件は、他のコンテナの開始を許可する前に、依存コンテナが開始されていることを検証します。
- COMPLETE - この条件は、他のコンテナの開始を許可する前に、依存コンテナの実行が完了 (終了) することを検証します。これは、スクリプトを実行して終了するだけの、重要性の低いコンテナのために便利です。この条件は、必須コンテナには設定できません。
- SUCCESS - この条件は COMPLETE と同じですが、コンテナが zero ステータスで終了していることも必要です。この条件は、必須コンテナには設定できません。
- HEALTHY — この条件は、他のコンテナの開始を許可する前に、依存コンテナがそのコンテナのヘルスチェックに合格したことを検証します。これには、タスク定義に設定されているヘルスチェックが依存コンテナにある必要があります。タスクの起動時にのみ、この条件が確認されます。

コンテナのタイムアウト

startTimeout

タイプ: 整数

必須: いいえ

値の例: 120

コンテナの依存関係解決の再試行を止めるまでの待機時間 (秒)。

例えば、タスク定義内に 2 つのコンテナを指定するとします。containerA は、COMPLETE、SUCCESS、または HEALTHY のいずれかのステータスに到達する containerB に依存関係を持ちます。startTimeout の値に containerB が指定されていて、コンテナが時間内に目標のステータスまで到達しない場合、containerA は開始しません。

> **Note**
>
> あるコンテナが依存関係における制限事項を満たさない場合、または制限を満たす前にタイムアウトした場合、Amazon ECS は、依存関係にある他のコンテナの状態も次に遷移させることはしません。

最大値は 120 秒です。

`stopTimeout`

タイプ: 整数

必須: いいえ

値の例: `120`

コンテナが正常に終了しなかった場合にコンテナが強制終了されるまでの待機時間 (秒)。

`stopTimeout` パラメータが指定されていない場合、Amazon ECS コンテナエージェント構成変数 `ECS_CONTAINER_STOP_TIMEOUT` に設定された値がデフォルトで使用されます。`stopTimeout` パラメータまたは `ECS_CONTAINER_STOP_TIMEOUT` エージェント設定変数のいずれも設定されていない場合、Linux コンテナと Windows コンテナに対して、デフォルト値の 30 秒が適用されます。コンテナインスタンスには、コンテナ停止タイムアウト値を有効にするために、コンテナエージェントのバージョン 1.26.0 以上が必要です。ただし、最新のコンテナエージェントのバージョンを使用することをお勧めします。エージェントのバージョンの確認方法と最新バージョンへの更新方法については、「Amazon ECS コンテナエージェントをアップデートする」を参照してください。Amazon ECS に最適化された Linux Amazon AMI を使用している場合、インスタンスには、少なくとも `ecs-init` パッケージの 1.26.0-1 バージョン以上が必要です。バージョン `20190301` 以降から起動されるコンテナインスタンスには、必要なバージョンのコンテナエージェントや `ecs-init` が含まれています。詳細については、「Amazon ECS に最適化された Linux AMI」を参照してください。

システムコントロール

`systemControls`

型: SystemControl オブジェクト

必須: いいえ

コンテナ内で設定する名前空間カーネルパラメータのリスト。このパラメータは docker create-container コマンドの Sysctls にマッピングされ、--sysctl オプションは docker run にマッピングされます。例えば、接続をより長く維持するように net.ipv4.tcp_keepalive_time 設定を構成できます。

awsvpc または host ネットワークモードも使用する単一のタスクで、複数のコンテナに対してネットワーク関連の systemControls パラメータを指定することは推奨されません。これを行うと、次のような欠点があります。

- awsvpc ネットワークモードを使用するタスクの場合、systemControls をコンテナ用に設定した場合、タスク内のすべてのコンテナに適用されます。単一のタスクの複数のコンテナに対して異なる systemControls を設定すると、最後に開始されたコンテナによって、有効になる systemControls が決定します。
- host ネットワークモードを使用するタスクでは、ネットワーク名前空間の systemControls はサポートされていません。

タスク内でコンテナに使用するため IPC リソース名前空間を設定している場合、システムコントロールには以下の条件が適用されます。詳細については、「IPC モード」を参照してください。

- host IPC モードを使用するタスクの場合、IPC 名前空間の systemControls はサポートされていません。
- task IPC モードを使用するタスクでは、IPC 名前空間の systemControls 値が、タスク内のすべてのコンテナに適用されます。

> **Note**
>
> このパラメータは Windows コンテナではサポートされません。

```
"systemControls": [
    {
         "namespace":"string",
         "value":"string"
    }
]
```

namespace

タイプ: 文字列

必須: いいえ

value を設定する名前空間カーネルパラメータ。

有効な IPC 名前空間の値: "fs.mqueue.*" で開始する "kernel.msgmax" | "kernel.msgmnb" | "kernel.msgmni" | "kernel.sem" | "kernel.shmall" | "kernel.shmmax" | "kernel.shmmni" | "kernel.shm_rmid_forced"、および Sysctls

有効なネットワーク名前空間値: "net.*" で始まる Sysctls

value

タイプ: 文字列

必須: いいえ

namespace で指定された名前空間カーネルパラメータの値。

インタラクティブ

interactive

タイプ: ブール値

必須: いいえ

このパラメータが true の場合、stdin または tty を割り当てる必要がある、コンテナ化されたアプリケーションをデプロイすることができます。このパラメータは docker create-container コマンドの OpenStdin にマッピングされ、--interactive オプションは docker run にマッピングされます。

デフォルトは false です。

擬似ターミナル

pseudoTerminal

タイプ: ブール値

必須: いいえ

このパラメータが true の場合、TTY が割り当てられます。このパラメータは docker create-container コマンドの Tty にマッピングされ、--tty オプションは docker run にマッピングされます。

デフォルトは `false` です。

Elastic Inference アクセラレーター名 (廃止)

タスク定義用の Elastic Inference アクセラレーターのリソース要件。

> **Note**
>
> Amazon Elastic Inference (EI) は利用できなくなりました。

以下のパラメータをタスク定義で使用できます。

`deviceName`

タイプ: 文字列

必須: はい

Elastic Inference アクセラレーターのデバイス名。`deviceName` は、コンテナ定義でも参照されます。`Elastic Inference accelerator` を参照してください。

`deviceType`

タイプ: 文字列

必須: はい

使用する Elastic Inference アクセラレーター。

タスク配置の制約事項

タスク定義の登録時、Amazon ECS でのタスク配置方法をカスタマイズするタスク配置の制約を指定できます。

アベイラビリティーゾーン、インスタンスタイプ、またはカスタム属性に基づいたタスク配置の制約を使用できます。詳細については、「Amazon ECS がタスクに使用するコンテナインスタンスを定義する」を参照してください。

以下のパラメータをコンテナ定義で使用できます。

`expression`

型: 文字列

必須: いいえ

制約に適用されるクラスタークエリ言語表現。詳細については、「Amazon ECS タスク用のコンテナインスタンスを定義する式を作成する」を参照してください。

`type`

型: 文字列

必須: はい

制約のタイプ。選択対象を有効な候補グループに制約するには、`memberOf` を使用します。

プロキシ設定

`proxyConfiguration`

タイプ: ProxyConfiguration オブジェクト

必須: いいえ

App Mesh プロキシ設定の詳細。

EC2 を使用するタスクでプロキシ設定を有効にする場合、コンテナインスタンスには、コンテナエージェントのバージョン 1.26.0 以上と `ecs-init` パッケージのバージョン 1.26.0-1 以上が必要です。コンテナインスタンスが Amazon ECS に最適化された AMI バージョン `20190301` 以降から起動される場合、コンテナエージェントおよび `ecs-init` の必要なバージョンが含まれます。詳細については、「Amazon ECS に最適化された Linux AMI」を参照してください。

> **Note**
>
> このパラメータは Windows コンテナではサポートされません。

```
"proxyConfiguration": {
    "type": "APPMESH",
    "containerName": "string",
```

```
    "properties": [
        {
           "name": "string",
           "value": "string"
        }
    ]
}
```

type

型: 文字列

重要な値: APPMESH

必須: いいえ

プロキシのタイプ。APPMESH はサポートされる唯一の値です。

containerName

型: 文字列

必須: はい

App Mesh プロキシとして機能するコンテナの名前です。

properties

タイプ: パラメータオブジェクトの配列

必須: いいえ

Container Network Interface(CNI) プラグインを提供するネットワーク構成パラメータのセットで、キーと値のペアとして指定されます。

- IgnoredUID - (必須) コンテナ定義の user パラメータで定義されるプロキシコンテナのユーザー ID (UID)。これは、プロキシがそれ自体のトラフィックを無視するようにするために使用されます。IgnoredGID を指定した場合は、このフィールドは空にできます。
- IgnoredGID - (必須) コンテナ定義の user パラメータで定義されるプロキシコンテナのグループ ID (GID)。これは、プロキシがそれ自体のトラフィックを無視するようにするために使用されます。IgnoredUID を指定した場合は、このフィールドは空にできます。
- AppPorts - (必須) アプリケーションが使用するポートのリスト。これらのポートへのネットワークトラフィックは ProxyIngressPort および ProxyEgressPort に転送されます。

- ProxyIngressPort - (必須) AppPorts への着信トラフィックが誘導されるポートを指定します。
- ProxyEgressPort - (必須) AppPorts からの発信トラフィックが誘導されるポートを指定します。
- EgressIgnoredPorts – (必須) 指定されたこれらのポートに向かうアウトバウンドトラフィックは無視され、ProxyEgressPort にリダイレクトされません。空のリストを指定できます。
- EgressIgnoredIPs – (必須) 指定されたこれらの IP アドレスに向かうアウトバウンドトラフィックは無視され、ProxyEgressPort にリダイレクトされません。空のリストを指定できます。

name

型: 文字列

必須: いいえ

キーと値のペアの名前。

value

型: 文字列

必須: いいえ

キーと値のペアの値。

ボリューム

タスク定義を登録する際、コンテナインスタンスの Docker デーモンに渡されるボリュームのリストを任意で指定することができます。すると、同じコンテナインスタンス上の他のコンテナで使用できるようになります。

使用できるデータボリュームの種類は以下のとおりです。

- Amazon EBS ボリューム – データ集約型のコンテナ化されたワークロード向けに、費用対効果が高く、耐久性があり、高性能なブロックストレージを提供します。スタンドアロンタスクを実行するとき、またはサービスを作成または更新するときに、Amazon ECS タスクごとに 1 つの Amazon EBS ボリュームをアタッチできます。Amazon EBS ボリュームは、Linux タスクでサ

ポートされています。詳細については、「Amazon ECS での Amazon EBS ボリュームの使用」を参照してください。

- Amazon EFS ボリューム - Amazon ECS タスクで使用するためのシンプルかつスケーラブルで、永続的なファイルストレージを提供します。Amazon EFSでは、ストレージ容量は伸縮性があります。この容量は、ファイルの追加や削除に伴い自動的に拡大および縮小されます。アプリケーションは、必要なときに必要なストレージを確保できます。Amazon EFS ボリュームがサポートされています。詳細については、「Amazon ECS での Amazon EFS ボリュームの使用」を参照してください。
- FSx for Windows File Server ボリューム – 完全マネージド型の Microsoft Windows ファイルサーバーを提供します。これらのファイルサーバは、Windows ファイルシステムによってバックアップされています。Amazon ECS と共に FSx for Windows File Server を使用する場合、永続的、分散型、共有型、および静的なファイルストレージを使用して、Windows タスクをプロビジョニングすることが可能です。詳細については、「Amazon ECS での FSx for Windows File Server ボリュームの使用」を参照してください。

  このオプションは Fargate の Windows コンテナではサポートされません。
- Docker ボリューム – ホストの Amazon EC2 インスタンスで `/var/lib/docker/volumes` に作成される Docker マネージドボリューム。Docker ボリュームドライバー (プラグインとも呼ばれる) は、ボリュームを外部ストレージシステム (Amazon EBS など) と統合するために使用します。組み込みの `local` ボリュームドライバーまたはサードパーティーのボリュームドライバーを使用できます。Docker ボリュームは、Amazon EC2 インスタンスでタスクを実行する場合にのみサポートされます。Windows コンテナでは、`local` ドライバーの使用のみがサポートされます。Docker ボリュームを使用するには、タスク定義で `dockerVolumeConfiguration` を指定します。
- バインドマウント – ホストマシン上のファイルやディレクトリがコンテナにマウントされます。バインドマウントホストボリュームがサポートされています。バインドマウントのホストボリュームを使用するには、タスク定義で `host` およびオプションの `sourcePath` 値を使用します。

詳細については、「Amazon ECS タスクのストレージオプション」を参照してください。

以下のパラメータをコンテナ定義で使用できます。

`name`

タイプ: 文字列

必須: いいえ

ボリュームの名前。最大 255 文字の英字 (大文字と小文字の区別あり)、数字、ハイフン (-)、アンダースコア (_) を使用できます。この名前は、コンテナ定義 `mountPoints` オブジェクトの `sourceVolume` パラメータで参照されます。

`host`

必須: いいえ

`host` パラメーターは、バインドマウントのライフサイクルを、タスクではなくホスト Amazon EC2 インスタンスと、それが格納されている場所に関連付けるために使用されます。`host` パラメーターが空の場合、Docker デーモンはデータボリュームのホストパスを割り当てますが、関連付けられたコンテナの実行が停止した後にデータが保持されるとは限りません。

Windows コンテナは `$env:ProgramData` と同じドライブに全部のディレクトリをマウントできます。

> **Note**
>
> `sourcePath` パラメータは、Amazon EC2 インスタンスまたは Amazon ECS マネージドインスタンスでホストされているタスクを使用する場合にのみサポートされます。

`sourcePath`

タイプ: 文字列

必須: いいえ

`host` パラメータを使用する場合は、`sourcePath` を指定して、コンテナに表示されるホスト Amazon EC2 インスタンスのパスを宣言します。このパラメータが空の場合は、Docker デーモンによってホストパスが割り当てられます。`host` パラメータに `sourcePath` の場所が含まれている場合、データボリュームは手動で削除するまでホスト Amazon EC2 インスタンスの指定された場所に保持されます。`sourcePath` の値がホスト Amazon EC2 インスタンスに存在しない場合は、Docker デーモンによって作成されます。その場所が存在する場合は、ソースパスフォルダの内容がエクスポートされます。

`configuredAtLaunch`

タイプ: ブール値

必須: いいえ

起動時にボリュームを設定可能にするかどうかを指定します。true に設定すると、スタンドアロンタスクを実行するとき、またはサービスを作成または更新するときにボリュームを設定できます。true に設定すると、タスク定義内で別のボリューム設定を提供することはできません。タスクにアタッチする Amazon EBS ボリュームを設定するには、このパラメータを true に設定する必要があります。configuredAtLaunch を true に設定し、ボリューム設定を起動フェーズに先送りすることで、ボリュームタイプや特定のボリューム設定に限定されないタスク定義を作成できます。こうすることで、タスク定義をさまざまな実行環境で再利用できるようになります。詳細については、「Amazon EBS ボリューム」を参照してください。

dockerVolumeConfiguration

タイプ: DockerVolumeConfiguration オブジェクト

必須: いいえ

このパラメータは、Docker ボリュームを使用する場合に指定します。Docker ボリュームは、EC2 インスタンスでタスクを実行する場合にのみサポートされます。Windows コンテナでは、local ドライバーの使用のみがサポートされます。バインドマウントを使用するには、代わりに host を指定します。

scope

型: 文字列

有効な値: task | shared

必須: いいえ

Docker ボリュームのスコープ。これにより、ボリュームのライフサイクルが決定されます。Docker ボリュームの範囲が task の場合は、タスクが開始すると自動的にプロビジョンされ、タスクが停止すると破棄されます。Docker ボリュームの範囲が shared の場合は、タスクの停止後も保持されます。

autoprovision

タイプ: ブール値

デフォルト値: false

必須: いいえ

この値が true の場合、既に存在していない場合は Docker ボリュームが作成されます。このフィールドは、scope が shared の場合にのみ使用されます。scope が task の場合、このパラメータは省略する必要があります。

`driver`

タイプ: 文字列

必須: いいえ

使用する Docker ボリュームドライバー。この名前はタスク配置に使用されるため、ドライバー値は Docker で提供されているドライバー名と一致する必要があります。ドライバーが Docker プラグイン CLI を使用してインストールされた場合は、`docker plugin ls` を使用してコンテナインスタンスからドライバー名を取得します。ドライバーが別の方法でインストール済みである場合は、Docker プラグイン検出を使用してドライバー名を取得します。

`driverOpts`

タイプ: 文字列

必須: いいえ

パススルーする Docker ドライバー固有のオプションのマップ。このパラメータは、Docker の「Create a volume」セクションの `DriverOpts` にマッピングされます。

`labels`

タイプ: 文字列

必須: いいえ

Docker ボリュームに追加するカスタムメタデータ。

`efsVolumeConfiguration`

タイプ: EFSVolumeConfiguration オブジェクト

必須: いいえ

このパラメータは、Amazon EFS ボリュームを使用する場合に指定します。

`fileSystemId`

型: 文字列

必須: はい

使用する Amazon EFS ファイルシステムの ID。

`rootDirectory`

型: 文字列

必須: いいえ

ホスト内にルートディレクトリとしてマウントする Amazon EFS ファイルシステム内のディレクトリ。このパラメータを省略すると、Amazon EFS ボリュームのルートが使用されます。/ を指定すると、このパラメータを省略した場合と同じ結果になります。

> **Important**
>
> `authorizationConfig` で EFS アクセスポイントが指定されている場合は、ルートディレクトリパラメータを省略するか / に設定して、EFS アクセスポイントにパスを設定する必要があります。

`transitEncryption`

タイプ: 文字列

有効な値: `ENABLED` | `DISABLED`

必須: いいえ

Amazon ECS ホストと Amazon EFS サーバー間で、転送中の Amazon EFS データの暗号化を有効にするかどうかを指定します。Amazon EFS IAM 認可を使用する場合は、転送中の暗号化を有効にする必要があります。このパラメータを省略すると、`DISABLED` のデフォルト値が使用されます。詳細については、Amazon Elastic ファイルシステムユーザーガイドの「転送中データの暗号化」を参照してください。

`transitEncryptionPort`

タイプ: 整数

必須: いいえ

Amazon ECS ホストと Amazon EFS サーバーとの間で、暗号化されたデータを送信するときに使用するポート。転送暗号化ポートを指定しない場合、タスクでは Amazon EFS マウントヘルパーが使用するポート選択戦略が使用されます。詳細については、[Amazon Elastic File System User Guide] (Amazon Elastic File System ユーザーガイド) の[EFS Mount Helper] (EFS マウントヘルパー) を参照してください。

`authorizationConfig`

タイプ: EFSAuthorizationConfig オブジェクト

必須: いいえ

Amazon EFS ファイルシステムに対する認可構成の詳細。

`accessPointId`

型: 文字列

必須: いいえ

使用するアクセスポイント ID。アクセスポイントが指定されている場合は、`efsVolumeConfiguration` のルートディレクトリ値を省略するか `/` に設定して、EFS アクセスポイントにパスを設定する必要があります。アクセスポイントを使用する場合は、`EFSVolumeConfiguration` で転送中の暗号化を有効にする必要があります。詳細については、Amazon Elastic ファイルシステムユーザーガイドの Amazon EFS アクセスポイントの使用を参照してください。

`iam`

型: 文字列

有効な値: `ENABLED` | `DISABLED`

必須: いいえ

タスク定義で定義した Amazon ECS タスクの IAM ロールを、Amazon EFS ファイルシステムのマウント時に使用するかどうかを指定します。使用する場合は、`EFSVolumeConfiguration` で転送中の暗号化を有効にする必要があります。このパラメータを省略すると、`DISABLED` のデフォルト値が使用されます。詳細については、「タスク用の IAM ロール」を参照してください。

`FSxWindowsFileServerVolumeConfiguration`

タイプ: FSxWindowsFileServerVolumeConfiguration オブジェクト

必須: はい

このパラメータは、タスクストレージに Amazon FSx for Windows File Server ファイルシステムを使用する場合に指定します。

`fileSystemId`

タイプ: 文字列

必須: はい

使用する FSx for Windows File Server ファイルシステムID。

`rootDirectory`

型: 文字列

必須: はい

ホスト内にルートディレクトリとしてマウントする FSx for Windows File Server ファイルシステム内のディレクトリ。

`authorizationConfig`

`credentialsParameter`

型: 文字列

必須: はい

認可の認証情報オプション。

options:

- AWS Secrets Manager シークレットの Amazon リソースネーム (ARN)。
- AWS Systems Manager パラメータの ARN。

`domain`

タイプ: 文字列

必須: はい

AWS Directory Service for Microsoft Active Directory (AWS Managed Microsoft AD) ディレクトリまたはセルフホスト型 EC2 Active Directory によってホストされる完全修飾ドメイン名。

タグ

タスク定義の登録する際、タスク定義に適用されるメタデータタグをオプションで指定できます。タグは、タスク定義を分類して組織化するのに役立ちます。各タグはキーとオプションの値で構成され

ます。両方を定義します。詳細については、「Amazon ECS リソースにタグ付けする」を参照してください。

> **Important**
>
> タグには、個人が特定可能な情報や、機密情報あるいは秘匿性の高い情報は追加しないでください。タグは、多くの AWS のサービス (請求など) からアクセスできます。タグは、プライベートデータや機密データに使用することを意図していません。

タグオブジェクトでは、次のパラメータを使用できます。

`key`

タイプ: 文字列

必須: いいえ

タグを構成するキーと値のペアの一部。キーは、より具体的なタグ値のカテゴリのように動作する、一般的なラベルです。

`value`

タイプ: 文字列

必須: いいえ

タグを構成するキーと値のペアのオプションの一部。値はタグカテゴリ (キー) の記述子として機能します。

その他のタスク定義パラメータ

以下のタスク定義パラメータは、Amazon ECS コンソールから、[Configure via JSON (JSON による設定)] オプションを使用してタスク定義を登録する際に使用します。詳細については、「コンソールを使用した Amazon ECS タスク定義の作成」を参照してください。

トピック

- IPC モード
- PID モード

- フォールトインジェクション

IPC モード

`ipcMode`

型: 文字列

必須: いいえ

タスクのコンテナで使用する IPC リソースの名前空間。有効な値は `host`、`task` または `none` です。`host` が指定されている場合、同一のコンテナインスタンス上にある (`host` IPC モードを指定した) タスク内のすべてのコンテナは、ホスト Amazon EC2 インスタンスと同じ IPC リソースを共有します。`task` が指定されている場合、指定されたタスク内のすべてのコンテナは同じ IPC リソースを共有します。`none` が指定されている場合、タスクのコンテナ内の IPC リソースはプライベートです。タスク内またはコンテナインスタンスの他のコンテナと共有されることはありません。値を指定しない場合、IPC リソース名前空間の共有はコンテナインスタンスの Docker デーモンの設定によって異なります。

`host` IPC モードを使用する場合は、意図せず IPC 名前空間が公開されるリスクが高いことに注意してください。

タスク内のコンテナに、`systemControls` を使用して名前空間のカーネルパラメータを設定している場合は、以下の点が IPC リソース名前空間に適用されます。

- `host` IPC モードを使用するタスクの場合、`systemControls` に関連する IPC 名前空間はサポートされません。
- `task` IPC モードを使用するタスクでは、IPC 名前空間に関連する `systemControls` が、タスク内のすべてのコンテナに適用されます。

PID モード

`pidMode`

タイプ: 文字列

有効な値: `host` | `task`

必須: いいえ

タスクのコンテナで使用するプロセス名前空間。有効な値は `host` または `task` です。例えば、サイドカーのモニタリングでは、`pidMode` が同じタスクで実行されている他のコンテナに関する情報にアクセスする必要となる場合があります。

`host` が指定されている場合、同じコンテナインスタンスで `host` PID モードを指定したタスク内のすべてのコンテナは、ホスト Amazon EC2 インスタンスと同じプロセス名前空間を共有します。

`task` が指定されている場合、指定したタスク内のすべてのコンテナは同じプロセス名前空間を共有します。

値が指定されていない場合、デフォルトは各コンテナのプライベート名前空間です。

`host` PID モードを使用する場合は、意図せずプロセス名前空間が公開されるリスクが高いことに注意してください。

> **Note**
>
> このパラメータは Windows コンテナではサポートされません。

フォールトインジェクション

`enableFaultInjection`

タイプ: ブール値

有効な値: `true` | `false`

必須: いいえ

タスクのペイロードでこのパラメータが `true` 設定されている場合、Amazon ECS はタスクのコンテナからのフォールトインジェクションリクエストを受け入れます。デフォルトでは、このパラメータは `false` に設定されます。

Amazon ECS タスク定義テンプレート

以下に示しているのは、空のタスク定義テンプレートです。このテンプレートを使用してタスク定義を作成します。これにより、コンソールの JSON 入力領域に貼り付けるか、ファイルに保存し

て AWS CLI の --cli-input-json オプションで使用できるようになります。詳細については、「Fargate での Amazon ECS タスク定義パラメータ」を参照してください。

EC2 テンプレート

```
{
  "family": "",
  "taskRoleArn": "",
  "executionRoleArn": "",
  "networkMode": "none",
  "containerDefinitions": [
    {
      "name": "",
      "image": "",
      "repositoryCredentials": {
        "credentialsParameter": ""
      },
      "cpu": 0,
      "memory": 0,
      "memoryReservation": 0,
      "links": [""],
      "portMappings": [
        {
          "containerPort": 0,
          "hostPort": 0,
          "protocol": "tcp"
        }
      ],
      "restartPolicy": {
        "enabled": true,
        "ignoredExitCodes": [0],
        "restartAttemptPeriod": 180
      },
      "essential": true,
      "entryPoint": [""],
      "command": [""],
      "environment": [
        {
          "name": "",
          "value": ""
        }
      ],
      "environmentFiles": [
        {
```

```
          "value": "",
          "type": "s3"
        }
      ],
      "mountPoints": [
        {
          "sourceVolume": "",
          "containerPath": "",
          "readOnly": true
        }
      ],
      "volumesFrom": [
        {
          "sourceContainer": "",
          "readOnly": true
        }
      ],
      "linuxParameters": {
        "capabilities": {
          "add": [""],
          "drop": [""]
        },
        "devices": [
          {
            "hostPath": "",
            "containerPath": "",
            "permissions": ["read"]
          }
        ],
        "initProcessEnabled": true,
        "sharedMemorySize": 0,
        "tmpfs": [
          {
            "containerPath": "",
            "size": 0,
            "mountOptions": [""]
          }
        ],
        "maxSwap": 0,
        "swappiness": 0
      },
      "secrets": [
        {
          "name": "",
```

```
          "valueFrom": ""
        }
      ],
      "dependsOn": [
        {
          "containerName": "",
          "condition": "COMPLETE"
        }
      ],
      "startTimeout": 0,
      "stopTimeout": 0,
      "hostname": "",
      "user": "",
      "workingDirectory": "",
      "disableNetworking": true,
      "privileged": true,
      "readonlyRootFilesystem": true,
      "dnsServers": [""],
      "dnsSearchDomains": [""],
      "extraHosts": [
        {
          "hostname": "",
          "ipAddress": ""
        }
      ],
      "dockerSecurityOptions": [""],
      "interactive": true,
      "pseudoTerminal": true,
      "dockerLabels": {
        "KeyName": ""
      },
      "ulimits": [
        {
          "name": "nofile",
          "softLimit": 0,
          "hardLimit": 0
        }
      ],
      "logConfiguration": {
        "logDriver": "splunk",
        "options": {
          "KeyName": ""
        },
        "secretOptions": [
```

```
          {
            "name": "",
            "valueFrom": ""
          }
        ]
      },
      "healthCheck": {
        "command": [""],
        "interval": 0,
        "timeout": 0,
        "retries": 0,
        "startPeriod": 0
      },
      "systemControls": [
        {
          "namespace": "",
          "value": ""
        }
      ],
      "resourceRequirements": [
        {
          "value": "",
          "type": "InferenceAccelerator"
        }
      ],
      "firelensConfiguration": {
        "type": "fluentbit",
        "options": {
          "KeyName": ""
        }
      }
    }
  ],
  "volumes": [
    {
      "name": "",
      "host": {
        "sourcePath": ""
      },
      "configuredAtLaunch": true,
      "dockerVolumeConfiguration": {
        "scope": "shared",
        "autoprovision": true,
        "driver": "",
```

```
        "driverOpts": {
          "KeyName": ""
        },
        "labels": {
          "KeyName": ""
        }
      },
      "efsVolumeConfiguration": {
        "fileSystemId": "",
        "rootDirectory": "",
        "transitEncryption": "DISABLED",
        "transitEncryptionPort": 0,
        "authorizationConfig": {
          "accessPointId": "",
          "iam": "ENABLED"
        }
      },
      "fsxWindowsFileServerVolumeConfiguration": {
        "fileSystemId": "",
        "rootDirectory": "",
        "authorizationConfig": {
          "credentialsParameter": "",
          "domain": ""
        }
      }
    }
  ],
  "placementConstraints": [
    {
      "type": "memberOf",
      "expression": ""
    }
  ],
  "requiresCompatibilities": ["EC2"],
  "cpu": "",
  "memory": "",
  "tags": [
    {
      "key": "",
      "value": ""
    }
  ],
  "pidMode": "task",
  "ipcMode": "task",
```

```
  "proxyConfiguration": {
    "type": "APPMESH",
    "containerName": "",
    "properties": [
      {
        "name": "",
        "value": ""
      }
    ]
  },
  "inferenceAccelerators": [
    {
      "deviceName": "",
      "deviceType": ""
    }
  ],
  "ephemeralStorage": {
    "sizeInGiB": 0
  },
  "runtimePlatform": {
    "cpuArchitecture": "X86_64",
    "operatingSystemFamily": "WINDOWS_SERVER_20H2_CORE"
  }
}
```

Fargate テンプレート

Important

Fargate では、以下のいずれかの値を持つ operatingSystemFamily パラメータを含める必要があります。

- LINUX

- WINDOWS_SERVER_2019_FULL
- WINDOWS_SERVER_2019_CORE
- WINDOWS_SERVER_2022_FULL
- WINDOWS_SERVER_2022_CORE

```
{
    "family": "",
    "runtimePlatform": {"operatingSystemFamily": ""},
    "taskRoleArn": "",
    "executionRoleArn": "",
    "networkMode": "awsvpc",
    "platformFamily": "",
    "containerDefinitions": [
        {
            "name": "",
            "image": "",
            "repositoryCredentials": {"credentialsParameter": ""},
            "cpu": 0,
            "memory": 0,
            "memoryReservation": 0,
            "links": [""],
            "portMappings": [
                {
                    "containerPort": 0,
                    "hostPort": 0,
                    "protocol": "tcp"
                }
            ],
            "essential": true,
            "entryPoint": [""],
            "command": [""],
            "environment": [
                {
                    "name": "",
                    "value": ""
                }
            ],
            "environmentFiles": [
                {
                    "value": "",
                    "type": "s3"
                }
            ],
            "mountPoints": [
                {
                    "sourceVolume": "",
                    "containerPath": "",
                    "readOnly": true
```

```
            }
        ],
        "volumesFrom": [
            {
                "sourceContainer": "",
                "readOnly": true
            }
        ],
        "linuxParameters": {
            "capabilities": {
                "add": [""],
                "drop": [""]
            },
            "devices": [
                {
                    "hostPath": "",
                    "containerPath": "",
                    "permissions": ["read"]
                }
            ],
            "initProcessEnabled": true,
            "sharedMemorySize": 0,
            "tmpfs": [
                {
                    "containerPath": "",
                    "size": 0,
                    "mountOptions": [""]
                }
            ],
            "maxSwap": 0,
            "swappiness": 0
        },
        "secrets": [
            {
                "name": "",
                "valueFrom": ""
            }
        ],
        "dependsOn": [
            {
                "containerName": "",
                "condition": "HEALTHY"
            }
        ],
```

```
            "startTimeout": 0,
            "stopTimeout": 0,
            "hostname": "",
            "user": "",
            "workingDirectory": "",
            "disableNetworking": true,
            "privileged": true,
            "readonlyRootFilesystem": true,
            "dnsServers": [""],
            "dnsSearchDomains": [""],
            "extraHosts": [
                {
                    "hostname": "",
                    "ipAddress": ""
                }
            ],
            "dockerSecurityOptions": [""],
            "interactive": true,
            "pseudoTerminal": true,
            "dockerLabels": {"KeyName": ""},
            "ulimits": [
                {
                    "name": "msgqueue",
                    "softLimit": 0,
                    "hardLimit": 0
                }
            ],
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {"KeyName": ""},
                "secretOptions": [
                    {
                        "name": "",
                        "valueFrom": ""
                    }
                ]
            },
            "healthCheck": {
                "command": [""],
                "interval": 0,
                "timeout": 0,
                "retries": 0,
                "startPeriod": 0
            },
```

```
            "systemControls": [
                {
                    "namespace": "",
                    "value": ""
                }
            ],
            "resourceRequirements": [
                {
                    "value": "",
                    "type": "GPU"
                }
            ],
            "firelensConfiguration": {
                "type": "fluentd",
                "options": {"KeyName": ""}
            }
        }
    ],
    "volumes": [
        {
            "name": "",
            "host": {"sourcePath": ""},
            "configuredAtLaunch":true,
            "dockerVolumeConfiguration": {
                "scope": "task",
                "autoprovision": true,
                "driver": "",
                "driverOpts": {"KeyName": ""},
                "labels": {"KeyName": ""}
            },
            "efsVolumeConfiguration": {
                "fileSystemId": "",
                "rootDirectory": "",
                "transitEncryption": "ENABLED",
                "transitEncryptionPort": 0,
                "authorizationConfig": {
                    "accessPointId": "",
                    "iam": "ENABLED"
                }
            }
        }
    ],
    "requiresCompatibilities": ["FARGATE"],
    "cpu": "",
```

```
    "memory": "",
    "tags": [
        {
            "key": "",
            "value": ""
        }
    ],
    "ephemeralStorage": {"sizeInGiB": 0},
    "pidMode": "task",
    "ipcMode": "none",
    "proxyConfiguration": {
        "type": "APPMESH",
        "containerName": "",
        "properties": [
            {
                "name": "",
                "value": ""
            }
        ]
    },
    "inferenceAccelerators": [
        {
            "deviceName": "",
            "deviceType": ""
        }
    ]
}
```

このタスク定義テンプレートは、以下の AWS CLI コマンドにより生成できます。

```
aws ecs register-task-definition --generate-cli-skeleton
```

Amazon ECS のタスク定義の例

これらの例とスニペットをコピーして、ユーザー自身のタスク定義の作成を開始することができます。

例をコピーして、コンソールで [JSON 経由の設定] オプションを使用するときに貼り付けることができます。アカウント ID を使用するなど、例を必ずカスタマイズしてください。スニペットはタスク定義 JSON に含めることができます。詳細については、「コンソールを使用した Amazon ECS

タスク定義の作成」および「Fargate での Amazon ECS タスク定義パラメータ」を参照してください。

その他のタスク定義の例については、GitHub の「AWS のタスク定義例」を参照してください。

トピック

- Web サーバー
- splunk ログドライバー
- fluentd ログドライバー
- gelf ログドライバー
- 外部インスタンス上のワークロード
- Amazon ECR イメージとタスク定義 IAM ロール
- コマンドによるエントリポイント
- コンテナの依存関係
- タスク定義のボリューム
- Windows のサンプルのタスク定義

Web サーバー

以下は、Web サーバーをセットアップするために、Fargate で Linux コンテナを使用するタスク定義の例です。

```
{
   "containerDefinitions": [
      {
         "command": [
            "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample App</
title> <style>body {margin-top: 40px; background-color: #333;} </style> </
head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1>
 <h2>Congratulations!</h2> <p>Your application is now running on a container in Amazon
 ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html && httpd-
foreground\""
         ],
         "entryPoint": [
            "sh",
            "-c"
         ],
```

```
         "essential": true,
         "image": "public.ecr.aws/docker/library/httpd:2.4",
         "logConfiguration": {
            "logDriver": "awslogs",
            "options": {
               "awslogs-group" : "/ecs/fargate-task-definition",
               "awslogs-region": "us-east-1",
               "awslogs-stream-prefix": "ecs"
            }
         },
         "name": "sample-fargate-app",
         "portMappings": [
            {
               "containerPort": 80,
               "hostPort": 80,
               "protocol": "tcp"
            }
         ]
      }
   ],
   "cpu": "256",
   "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
   "family": "fargate-task-definition",
   "memory": "512",
   "networkMode": "awsvpc",
   "runtimePlatform": {
        "operatingSystemFamily": "LINUX"
    },
   "requiresCompatibilities": [
       "FARGATE"
    ]
}
```

以下は、Web サーバーをセットアップするために、Fargate で Windows コンテナを使用するタスク定義の例です。

```
{
    "containerDefinitions": [
        {
            "command": ["New-Item -Path C:\\inetpub\\wwwroot\\index.html -Type file
 -Value '<html> <head> <title>Amazon ECS Sample App</title> <style>body {margin-top:
 40px; background-color: #333;} </style> </head><body> <div style=color:white;text-
align:center> <h1>Amazon ECS Sample App</h1> <h2>Congratulations!</h2> <p>Your
```

```
 application is now running on a container in Amazon ECS.</p>'; C:\\ServiceMonitor.exe
 w3svc"],
            "entryPoint": [
                "powershell",
                "-Command"
            ],
            "essential": true,
            "cpu": 2048,
            "memory": 4096,
            "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-
ltsc2019",
            "name": "sample_windows_app",
            "portMappings": [
                {
                    "hostPort": 80,
                    "containerPort": 80,
                    "protocol": "tcp"
                }
            ]
        }
    ],
    "memory": "4096",
    "cpu": "2048",
    "networkMode": "awsvpc",
    "family": "windows-simple-iis-2019-core",
    "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
    "runtimePlatform": {"operatingSystemFamily": "WINDOWS_SERVER_2019_CORE"},
    "requiresCompatibilities": ["FARGATE"]
}
```

splunk ログドライバー

以下のスニペットでは、タスク定義で、リモートサービスにログを送信する splunk ログドライバーを使用する方法を示しています。Splunk トークンパラメータは、機密データとして扱われる可能性があるため、シークレットのオプションとして指定しています。詳細については、「Amazon ECS コンテナに機密データを渡す」を参照してください。

```
"containerDefinitions": [{
  "logConfiguration": {
   "logDriver": "splunk",
   "options": {
    "splunk-url": "https://cloud.splunk.com:8080",
    "tag": "tag_name",
```

```
   },
   "secretOptions": [{
    "name": "splunk-token",
    "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:splunk-token-
KnrBkD"
}],
```

fluentd ログドライバー

以下のスニペットでは、タスク定義で、リモートサービスにログを送信する fluentd ログドライバーを使用する方法を示しています。fluentd-address 値は機密データとして扱われる可能性があるため、シークレットのオプションとして指定しています。詳細については、「Amazon ECS コンテナに機密データを渡す」を参照してください。

```
"containerDefinitions": [{
 "logConfiguration": {
  "logDriver": "fluentd",
  "options": {
   "tag": "fluentd demo"
  },
  "secretOptions": [{
   "name": "fluentd-address",
   "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:fluentd-address-
KnrBkD"
  }]
 },
 "entryPoint": [],
 "portMappings": [{
             "hostPort": 80,
             "protocol": "tcp",
             "containerPort": 80
             },
             {
  "hostPort": 24224,
  "protocol": "tcp",
  "containerPort": 24224
 }]
}],
```

gelf ログドライバー

以下のスニペットでは、タスク定義で、Gelf ログを入力として受け取る Logstash を実行しているリモートホストにログを送信する gelf ログドライバーを使用する方法を示しています。詳細については、「logConfiguration」を参照してください。

```
"containerDefinitions": [{
 "logConfiguration": {
  "logDriver": "gelf",
  "options": {
   "gelf-address": "udp://logstash-service-address:5000",
   "tag": "gelf task demo"
  }
 },
 "entryPoint": [],
 "portMappings": [{
   "hostPort": 5000,
   "protocol": "udp",
   "containerPort": 5000
  },
  {
   "hostPort": 5000,
   "protocol": "tcp",
   "containerPort": 5000
  }
 ]
}],
```

外部インスタンス上のワークロード

Amazon ECS タスク定義を登録するときは、requiresCompatibilitiesパラメーターを選択し、EXTERNALで、外部インスタンスで Amazon ECS ワークロードを実行するときに、タスク定義が使用できる互換性があることを検証します。コンソールを使用してタスク定義を登録する場合は、JSON エディターを使用する必要があります。詳細については、「コンソールを使用した Amazon ECS タスク定義の作成」を参照してください。

> **Important**
>
> タスクにタスク実行 IAM ロールが必要な場合は、タスク定義で指定されていることを確認してください。

ワークロードをデプロイするときは、EXTERNAL起動タイプを、サービスの作成時またはスタンドアロンタスクの実行時に選択します。

以下は、タスク定義の例です。

Linux

```
{
 "requiresCompatibilities": [
  "EXTERNAL"
 ],
 "containerDefinitions": [{
  "name": "nginx",
  "image": "public.ecr.aws/nginx/nginx:latest",
  "memory": 256,
  "cpu": 256,
  "essential": true,
  "portMappings": [{
   "containerPort": 80,
   "hostPort": 8080,
   "protocol": "tcp"
  }]
 }],
 "networkMode": "bridge",
 "family": "nginx"
}
```

Windows

```
{
 "requiresCompatibilities": [
  "EXTERNAL"
 ],
 "containerDefinitions": [{
  "name": "windows-container",
  "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-ltsc2019",
  "memory": 256,
  "cpu": 512,
  "essential": true,
  "portMappings": [{
   "containerPort": 80,
   "hostPort": 8080,
   "protocol": "tcp"
```

```
  }]
 }],
 "networkMode": "bridge",
 "family": "windows-container"
}
```

Amazon ECR イメージとタスク定義 IAM ロール

以下のスニペットでは、aws-nodejs-sample という名前の Amazon ECR イメージで、123456789012.dkr.ecr.us-west-2.amazonaws.com レジストリの v1 タグを使用しています。このタスクのコンテナは、arn:aws:iam::123456789012:role/AmazonECSTaskS3BucketRole ロールから IAM アクセス許可を継承します。詳細については、「Amazon ECS タスクの IAM ロール」を参照してください。

```
{
    "containerDefinitions": [
        {
            "name": "sample-app",
            "image": "123456789012.dkr.ecr.us-west-2.amazonaws.com/aws-nodejs-
sample:v1",
            "memory": 200,
            "cpu": 10,
            "essential": true
        }
    ],
    "family": "example_task_3",
    "taskRoleArn": "arn:aws:iam::123456789012:role/AmazonECSTaskS3BucketRole"
}
```

コマンドによるエントリポイント

以下のスニペットでは、エントリポイントとコマンド引数を使用する Docker コンテナの構文を示しています。このコンテナは example.com に対して ping を 4 回実行してから終了します。

```
{
    "containerDefinitions": [
        {
            "memory": 32,
            "essential": true,
            "entryPoint": ["ping"],
```

```
            "name": "alpine_ping",
            "readonlyRootFilesystem": true,
            "image": "alpine:3.4",
            "command": [
                "-c",
                "4",
                "example.com"
            ],
            "cpu": 16
        }
    ],
    "family": "example_task_2"
}
```

コンテナの依存関係

このスニペットでは、コンテナの依存関係が指定されている複数のコンテナを含むタスク定義の構文を示しています。次のタスク定義では、app コンテナ が開始される前に、envoy コンテナが必要とされるコンテナのヘルスチェックパラメータにより決定される正常なステータスに達する必要があります。詳細については、「コンテナの依存関係」を参照してください。

```
{
  "family": "appmesh-gateway",
  "runtimePlatform": {
        "operatingSystemFamily": "LINUX"
  },
  "proxyConfiguration":{
      "type": "APPMESH",
      "containerName": "envoy",
      "properties": [
          {
              "name": "IgnoredUID",
              "value": "1337"
          },
          {
              "name": "ProxyIngressPort",
              "value": "15000"
          },
          {
              "name": "ProxyEgressPort",
              "value": "15001"
          },
          {
```

```
                "name": "AppPorts",
                "value": "9080"
            },
            {
                "name": "EgressIgnoredIPs",
                "value": "169.254.170.2,169.254.169.254"
            }
        ]
    },
    "containerDefinitions": [
      {
        "name": "app",
        "image": "application_image",
        "portMappings": [
          {
            "containerPort": 9080,
            "hostPort": 9080,
            "protocol": "tcp"
          }
        ],
        "essential": true,
        "dependsOn": [
          {
            "containerName": "envoy",
            "condition": "HEALTHY"
          }
        ]
      },
      {
        "name": "envoy",
        "image": "840364872350.dkr.ecr.region-code.amazonaws.com/aws-appmesh-
envoy:v1.15.1.0-prod",
        "essential": true,
        "environment": [
          {
            "name": "APPMESH_VIRTUAL_NODE_NAME",
            "value": "mesh/meshName/virtualNode/virtualNodeName"
          },
          {
            "name": "ENVOY_LOG_LEVEL",
            "value": "info"
          }
        ],
        "healthCheck": {
```

```
        "command": [
          "CMD-SHELL",
          "echo hello"
        ],
        "interval": 5,
        "timeout": 2,
        "retries": 3
      }
    }
  ],
  "executionRoleArn": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
  "networkMode": "awsvpc"
}
```

タスク定義のボリューム

以下の情報を参考にして、タスクでボリュームを指定する方法を理解してください。

- Amazon EBS ボリュームの設定方法については、「Amazon ECS のデプロイ時に Amazon EBS ボリューム設定を指定する」を参照してください。
- Amazon EFS ボリュームの設定方法については、「コンソールを使用した Amazon ECS での Amazon EFS ファイルシステムの設定」を参照してください。
- FSx for Windows File Server ボリュームの設定方法については、「Amazon ECS 用の FSx for Windows File Server ファイルシステムを設定する方法について説明します。」を参照してください。
- Docker ボリュームの設定方法については、「Amazon ECS の Docker ボリュームの例」を参照してください。
- バインドマウントの設定方法の詳細については、「Amazon ECS のバインドマウントの例」を参照してください。

Windows のサンプルのタスク定義

Amazon ECS で Windows コンテナを使い始める際に役立つタスク定義サンプルを以下に示します。

Example Windows 用の Amazon ECS コンソールサンプルアプリケーション

次のタスク定義は、Amazon ECS の初回実行ウィザードで生成される Amazon ECS コンソールのサンプルアプリケーションです。`microsoft/iis` Windows コンテナイメージを使用するために移植されています。

```
{
  "family": "windows-simple-iis",
  "containerDefinitions": [
    {
      "name": "windows_sample_app",
      "image": "mcr.microsoft.com/windows/servercore/iis",
      "cpu": 1024,
      "entryPoint":["powershell", "-Command"],
      "command":["New-Item -Path C:\\inetpub\\wwwroot\\index.html -Type file -
Value '<html> <head> <title>Amazon ECS Sample App</title> <style>body {margin-top:
 40px; background-color: #333;} </style> </head><body> <div style=color:white;text-
align:center> <h1>Amazon ECS Sample App</h1> <h2>Congratulations!</h2> <p>Your
 application is now running on a container in Amazon ECS.</p>'; C:\\ServiceMonitor.exe
 w3svc"],
      "portMappings": [
        {
          "protocol": "tcp",
          "containerPort": 80
        }
      ],
      "memory": 1024,
      "essential": true
    }
  ],
  "networkMode": "awsvpc",
  "memory": "1024",
  "cpu": "1024"
}
```

Amazon ECS クラスター

Amazon ECS クラスターは、コンテナ化されたアプリケーションのインフラストラクチャ容量を提供する、タスクまたはサービスの論理グループです。クラスター作成時には、3 つの主要なインフラストラクチャタイプから選択します。各タイプは、異なるユースケースと運用要件に合わせて最適化されています。

適切なクラスタータイプの選択

Amazon ECS では、クラスター用に 3 つのインフラストラクチャタイプが用意されています。ワークロード要件、運用設定、コスト最適化の目標に最適なタイプを選択します。

Amazon ECS マネージドインスタンス (推奨)

ほとんどのワークロードに最適 – AWS は、プロビジョニング、パッチ適用、スケーリングなど、基盤となる Amazon EC2 インスタンスを完全に管理します。このオプションを使用すると、パフォーマンス、コスト効率、運用のシンプルさについて最適なバランスが得られます。

次の場合に使用:

- AWS にインフラストラクチャ管理を任せたい
- 自動最適化による費用対効果の高いコンピューティングが必要
- インフラストラクチャではなくアプリケーションに集中したい
- 柔軟なスケーリングで予測可能なパフォーマンスが必要

Fargate

サーバーレスコンピューティング – インフラストラクチャを管理せずに、タスクが使用するリソースに対してのみ料金を支払うサーバーレスコンピューティング。可変ワークロードや、すぐに使用開始する場合に最適です。

次の場合に使用:

- 完全にサーバーレスなオペレーションが必要
- 予測不可能なワークロードまたは可変ワークロードがある
- 運用上のオーバーヘッドを最小限に抑えたい
- 迅速なデプロイとスケーリングが必要

Amazon EC2 インスタンス

フルコントロール – インスタンスの選択、設定、メンテナンスなど、基盤となる Amazon EC2 インスタンスを直接管理します。

次の場合に使用:

- 特定のインスタンスタイプまたは設定が必要
- 活用する既存の Amazon EC2 インフラストラクチャがある
- カスタム AMI または特殊なソフトウェアが必要
- 基盤となるインフラストラクチャを最大限制御する必要がある

> **Note**
>
> Amazon ECS マネージドインスタンスは、パフォーマンス、コスト最適化、運用のシンプルさを適切に組み合わせることができる一方で、AWS がインフラストラクチャ管理タスクを処理できるようになるため、ほとんどの新しいワークロードに推奨される選択肢です。

クラスターコンポーネント

クラスターは、インフラストラクチャの容量に加えて以下のリソースで構成されます。

- タスクとサービスが実行されるネットワーク (VPC およびサブネット)

  キャパシティとして Amazon ECS マネージドインスタンスまたは Amazon EC2 インスタンスを使用する場合、サブネットはアベイラビリティーゾーン、ローカルゾーン、Wavelength ゾーン、または AWS Outposts に含められます。
- オプションの名前空間

  名前空間は、Service Connect とのサービス間通信に使用されます。
- モニタリングオプション

  CloudWatch Container Insights は追加料金がかかるフルマネージドサービスです。Amazon ECS のメトリクスとログを自動で収集、集計、要約します。

クラスターの概念

Amazon ECS クラスターに関する全般的な概念を以下に示します。

- クラスターを作成してリソースを分離します。
- クラスターは AWS リージョン 固有です。
- クラスターは、次のいずれかの状態になります。

  ACTIVE

  クラスターはタスクを受け入れる準備ができており、該当する場合は、クラスターにコンテナインスタンスを登録できます。

  PROVISIONING

  クラスターにはキャパシティプロバイダーが関連付けられており、キャパシティプロバイダーに必要なリソースが作成されています。

  プロビジョン解除中

  クラスターにはキャパシティプロバイダーが関連付けられており、キャパシティプロバイダーに必要なリソースが削除されています。

  FAILED

  クラスターにはキャパシティプロバイダーが関連付けられており、キャパシティプロバイダーに必要なリソースの作成に失敗しました。

  INACTIVE

  クラスターは削除されました。`INACTIVE` ステータスのクラスターは、一定期間アカウント内で検出可能なままになる場合があります。この動作は今後変更される可能性があるため、`INACTIVE` クラスターの永続的な使用を前提としないようにしてください。
- クラスターには、Amazon EC2 マネージドインスタンス、AWS Fargate、Amazon EC2 インスタンス、または外部インスタンスでホストされているタスクを混在させることができます。タスクは、起動タイプまたはキャパシティプロバイダー戦略として、Amazon ECS マネージドインスタンス、Fargate、または EC2 のインフラストラクチャで実行できます。EC2 キャパシティプロバイダーを使用する場合、Amazon ECS は Amazon EC2 Auto Scaling グループのキャパシティを追跡およびスケーリングしません。
- クラスターには、Amazon ECS マネージドインスタンスキャパシティプロバイダー、Auto Scaling グループキャパシティプロバイダー、および Fargate キャパシティプロバイダーの組み合わせを含めることができます。キャパシティプロバイダー戦略には、Amazon ECS マネージドインスタン

スキャパシティプロバイダー、Auto Scaling グループキャパシティプロバイダー、または Fargate キャパシティプロバイダーのみを含めることができます。

- Amazon ECS マネージドインスタンスと EC2、または Auto Scaling グループのキャパシティプロバイダーには、異なるインスタンスタイプを使用できます。インスタンスは、一度に 1 つのクラスターにしか登録できません。
- カスタム IAM ポリシーを作成することで、クラスターへのアクセスを制限できます。詳細については、「Amazon Elastic Container Service のアイデンティティベースのポリシーの例」の「Amazon ECS クラスターの例」セクションを参照してください。
- サービス自動スケーリングを使用して Fargate タスクをスケーリングできます。詳細については、「Amazon ECS サービスを自動的にスケールする」を参照してください。
- クラスターのデフォルトの Service Connect 名前空間を設定できます。デフォルトの Service Connect 名前空間を設定したら、Service Connect を有効にすることで、クラスター内で作成された新しいサービスを名前空間のクライアントサービスとして追加できます。追加の設定は必要ありません。詳細については、「Service Connect を使用して Amazon ECS サービスを短縮名で接続する」を参照してください。

キャパシティープロバイダー

Amazon ECS キャパシティープロバイダーは、クラスター内のタスクに対するインフラストラクチャのスケーリングを管理できます。各クラスターには、1 つ以上のキャパシティプロバイダーがあり、さらにオプションとしてキャパシティプロバイダー戦略があります。クラスターにデフォルトのキャパシティープロバイダー戦略を割り当てられます。キャパシティープロバイダー戦略は、クラスターの複数のキャパシティープロバイダー間にタスクを分散する方法を決定します。スタンドアロンタスクを実行するか、サービスを作成するときは、クラスターのデフォルトのキャパシティプロバイダー戦略か、クラスターのデフォルト戦略をオーバーライドするキャパシティプロバイダー戦略のいずれかを使用します。クラスターのデフォルトのキャパシティプロバイダー戦略は、タスクまたはサービスの起動タイプまたはキャパシティプロバイダー戦略を指定しない場合にのみ適用されます。これらのパラメータのいずれかを指定した場合、デフォルトの戦略は使用されません。

Amazon ECS には、クラスター用に 3 種類のキャパシティプロバイダーを提供します。

Amazon ECS マネージドインスタンスのキャパシティプロバイダー

AWS は、プロビジョニング、パッチ適用、スケーリング、ライフサイクル管理など、基盤となる Amazon EC2 インスタンスを完全に管理します。これにより、パフォーマンス、コスト効率、運用のシンプルさの最適なバランスが得られます。Amazon ECS マネージドインスタンスキャパ

シティプロバイダーは、ワークロードの要件に基づいてインスタンスの選択とスケーリングを自動的に最適化します。

Amazon ECS マネージドインスタンスには次のメリットがあります。

- インスタンスの自動プロビジョニングとスケーリング
- マネージドパッチ適用とセキュリティ更新
- インテリジェントなインスタンス選択によるコスト最適化
- 運用オーバーヘッドの削減

Fargate キャパシティプロバイダー

インフラストラクチャを管理せずに、タスクが使用するリソースに対してのみ料金を支払うサーバーレスコンピューティング。事前定義されたキャパシティプロバイダー (Fargate および Fargate Spot) をクラスターに関連付けるだけで利用できます。

Auto Scaling グループキャパシティープロバイダー

キャパシティとして Amazon EC2 インスタンスを使用する場合は、Auto Scaling グループを使用して Amazon EC2 インスタンスを管理します。Auto Scaling により、アプリケーションの負荷を処理するために適切な数の使用可能な Amazon EC2 インスタンスを確保できるようになります。基盤となるインフラストラクチャを完全に制御できます。

クラスターには、Amazon ECS マネージドインスタンス、AWS Fargate、Amazon EC2 インスタンス、または外部インスタンスでホストされているタスクを混在させることができます。タスクは、起動タイプまたはキャパシティプロバイダー戦略として、Amazon ECS マネージドインスタンス、Fargate、または EC2 のインフラストラクチャで実行できます。EC2 を起動タイプとして使用する場合、Amazon ECS は Amazon EC2 Auto Scaling グループのキャパシティーを追跡およびスケーリングしません。

クラスターには、Amazon ECS マネージドインスタンスキャパシティプロバイダー、Auto Scaling グループキャパシティプロバイダー、および Fargate キャパシティプロバイダーの組み合わせを含めることができます。キャパシティプロバイダー戦略には、Amazon ECS マネージドインスタンスキャパシティプロバイダー、Auto Scaling グループキャパシティプロバイダー、または Fargate キャパシティプロバイダーのみを含めることができます。

Amazon ECS マネージドインスタンスのクラスター

Amazon ECS キャパシティープロバイダーは、クラスター内のタスクに対するインフラストラクチャのスケーリングを管理できます。クラスターを作成したら、1 つ以上のキャパシティプロバイ

ダーと、クラスターのキャパシティプロバイダー戦略 (オプション) を作成します。キャパシティープロバイダー戦略は、クラスターの複数のキャパシティープロバイダー間にタスクを分散する方法を決定します。スタンドアロンタスクを実行するか、サービスを作成するときは、クラスターのデフォルトのキャパシティプロバイダー戦略か、クラスターのデフォルト戦略をオーバーライドするキャパシティプロバイダー戦略のいずれかを使用します。

Amazon ECS マネージドインスタンスには、次のキャパシティプロバイダーがあります。

タイプ	インスタンスの選択に使用される基準

デフォルト	次に示すタスク定義とサービスパラメータ要件を満たす最も費用対効果の高いインスタンス。
	• タスク定義
	  • operatingSystemFamily
	  • cpuArchitecture
	  • cpu
	  • メモリ
	• サービスの定義
	  • placementConstraints
	  • placementStrategy

カスタム	クラスターの作成時に指定した属性とタイプの要件を満たすインスタンス。属性の詳細については、「Amazon ECS コンテナインスタンス属性」を参照してください。インスタンスタイプの詳細については、「Amazon EC2 インスタンスタイプ」の「Amazon EC2 インスタンスタイプの仕様」を参照してください。

Amazon ECS はインスタンスを起動し、起動したインスタンスを Amazon ECS マネージドインスタンスキャパシティプロバイダーに関連付けます。カスタムキャパシティプロバイダーの場合、Amazon ECS はキャパシティプロバイダーも作成します。

Amazon ECS マネージドインスタンスのクラスターを作成する

クラスターを作成して、タスクとサービスを実行するインフラストラクチャを定義します。

Amazon ECS マネージドインスタンスのクラスターを作成すると、Amazon ECS はデフォルト設定でマネージドインスタンスキャパシティプロバイダーを自動的に作成します。このキャパシティプロバイダーは、ワークロード用に最もコストが最適化されたインスタンスタイプを選択します。特定のインスタンス属性またはタイプが必要な場合は、カスタムキャパシティプロバイダーを作成することもできます。

クラスターの作成プロセスをできるだけ簡単にするために、コンソールには多くの選択肢がデフォルトで用意されています。

- AWS Cloud Map に、クラスターと同じ名前のデフォルトの名前空間を作成します。名前空間を使用すると、クラスターで作成したサービスを、追加の設定なしで名前空間内の他のサービスに接続できます。

  詳細については、「Amazon ECS サービスを相互接続する」を参照してください。

以下のオプションを変更できます。

- クラスターに関連付けられたデフォルトの名前空間を変更します。

  名前空間を使用すると、クラスターで作成したサービスを、追加の設定なしで名前空間内の他のサービスに接続できます。デフォルトの名前空間は、クラスター名と同じです。詳細については、「Amazon ECS サービスを相互接続する」を参照してください。
- マネージドストレージに AWS KMS キーを割り当てます。キーの作成方法の詳細については、AWS Key Management Service ユーザーガイドの「KMS キーを作成する」を参照してください。
- クラスターを識別しやすいようにタグを追加します。

前提条件

これを開始する前に、「Amazon ECS を使用するようにセットアップする」 の手順を完了し、適切な IAM 許可を割り当てる必要があります。詳細については、「the section called “Amazon ECS クラスターの例”」を参照してください。

クラスターを作成するユーザーには、追加のアクセス許可 `iam:CreateServiceLinkedRole` が必要です。

`instanceRequirements` を指定しない場合、Amazon ECS はタスク定義の要件に基づいて、最もコストが最適化されたインスタンスタイプを自動的に選択します。特定のインスタンス属性またはタイプを使用するには、`instanceRequirements` でキャパシティプロバイダーを作成します。

インスタンスの選択方法を理解します。詳細については、「Amazon ECS マネージドインスタンスにおけるインスタンス選択のベストプラクティス」を参照してください。

Amazon ECS マネージドインスタンスに必要な IAM ロールがあること。これには、以下が含まれます。

- インフラストラクチャロール – Amazon ECS がユーザーに代わって AWS サービスを呼び出し、Amazon ECS マネージドインスタンスインフラストラクチャを管理できるようにします。

  詳細については、「Amazon ECS インフラストラクチャ IAM ロール」を参照してください。
- インスタンスプロファイル – マネージドインスタンスで実行されている Amazon ECS コンテナエージェントと Docker デーモンへのアクセス許可を提供します。

  詳細については、「Amazon ECS マネージドインスタンスのインスタンスプロファイル」を参照してください。

コンソール手順

新しいクラスターを作成するには (Amazon ECS コンソール)

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションバーから、使用するリージョンを選択します。
3. ナビゲーションペインで [Clusters] (クラスター) を選択します。
4. [Clusters] (クラスター) ページで、[Create Cluster] (クラスターの作成) を選択します。
5. [クラスター設定] で以下を設定します。
   - [クラスター名] に一意の名前を入力します。

     名前には、最大 255 文字 (大文字と小文字)、数字、およびハイフンを含めることができます。
   - (オプション) Service Connect に使用する名前空間をクラスター名と別のものにするには、[名前空間] に一意の名前を入力します。
6. [カスタムキャパシティプロバイダー] の場合は、次の操作を行います。

- [EC2 キャパシティの取得方法を選択] で、[Amazon ECS マネージドインスタンス] を選択します。
- [インスタンスプロファイル] で、インスタンスプロファイルのロールを選択します。
- インフラストラクチャロール については、インフラストラクチャロールを選択します。
- カスタムキャパシティプロバイダーを使用するには、[インスタンス選択] で [カスタムを使用]を選択します。次に、属性ごとに [属性値] を入力します。

7. (オプション) Container Insights を使用して モニタリング を展開し、次のいずれかのオプションを選択します。

   - オブザーバビリティが強化された推奨の Container Insights を使用するには、[オブザーバビリティが強化された Container Insights] を選択します。
   - Container Insights を使用するには、[Container Insights] を選択します。

8. (オプション) マネージドストレージのデータを暗号化します。[暗号化] の [マネージドストレージ] に、マネージドストレージデータの暗号化に使用する AWS KMS キーの ARN を入力します。
9. (オプション) クラスターを識別しやすくするには、[Tags] (タグ) を展開し、タグを設定します。

   [タグの追加] [タグの追加] を選択して、以下を実行します。

   - [キー] にはキー名を入力します。
   - [値] にキー値を入力します。

10. [作成] を選択します。

AWS CLI の手順

AWS CLI を使用して、Amazon ECS マネージドインスタンスのクラスターを作成できます。AWS CLI の最新バージョンを使用する。最新のバージョンにアップグレードする方法については、「AWS CLI の最新バージョンをインストールまたは更新」を参照してください。

> **Note**
>
> デュアルスタックサービスエンドポイントを使用することで、AWS AWS CLI、SDK、Amazon ECS API から、IPv4 と IPv6 の両方を経由して Amazon ECS とやり取りができます。詳細については、「the section called “デュアルスタックのエンドポイントの使用”」を参照してください。

新しいクラスターを作成するには (AWS CLI)

1. 以下のコマンドを使用して、一意の名前を付けたクラスターを作成します。

```
aws ecs create-cluster --cluster-name managed-instances-cluster
```

出力:

```
{
    "cluster": {
        "status": "ACTIVE",
        "defaultCapacityProviderStrategy": [],
        "statistics": [],
        "capacityProviders": [],
        "tags": [],
        "clusterName": "managed-instances-cluster",
        "settings": [
            {
                "name": "containerInsights",
                "value": "disabled"
            }
        ],
        "registeredContainerInstancesCount": 0,
        "pendingTasksCount": 0,
        "runningTasksCount": 0,
        "activeServicesCount": 0,
        "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/managed-instances-
cluster"
    }
}
```

2. (オプション) クラスターのオブザーバビリティが強化された Container Insights を有効にするには、次のコマンドを使用します。

```
aws ecs put-account-setting --name containerInsights --value enhanced
```

3. (オプション) クラスターにタグを追加するには、次のコマンドを使用します。

```
aws ecs tag-resource --resource-arn
 arn:aws:ecs:region:aws_account_id:cluster/managed-instances-cluster --tags
 key=Environment,value=Production
```

次のステップ

Amazon ECS マネージドインスタンスのタスク定義を作成します。詳細については、「コンソールを使用した Amazon ECS タスク定義の作成」を参照してください。

スタンドアロンタスクとして、またはサービスの一部としてアプリケーションを実行します。詳細については次を参照してください:

- Amazon ECS タスクとしてのアプリケーションの実行
- Amazon ECS のローリング更新デプロイの作成

Amazon ECS マネージドインスタンスを使用するようにクラスターを更新する

Amazon ECS マネージドインスタンスを使用するように既存のクラスターを更新できます。

Amazon ECS マネージドインスタンスをクラスターに追加すると、Amazon ECS はデフォルト設定でマネージドインスタンスキャパシティプロバイダーを自動的に作成します。このキャパシティプロバイダーは、ワークロードに最もコストが最適化された汎用インスタンスタイプを選択します。特定のインスタンス属性またはタイプが必要な場合は、カスタムキャパシティプロバイダーを作成することもできます。

前提条件

`instanceRequirements` を指定しない場合、Amazon ECS はタスク定義の要件に基づいて、最もコストが最適化されたインスタンスタイプを自動的に選択します。特定のインスタンス属性またはタイプを使用するには、`instanceRequirements` でキャパシティプロバイダーを作成します。

Amazon ECS マネージドインスタンスに必要な IAM ロールがあること。これには、以下が含まれます。

- インフラストラクチャロール – Amazon ECS がユーザーに代わって AWS サービスを呼び出し、Amazon ECS マネージドインスタンスインフラストラクチャを管理できるようにします。

  詳細については、「Amazon ECS インフラストラクチャ IAM ロール」を参照してください。
- インスタンスプロファイル – マネージドインスタンスで実行されている Amazon ECS コンテナエージェントと Docker デーモンへのアクセス許可を提供します。

詳細については、「Amazon ECS マネージドインスタンスのインスタンスプロファイル」を参照してください。

更新に関する考慮事項

Amazon ECS マネージドインスタンスのクラスターを更新する場合は、次の点を考慮してください。

- 実行中のタスク – クラスター設定を更新しても、現在実行中のタスクには影響しません。更新後に起動される新しいタスクに変更が適用されます。
- キャパシティプロバイダーの変更 – キャパシティプロバイダー設定を変更すると、既存のマネージドインスタンスは引き続き実行されますが、新しいインスタンスは更新された設定を使用します。
- 変更のモニタリング – Container Insights を有効または無効にすると、クラスター全体のメトリクス収集に影響します。

コンソール手順

クラスターを更新するには (Amazon ECS コンソール)

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションバーから、使用するリージョンを選択します。
3. ナビゲーションペインで [Clusters] (クラスター) を選択します。
4. [クラスター] ページで、更新するクラスターを選択します。
5. [クラスターを更新] を選択します。
6. (オプション) キャパシティプロバイダーの設定を変更するには、[カスタムキャパシティプロバイダー]で、必要に応じて以下を更新します。
   - [インスタンスプロファイル] で、必要に応じて別のインスタンスプロファイルロールを選択します。
   - [インフラストラクチャロール] で、必要に応じて別のインフラストラクチャロールを選択します。
   - カスタムキャパシティプロバイダーを使用するには、[インスタンスの選択] で [属性値] の設定を更新します。
7. [更新] を選択します。

AWS CLI の手順

AWS CLI を使用して、Amazon ECS マネージドインスタンスのクラスターを更新できます。AWS CLI の最新バージョンを使用する。最新のバージョンにアップグレードする方法については、「AWS CLI の最新バージョンをインストールまたは更新」を参照してください。

> **Note**
>
> デュアルスタックサービスエンドポイントを使用することで、AWS AWS CLI、SDK、Amazon ECS API から、IPv4 と IPv6 の両方を経由して Amazon ECS とやり取りができます。詳細については、「the section called “デュアルスタックのエンドポイントの使用”」を参照してください。

クラスターを更新するには (AWS CLI)

1. 次のキャパシティプロバイダーを作成します。次のコマンドを実行してください:

   user-input を独自の値に置き換えます。

   ```
   aws ecs create-capacity-provider \
       --name my-managed-instances-provider \
       --managed-instances-provider \
       --instance-profile arn:aws:iam::123456789012:instance-profile/
   ecsInstanceProfile \
       --infrastructure-role-arn arn:aws:iam::123456789012:role/ecsInfrastructureRole
    \
       --instance-requirements '{
           "vCpuCount": {"min": 2, "max": 8},
           "memoryMiB": {"min": 4096, "max": 16384}
       }
   ```

2. 次のコマンドを使用して、クラスターにキャパシティプロバイダーを追加します。

   user-input を独自の値に置き換えます。

   ```
   aws ecs put-cluster-capacity-providers --cluster managed-instances-cluster --
   capacity-providers my-managed-instances-provider --default-capacity-provider-
   strategy capacityProvider=my-managed-instances-provider,weight=1
   ```

Amazon ECS マネージドインスタンスのキャパシティプロバイダー

Amazon ECS マネージドインスタンスキャパシティプロバイダーは、AWS で運用上およびセキュリティ上の責任を管理しながら、幅広い AWS の機能と Amazon EC2 サービスにアクセスできるコンテナコンピューティングモデルを提供します。AWS は、ソフトウェアと OS のパッチ適用、インスタンスのスケーリング、メンテナンスを処理し、すべての AWS 機能と統合へのアクセスを維持しながら、運用における Fargate のメリットを提供します。

Amazon ECS は、Amazon ECS マネージドインスタンスの起動テンプレートを作成します。このテンプレートは、タスクのインスタンスプロファイル、ネットワークとストレージの設定、容量オプション、柔軟なインスタンスタイプ選択のためのインスタンス要件など、Amazon ECS が Amazon ECS マネージドインスタンスを起動する条件を定義します。

カスタムキャパシティプロバイダーを使用する場面

ワークロードで以下が必要な場合は、カスタムキャパシティプロバイダーの使用を検討します。

- 特定のコンピューティング要件: アクセラレーテッドコンピューティング、特定の CPU 命令セット、ネットワークの高パフォーマンス、標準の Fargate オプションでは利用できない大きなメモリ設定を必要とするアプリケーションなど。
- GPU ワークロード: NVIDIA または AMD GPU へのアクセスを必要とする機械学習推論、リアルタイムイメージレンダリング、ビデオエンコーディング、その他の GPU アクセラレーションアプリケーションなど。
- キャパシティ予約: 予測可能なキャパシティの可用性を必要とするミッションクリティカルなワークロード。
- 高度なオブザーバビリティ: eBPF ベースのモニタリングソリューションやネットワーク分析ツールなど、基盤となる OS への特権アクセスを必要とするセキュリティおよびモニタリングツール。
- コストの最適化: マルチタスク配置、共有インフラストラクチャコンポーネント、大型のインスタンスタイプの使用率を最大化する必要があるワークロード。

モニタリングオプション

Amazon ECS マネージドインスタンスは、コンテナ化されたワークロードのパフォーマンス、ヘルス、リソース使用率の追跡に役立つ包括的なモニタリング機能を提供します。運用要件に基づき、さまざまなモニタリングレベルから選択できます。

- 基本モニタリング – ほとんどのメトリクスでは 5 分間隔、ステータスチェックでは 1 分間隔で必須メトリクスを提供します。デフォルトで有効になっています。追加料金はありません。

- 詳細モニタリング – すべてのメトリクスを 1 分間隔で利用できるため、可視性が向上し、運用上の問題の検出と対応が高速化します。詳細については、「Amazon ECS マネージドインスタンスの詳細なモニタリング」を参照してください。

どちらのモニタリングオプションも CloudWatch とシームレスに統合され、最適なアプリケーションのパフォーマンスと可用性の維持に役立つダッシュボード、アラーム、自動応答機能を提供します。

キャパシティープロバイダーに関する考慮事項

クラスターには、Amazon ECS マネージドインスタンスキャパシティプロバイダー、Auto Scaling グループキャパシティプロバイダー、および Fargate キャパシティプロバイダーの組み合わせを含めることができます。キャパシティプロバイダー戦略には、Amazon ECS マネージドインスタンスキャパシティプロバイダー、Auto Scaling グループキャパシティプロバイダー、または Fargate キャパシティプロバイダーのみを含めることができます。

タグ伝達

Amazon ECS マネージドインスタンスキャパシティプロバイダーは、タグの伝播をサポートしています。タグの伝播では、キャパシティプロバイダーによって管理されるすべてのリソース (マネージドインスタンス、Amazon ECS コンテナインスタンス、起動テンプレート、ボリューム、Elastic Network Interfaces) は、キャパシティプロバイダーレベルで指定されたものと同じタグでタグ付けされます。キャパシティプロバイダーの作成時にタグを指定し、`propagateTags` パラメータを `CAPACITY_PROVIDER` として指定することでタグの伝播を有効にできます。

Amazon ECS マネージドインスタンスでのタグ付けの詳細については、「Amazon ECS マネージドインスタンスのタグ」を参照してください。

Amazon ECS マネージドインスタンスのキャパシティプロバイダー更新のベストプラクティス

最高レベルの安全性とロールバックのサポートを利用するには、キャパシティプロバイダーをイミュータブルリソースとして扱うことをお勧めします。キャパシティプロバイダーの設定を更新する必要がある場合は、次の推奨ワークフローに従います。

1. 既存の設定を変更する代わりに、更新した設定の新規キャパシティプロバイダーを作成します。
2. 新規キャパシティプロバイダーを使用するように各サービスを更新し、デプロイを完了できるようにします。
3. 新しい設定が正常に機能することを確認したら、古いキャパシティプロバイダーを削除します。

このアプローチにはいくつかの利点があります。

- ロールアウトを管理可能 – サービスを一度に 1 つずつ更新し、影響をモニタリングできます。
- ロールバックが簡単 – 問題が発生した場合は、すぐにサービスを元に戻し、以前のキャパシティプロバイダーを使用することができます。
- 影響が及ぶ範囲を縮小 – 新しい設定で問題が発生した場合でも、直ちにすべてのワークロードには影響しません。

> **Note**
>
> CloudFormation を使用している場合は、スタックの変更をロールバックできるように、後のデプロイまで古いキャパシティプロバイダーを維持することを検討してください。

このアプローチでは、設置したキャパシティプロバイダーは更新できますが、制御されない影響の及ぶ範囲が大きくなります。インプレース更新では、今後プロビジョニングされるすべての新しいキャパシティに新しい設定が適用されますが、サービスのデプロイはトリガーされません。したがって、時間が経過してサービスのスケーリングが必要になってから設定の問題が検出される可能性があります。

Amazon ECS マネージドインスタンスのキャパシティプロバイダーを作成する

Amazon ECS マネージドインスタンスは、キャパシティプロバイダを使用してワークロードのコンピューティングキャパシティを管理します。`instanceRequirements` を指定せずにキャパシティプロバイダーを作成すると、Amazon ECS は、最もコストが最適化された汎用インスタンスタイプを自動的に選択します。`instanceRequirements` でキャパシティプロバイダーを作成して、インスタンスタイプ、CPU 製造元、アクセラレータータイプ、その他の要件などのインスタンス属性を指定することもできます。

カスタムキャパシティプロバイダーは、属性ベースのインスタンスタイプの選択を使用しており、一連の属性としてインスタンス要件を表現できます。これらの要件は、一致するすべての Amazon EC2 インスタンスタイプに自動的に変換されるため、インスタンスタイプ設定の作成とメンテナンスが簡素化されます。インスタンスの要件と属性ベースの選択の詳細については、「Amazon EC2 ユーザーガイド」の「Amazon EC2 Fleet の属性ベースのインスタンスタイプの選択」ドキュメントを参照してください。

前提条件

作業を開始する前に、以下を完了していることを確認してください。

- 使用するモニタリングのタイプを決定する。詳細については、「the section called “Amazon ECS マネージドインスタンスの詳細なモニタリング”」を参照してください。
- 既存のクラスターがあるか、クラスターを作成する予定である。詳細については、「Amazon ECS マネージドインスタンスのクラスターを作成する」を参照してください。
- Amazon ECS マネージドインスタンスに必要な IAM ロールがあること。これには、以下が含まれます。
  - インフラストラクチャロール – Amazon ECS がユーザーに代わって AWS サービスを呼び出し、Amazon ECS マネージドインスタンスインフラストラクチャを管理できるようにします。

    詳細については、「Amazon ECS インフラストラクチャ IAM ロール」を参照してください。
  - インスタンスプロファイル – マネージドインスタンスで実行されている Amazon ECS コンテナエージェントと Docker デーモンへのアクセス許可を提供します。

    詳細については、「Amazon ECS マネージドインスタンスのインスタンスプロファイル」を参照してください。

インスタンスの選択方法を理解します。詳細については、「Amazon ECS マネージドインスタンスにおけるインスタンス選択のベストプラクティス」を参照してください。

コンソール手順

Amazon ECS マネージドインスタンスのキャパシティプロバイダーを作成するには (Amazon ECS コンソール)

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションバーから、使用するリージョンを選択します。
3. ナビゲーションペインで [Clusters] (クラスター) を選択します。
4. [クラスター] ページでクラスターの名前を選択します。
5. クラスターのページで、[インフラストラクチャ] タブを選択します。
6. [キャパシティプロバイダー] セクションで、[キャパシティプロバイダーを作成] を選択します。
7. [キャパシティプロバイダー設定]で、以下を設定します。
   - [キャパシティプロバイダー名] に、一意のキャパシティプロバイダー名を入力します。

- [キャパシティプロバイダータイプ] で、[Amazon ECS マネージドインスタンス] を選択します。

8. [インスタンス設定] で以下を設定します。

   - [インスタンスプロファイル] で、Amazon ECS マネージドインスタンス用に作成したインスタンスプロファイルロールを選択します。
   - [インフラストラクチャロール] で、Amazon ECS マネージドインスタンス用に作成したインフラストラクチャロールを選択します。

9. [インスタンス要件]で、インスタンスの属性を指定します。次の項目を任意で組み合わせて設定できます。

   - [vCPU カウント] – vCPU の数を指定します (4 など。8-16 のように範囲指定も可)。
   - [メモリ (MiB)] – メモリの量を MiB 単位で指定します (8192 など。16384-32768 のように範囲指定も可)。
   - [インスタンスタイプ] – 特定のインスタンスタイプを指定します (`m5.large,m5.xlarge,c5.large` など)。
   - [CPU メーカー] – `intel`、`amd`、`amazon-web-services` から選択します。
   - [アクセラレーターのタイプ] – `gpu`、`fpga`、`inference` などのアクセラレータータイプを指定します。
   - [アクセラレーター数] – アクセラレーターの数を指定します (1 など。2-4 のように範囲指定も可)。

10. [詳細設定] で、次のいずれかのモニタリングオプションを選択します。

    - CloudWatch でステータスチェックメトリクスを送信する場合は、[基本] を選択します。
    - CloudWatch ですべてのメトリクスを送信する場合は、[詳細] を選択します。

11. (オプション) キャパシティプロバイダーを識別しやすくするには、[タグ] を展開し、タグを設定します。

    キャパシティプロバイダーからマネージドリソース (キャパシティプロバイダーから起動されたインスタンスなど) へのタグの伝播を有効にするには、[タグの伝播元] で [キャパシティプロバイダー] を選択します。

    [タグの追加] [タグの追加] を選択して、以下を実行します。

    - [キー] にはキー名を入力します。
    - [値] にキー値を入力します。

12. [作成] を選択します。

AWS CLI の手順

AWS CLI を使用して、Amazon ECS マネージドインスタンスのキャパシティプロバイダーを作成できます。AWS CLI の最新バージョンを使用する。最新のバージョンにアップグレードする方法については、「AWS CLI の最新バージョンをインストールまたは更新」を参照してください。

Amazon ECS マネージドインスタンスのキャパシティプロバイダーを作成するには (AWS CLI)

1. 次のコマンドを実行してください:

```
aws ecs create-capacity-provider --cli-input-json file://capacity-provider-
definition.json
```

以下のように、capacity-provider-definition.json を使用して、基本的なインスタンス要件、インスタンスストレージサイズを指定し、タグ伝播を有効にすることができます。

```
{
    "name": "my-managed-instances-provider",
    "cluster": "my-cluster",
    "tags": [
        {
            "key": "version",
            "value": "test"
        }
    ],
    "managedInstancesProvider": {
        "infrastructureRoleArn": "arn:aws:iam::123456789012:role/
ecsInfrastructureRole",
        "instanceLaunchTemplate": {
            "ec2InstanceProfileArn": "arn:aws:iam::123456789012:instance-profile/
ecsInstanceRole",
            "instanceRequirements": {
                "vCpuCount": {
                    "min": 4,
                    "max": 8
                },
                "memoryMiB": {
                    "min": 8192,
                    "max": 16384
```

```
                }
            },
            "networkConfiguration": {
                "subnets": [
                    "subnet-abcdef01234567",
                    "subnet-bcdefa98765432"
                ],
                "securityGroups": [
                    "sg-0123456789abcdef"
                ]
            },
            "storageConfiguration": {
                "storageSizeGiB": 100
            },
            "monitoring": "basic"
        },
        "propagateTags": "CAPACITY_PROVIDER"
    }
}
```

2. キャパシティプロバイダーが正常に作成されたことを確認します。

```
aws ecs describe-capacity-providers \
    --capacity-providers my-managed-instances-provider
```

次のステップ

キャパシティプロバイダーの作成後は、サービスの作成時またはタスクの実行時に使用できます。

- サービスでキャパシティプロバイダーを使用する場合は、「Amazon ECS のローリング更新デプロイの作成」を参照してください。
- スタンドアロンタスクでキャパシティプロバイダーを使用する場合は、「Amazon ECS タスクとしてのアプリケーションの実行」を参照してください。

Amazon ECS マネージドインスタンスのモニタリングを更新する

Amazon ECS マネージドインスタンスキャパシティプロバイダーのモニタリングオプションを変更して、基本モニタリングと詳細モニタリングを変更できます。この機能で、キャパシティプロバイダーを再作成することなく、収集されたモニタリングデータのレベルを調整できます。

モニタリングオプションの詳細については、「Amazon ECS マネージドインスタンスのモニタリング」を参照してください。

コンソール手順

Amazon ECS マネージドインスタンスのモニタリングを更新するには (Amazon ECS コンソール)

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションバーから、使用するリージョンを選択します。
3. ナビゲーションペインで [Clusters] (クラスター) を選択します。
4. [クラスター] ページでクラスターの名前を選択します。
5. クラスターのページで、[インフラストラクチャ] タブを選択します。
6. [詳細設定] で、次のいずれかのモニタリングオプションを選択します。
   - CloudWatch でステータスチェックメトリクスを送信する場合は、[基本] を選択します。
   - CloudWatch ですべてのメトリクスを送信する場合は、[詳細] を選択します。
7. [更新] を選択します。

既存の Amazon ECS マネージドインスタンスキャパシティプロバイダーに関連付けられているタグを更新するには、次の手順に従います。

1. ナビゲーションペインで [Clusters] (クラスター) を選択します。
2. [クラスター] ページで、[インフラストラクチャ] を選択します。
3. [インフラストラクチャページ] で、作成したキャパシティプロバイダーを選択します。
4. [キャパシティプロバイダー] ページで、[タグ] を選択します。
5. [タグ] の項目で、[タグの管理] を選択します。
6. タグを追加するには、追加するタグのキーと値を指定し、[保存] を選択します。一度に複数のタグを追加するには、追加するタグごとに [タグを追加] を選択します。最大 50 個のタグを追加できます。

   タグを削除するには[削除] を選択してください。

> **Note**
>
> タグ伝播が有効になっている場合、キャパシティプロバイダーの作成後に追加または削除されたタグは、それ以前にキャパシティプロバイダーで作成されたリソースには適用されません。

AWS CLI の手順

AWS CLI を使用して、Amazon ECS マネージドインスタンスのキャパシティプロバイダーを更新できます。AWS CLI の最新バージョンを使用する。最新のバージョンにアップグレードする方法については、「AWS CLI の最新バージョンをインストールまたは更新」を参照してください。

Amazon ECS マネージドインスタンスのモニタリングを更新するには (AWS CLI)

1. 詳細モニタリングを有効にするには、次のコマンドを使用します。

```
aws ecs update-capacity-provider \
    --name my-managed-instances-provider \
    --managed-instances-provider '{
        "instanceLaunchTemplateUpdate": {
            "monitoring": "DETAILED"
        }
    }'
```

2. 基本モニタリングを有効にするには、次のコマンドを使用します。

```
aws ecs update-capacity-provider \
    --name my-managed-instances-provider \
    --managed-instances-provider '{
        "instanceLaunchTemplateUpdate": {
            "monitoring": "BASIC"
        }
    }'
```

Amazon ECS マネージドインスタンスキャパシティプロバイダーを削除する

Amazon ECS マネージドインスタンスキャパシティプロバイダーは、使い終わったら削除できます。グループが削除されると、Amazon ECS マネージドインスタンスキャパシティプロバイダー

は INACTIVE 状態に移行します。INACTIVE ステータスのキャパシティープロバイダーは、一定期間アカウント内で検出可能になる場合があります。ただし、この動作は今後変更される予定であり、INACTIVE のキャパシティープロバイダーを永続的に使用するのは避けてください。Amazon ECS マネージドインスタンスキャパシティプロバイダーを削除する前に、すべてのサービスからキャパシティプロバイダー戦略を削除する必要があります。サービスのキャパシティプロバイダー戦略からキャパシティプロバイダーを削除するには、UpdateService API または Amazon ECS コンソールのサービス更新ワークフローを使用できます。[新しいデプロイの強制] オプションを使用して、キャパシティプロバイダーによって提供される Amazon ECS マネージドインスタンスキャパシティを使用する任意のタスクを、残りのキャパシティプロバイダーのキャパシティを使用するように移行することができます。

クラスターのキャパシティプロバイダーを削除するには (Amazon ECS コンソール)

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。
3. [Clusters] (クラスター) ページで、クラスターを選択します。
4. [クラスター: ***##***] ページの [インフラストラクチャ] で、Amazon ECS マネージドインスタンスキャパシティプロバイダーを選択し、[削除] を選択します。
5. 確認ボックスに、「delete ***Amazon ECS ##########################***」と入力します。
6. [削除] を選択します。

Amazon ECS マネージドインスタンスで実行されている Amazon ECS ワークロードを安全に停止する

Amazon ECS タスクスケールイン保護を使用すると、サービスのスケーリングまたはデプロイからのスケールインイベントによってタスクが終了されるのを防ぐことができます。

特定のアプリケーションでは、使用率が低いときやサービスのデプロイ中に、ミッションクリティカルなタスクがスケールインイベントによって終了されるのを防ぐメカニズムが必要です。例えば、次のようになります。

- ビデオトランスコーディングジョブなどのキュー処理型の非同期アプリケーションでは、サービスの累積使用率が低い場合でも、一部のタスクは何時間も実行する必要があります。
- ゲームアプリケーションによっては、ゲームサーバーはタスクとして実行されますが、サーバー再起動時の起動待ち時間を短縮するため、すべてのユーザーがログアウトした場合でも実行を続ける必要があります。

- 新しいコードバージョンをデプロイする場合、再処理にはコストがかかるため、タスクは実行し続ける必要があります。

サービスに属するタスクがスケールインイベントで終了されることを防ぐには、ProtectionEnabled 属性を true に設定します。ProtectionEnabled を true に設定すると、タスクはデフォルトで 2 時間保護されます。その後、保護期間は、ExpiresInMinutes 属性を使用してカスタマイズできます。タスクの保護期間は最短で 1 分間、最長で 2,880 分 (48 時間) です。AWS CLI を使用している場合は、--protection-enabled オプションを指定できます。

タスクが必要な作業を完了した後は、ProtectionEnabled 属性を false に設定して、以降のスケールインイベントによってタスクが終了されるようにできます。AWS CLI を使用している場合は、--no-protection-enabled オプションを指定できます。

タスクスケールイン保護メカニズム

Amazon ECS コンテナエージェントエンドポイントまたは Amazon ECS API を使用して、タスクスケールイン保護を設定および取得できます。

- Amazon ECS コンテナエージェントエンドポイント

  保護の必要性を自己判断できるタスクには、Amazon ECS コンテナエージェントエンドポイントを使用することをお勧めします。このアプローチは、キューベースのワークロードまたはジョブ処理ワークロードに使用します。

  コンテナが、例えば SQS メッセージを使用するなどして処理を開始すると、コンテナ内からタスクスケールイン保護のエンドポイントパス「$ECS_AGENT_URI/task-protection/v1/state」を使用して ProtectionEnabled 属性を設定できます。スケールインイベントの際、Amazon ECS はこのタスクを終了しません。タスクの処理が完了した後は、同じエンドポイントを使用して ProtectionEnabled 属性をクリアし、以降のスケールインイベントでタスクが終了されるようにできます。

  Amazon ECS コンテナエージェントエンドポイントの詳細については、「Amazon ECS タスクスケールイン保護のエンドポイント」を参照してください。

- Amazon ECS API

  アプリケーションにアクティブなタスクの状態を追跡するコンポーネントがある場合は、Amazon ECS API を使用してタスクスケールイン保護を設定および取得できます。UpdateTaskProtection を使用して、1 つ以上のタスクを保護対象としてマークします。GetTaskProtection を使用して保護ステータスを取得します。

このアプローチの例としては、アプリケーションがゲームサーバーセッションを Amazon ECS タスクとしてホストしている場合が挙げられます。ユーザーがサーバー上のセッション (タスク) にログインすると、そのタスクを保護対象としてマークできます。ユーザーがログアウトした後は、サーバーのアイドル状態を維持することの要件に応じて、このタスク限定でクリアすることも、アクティブなセッションがなくなった同様のタスクについて定期的に保護をクリアすることもできます。

詳細については、「Amazon Elastic Container Service API リファレンス」の「UpdateTaskProtection」と「GetTaskProtection」を参照してください。

両方のアプローチを組み合わせることができます。例えば、Amazon ECS エージェントエンドポイントを使用してコンテナ内からタスク保護を設定し、Amazon ECS API を使用して外部コントローラーサービスからタスク保護を削除するなどです。

考慮事項

タスクスケールイン保護を使用する前に、次の点を考慮してください。

- タスクのスケールイン保護は、サービスからデプロイされたタスクでのみサポートされます。
- タスクのスケールイン保護は、Amazon ECS マネージドインスタンスで実行されているサービスからデプロイされたタスクでサポートされます。
- Amazon ECS エージェントには再試行メカニズムが組み込まれており、そのインターフェイスはよりシンプルであるため、Amazon ECS コンテナエージェントエンドポイントを使用することをお勧めします。
- 既に保護がオンになっているタスクのために `UpdateTaskProtection` を呼び出すことで、タスクのスケールイン保護の有効期間をリセットできます。
- タスクが必要な作業を完了するのに必要な時間を判断し、それに応じて `expiresInMinutes` プロパティを設定してください。保護の有効期限を必要以上に長く設定すると、コストが発生すると共に、新しいタスクのデプロイが遅れることになります。
- デプロイに関する考慮事項:
  - サービスがローリングアップデートを使用する場合、新しいタスクは作成できますが、古いバージョンを実行しているタスクは、`protectionEnabled` がクリアされるか、または失効するまで終了しません。デプロイ設定の `maximumPercentage` パラメータは、古いタスクが保護されている場合でも新しいタスクを作成できるように値を調整できます。

- CloudFormation を使用する場合、更新スタックのタイムアウトは 3 時間です。そのため、タスク保護を 3 時間以上設定すると、CloudFormation のデプロイで障害が発生してロールバックが発生する可能性があります。

  古いタスクが保護されている間、CloudFormation スタックは UPDATE_IN_PROGRESS を表示します。タスクのスケールイン保護が削除されるか、または 3 時間のウィンドウ内に失効する場合、デプロイは成功し、UPDATE_COMPLETE ステータスに移行します。デプロイが UPDATE_IN_PROGRESS のまま 3 時間以上滞っていると、デプロイは失敗して UPDATE_FAILED 状態が表示され、古いタスクセットにロールバックされてしまいます。
- 保護されたタスクが原因でデプロイ (ローリングまたはブルー/グリーン) が定常状態に到達できない場合、Amazon ECS はサービスイベントを送信し、これによりユーザーは是正措置を講じることができます。タスクの保護ステータスを更新しようとする際に DEPLOYMENT_BLOCKED エラーメッセージが表示される場合、サービスの要求されるタスク数よりも多くの保護されたタスクがあることを意味します。この問題を解決するには、次のいずれかの操作を実行します。
  - 現在のタスク保護の有効期限が切れるのを待ちます。次に、タスク保護を設定します。
  - どのタスクを停止できるかを決定します。その後、これらのタスクについて protectionEnabled オプションを false に設定して UpdateTaskProtection を使用します。
  - サービスの必要なタスク数を増やして、保護されているタスクの数よりも多くします。

タスクスケールイン保護に必要な IAM アクセス許可

タスクには、以下のアクセス許可を持つ Amazon ECS タスクロールが必要です。

- ecs:GetTaskProtection: Amazon ECS コンテナエージェントが GetTaskProtection を呼び出すことを許可します。
- ecs:UpdateTaskProtection: Amazon ECS コンテナエージェントが UpdateTaskProtection を呼び出すことを許可します。

Amazon ECS タスクスケールイン保護のエンドポイント

Amazon ECS コンテナエージェントは、ECS_AGENT_URI 環境変数を Amazon ECS タスクのコンテナに自動的に挿入して、コンテナエージェント API エンドポイントとやり取りする方法を提供します。

保護の必要性を自己判断できるタスクには、Amazon ECS コンテナエージェントエンドポイントを使用することをお勧めします。

コンテナが処理を開始すると、コンテナ内からタスクスケールイン保護のエンドポイントパス「$ECS_AGENT_URI/task-protection/v1/state」を使用して protectionEnabled 属性を設定できます。

コンテナ内からこの URI への PUT リクエストを使用すると、タスクのスケールイン保護が設定されます。この URI への GET リクエストは、タスクの現在の保護ステータスを返します。

タスクスケールイン保護のリクエストパラメータ

次のリクエストパラメータで、${ECS_AGENT_URI}/task-protection/v1/state エンドポイントを使用してタスクスケールインプロテクションを設定できます。

ProtectionEnabled

true を指定して、タスクを保護できるようにマークします。false を指定して保護を解除し、タスクを終了できるようにします。

タイプ: ブール値

必須: はい

ExpiresInMinutes

タスクが保護されている時間 (分)。最小 1 分から最大 2,880 分 (48 時間) まで指定できます。この期間中、自動スケーリングサービスまたはデプロイからのスケールインイベントによってタスクが終了することはありません。この時間が経過すると、protectionEnabled パラメータは false に設定されます。

時間を指定しない場合、タスクは自動的に 120 分 (2 時間) 保護されます。

タイプ: 整数

必須: いいえ

次の例は、異なる継続時間を持つタスク保護を設定する方法を示します。

デフォルトの期間を使用してタスクを保護する方法の例

この例は、デフォルトの期間が 2 時間に設定されているタスクを保護する方法を示しています。

```
curl --request PUT --header 'Content-Type: application/json' ${ECS_AGENT_URI}/task-
protection/v1/state --data '{"ProtectionEnabled":true}'
```

タスクを 60 分間保護する方法の例

この例は、expiresInMinutes パラメータを使用してタスクを 60 分間保護する方法を示しています。

```
curl --request PUT --header 'Content-Type: application/json' ${ECS_AGENT_URI}/task-
protection/v1/state --data '{"ProtectionEnabled":true,"ExpiresInMinutes":60}'
```

タスクを 24 時間保護する方法の例

この例は、expiresInMinutes パラメータを使用してタスクを 24 時間保護する方法を示しています。

```
curl --request PUT --header 'Content-Type: application/json' ${ECS_AGENT_URI}/task-
protection/v1/state --data '{"ProtectionEnabled":true,"ExpiresInMinutes":1440}'
```

PUT リクエストは次のレスポンスを返します。

```
{
  "protection": {
    "ExpirationDate": "2023-12-20T21:57:44.837Z",
    "ProtectionEnabled": true,
    "TaskArn": "arn:aws:ecs:us-west-2:111122223333:task/1234567890abcdef0"
  }
}
```

タスクスケールイン保護のレスポンスパラメータ

次の情報が、JSON レスポンスタスクのスケールインプロテクションエンドポイント ${ECS_AGENT_URI}/task-protection/v1/state から返されます。

ExpirationDate

タスクの保護が期限切れになる工ポックタイム。タスクが保護されていない場合、この値は NULL です。

`ProtectionEnabled`

タスクのプロテクションステータス。タスクのスケールインプロテクションが有効になっている場合、値は `true` です。それ以外の場合は、`false` です。

`TaskArn`

コンテナが属しているタスクの完全な Amazon リソースネーム (ARN)。

次の例は、保護されたタスクについて返される詳細を示しています。

```
curl --request GET ${ECS_AGENT_URI}/task-protection/v1/state
```

```
{
    "protection":{
        "ExpirationDate":"2023-12-20T21:57:44Z",
        "ProtectionEnabled":true,
        "TaskArn":"arn:aws:ecs:us-west-2:111122223333:task/1234567890abcdef0"
    }
}
```

障害が発生すると、次の情報が返されます。

`Arn`

タスクの完全な Amazon リソースネーム (ARN)。

`Detail`

障害に関する詳細。

`Reason`

失敗の理由。

次の例は、保護されていないタスクについて返される詳細を示しています。

```
{
    "failure":{
        "Arn":"arn:aws:ecs:us-west-2:111122223333:task/1234567890abcdef0",
        "Detail":null,
```

```
        "Reason":"TASK_NOT_VALID"
    }
}
```

例外が発生すると、次の情報が返されます。

`requestID`

例外が発生する Amazon ECS API 呼び出しの AWS リクエスト ID。

`Arn`

タスクまたはサービスの完全な Amazon リソースネーム (ARN)。

`Code`

エラーコードです。

`Message`

エラーメッセージです。

> **Note**
>
> `RequestError` または `RequestTimeout` エラーが表示される場合は、ネットワークに問題がある可能性があります。Amazon ECS の VPC エンドポイントを使用してみてください。

次の例は、エラーが発生したときに返される詳細を示したものです。

```
{
    "requestID":"12345-abc-6789-0123-abc",
    "error":{
        "Arn":"arn:aws:ecs:us-west-2:555555555555:task/my-cluster-
name/1234567890abcdef0",
        "Code":"AccessDeniedException",
        "Message":"User: arn:aws:sts::444455556666:assumed-role/my-ecs-task-
role/1234567890abcdef0 is not authorized to perform: ecs:GetTaskProtection on resource:
 arn:aws:ecs:us-west-2:555555555555:task/test/1234567890abcdef0 because no identity-
based policy allows the ecs:GetTaskProtection action"
    }
}
```

ネットワークの問題や Amazon ECS コントロールプレーンがダウンしているなどの理由で Amazon ECS エージェントが Amazon ECS エンドポイントから応答を得られない場合、次のエラーが表示されます。

```
{
  "error": {
    "Arn": "arn:aws:ecs:us-west-2:555555555555:task/my-cluster-name/1234567890abcdef0",
    "Code": "RequestCanceled",
    "Message": "Timed out calling Amazon ECS Task Protection API"
  }
}
```

Amazon ECS エージェントが Amazon ECS からスロットリング例外を受け取ると、次のエラーが表示されます。

```
{
  "requestID": "12345-abc-6789-0123-abc",
  "error": {
    "Arn": "arn:aws:ecs:us-west-2:555555555555:task/my-cluster-name/1234567890abcdef0",
    "Code": "ThrottlingException",
    "Message": "Rate exceeded"
  }
}
```

Fargate 用の Amazon ECS クラスター

AWS Fargate キャパシティプロバイダーで Amazon ECS を使用すると、Amazon ECS タスクで Fargate と Fargate Spot キャパシティの両方を使用できます。

Fargate Spot を使用すると、割り込み許容のある Amazon ECS タスクを、Fargate 料金と比較して割引料金で実行できます。Fargate Spot は、予備のコンピュートキャパシティーでタスクを実行します。AWS がキャパシティを戻す必要がある場合、タスクは中断され、2 分間の警告が表示されます。

Fargate と Fargate Spot キャパシティープロバイダーを使用するタスクが停止した場合、タスク状態の変更イベントが Amazon EventBridge に対し送信されます。停止した理由は原因を説明しています。詳細については、「Amazon ECS タスク状態変更イベント」を参照してください。

クラスターには、Fargate キャパシティプロバイダーと Auto Scaling グループキャパシティプロバイダーを混在させることができます。ただし、キャパシティプロバイダー戦略に含めることができるの

は Fargate または Auto Scaling グループキャパシティプロバイダーのみで、両方を含めることはできません。詳細については、「Auto Scaling グループキャパシティープロバイダー」を参照してください。

キャパシティプロバイダーを使用する場合は、次の点を考慮します。

- キャパシティープロバイダーは、キャパシティープロバイダー戦略に関連付ける前に、クラスターに関連付ける必要があります。
- キャパシティープロバイダー戦略には、最大 20 のキャパシティープロバイダーを指定できます。
- Auto Scaling グループキャパシティプロバイダーを使用するサービスは、Fargate キャパシティプロバイダーを使用するように更新することはできません。逆の場合も同様です。
- キャパシティプロバイダー戦略では、コンソールでキャパシティプロバイダーに `weight` 値が指定されていない場合、1 のデフォルト値が使用されます。API または AWS CLI を使用する場合は、`0` のデフォルト値が使用されます。
- キャパシティプロバイダー戦略内で複数のキャパシティプロバイダーを指定する場合、少なくとも 1 つのキャパシティプロバイダーのウェイト値が 0 より大きい必要があります。ウェイトが 0 のキャパシティープロバイダーはタスクの配置に使用されません。戦略に複数のキャパシティプロバイダーを指定し、すべて同じウェイトを 0 にした場合、キャパシティプロバイダー戦略を使用する `RunTask` または `CreateService` のアクションは失敗します。
- キャパシティプロバイダー戦略では、1 つのキャパシティプロバイダーのみが定義されたベース値を持つことができます。ベース値を指定しない場合は、デフォルト値の 0 が使用されます。
- クラスターには、Auto Scaling グループキャパシティプロバイダーと Fargate キャパシティプロバイダーの両方を混在させることができます。ただし、キャパシティプロバイダー戦略に含めることができるのは Auto Scaling グループまたは Fargate キャパシティプロバイダーのいずれかのみとなっており、両方を含めることはできません。
- クラスターには、両方のキャパシティプロバイダーを使用するサービスとスタンドアロンタスクを混在させることができます。サービスは、起動タイプではなくキャパシティプロバイダー戦略を使用するように更新できます。ただし、その場合は強制的に新しいデプロイを行う必要があります。

Fargate Spot 終了通知

需要が非常に多い時期は、Fargate Spot のキャパシティーが利用できない場合があります。これにより、Fargate Spot のタスクが遅れる可能性があります。これが発生すると、Amazon ECS サービスは必要なキャパシティーが使用可能になるまでタスクの起動を再試行します。Fargate が Spot キャパシティーをオンデマンドキャパシティに置き換えることはありません。

スポットの中断により Fargate Spot キャパシティーを使用するタスクが停止すると、タスクが停止する前に 2 分間の警告が送信されます。警告は、タスク状態変更イベントとして Amazon EventBridge に送信され、実行中のタスクに SIGTERM シグナルとして送信されます。サービスの一部として Fargate Spot を使用する場合、このシナリオでは、サービススケジューラは中断信号を受信し、利用可能なキャパシティがあれば Fargate Spot で追加のタスクを起動しようとします。タスクが 1 つしかないサービスは、キャパシティが利用可能になるまで中断されます。正常なシャットダウンの詳細については、「ECS による正常なシャットダウン」を参照してください。

タスクが停止する前にコンテナが正常に終了するように、以下を設定できます。

- タスクが使用しているコンテナ定義には、stopTimeout 秒以下の 120 値を指定できます。デフォルトの stopTimeout 値は 30 秒です。stopTimeout 値を長く指定すると、タスク状態変更イベントが受信した時点から、コンテナが強制的に停止するまでの時間を長くすることができます。詳細については、「コンテナのタイムアウト」を参照してください。
- クリーンアップアクションを実行するには、コンテナ内から SIGTERM シグナルを受信する必要があります。このシグナルの処理に失敗すると、stopTimeout が設定された後に SIGKILL シグナルを受信し、データが損失または破損する可能性があります。

タスク状態変更イベントのスニペットを以下に示します。このスニペットには、Fargate Spot 中断の停止理由と停止コードが表示されます。

```
{
  "version": "0",
  "id": "9bcdac79-b31f-4d3d-9410-fbd727c29fab",
  "detail-type": "ECS Task State Change",
  "source": "aws.ecs",
  "account": "111122223333",
  "resources": [
    "arn:aws:ecs:us-east-1:111122223333:task/b99d40b3-5176-4f71-9a52-9dbd6f1cebef"
  ],
  "detail": {
    "clusterArn": "arn:aws:ecs:us-east-1:111122223333:cluster/default",
    "createdAt": "2016-12-06T16:41:05.702Z",
    "desiredStatus": "STOPPED",
    "lastStatus": "RUNNING",
    "stoppedReason": "Your Spot Task was interrupted.",
    "stopCode": "SpotInterruption",
    "taskArn": "arn:aws:ecs:us-east-1:111122223333:task/
b99d40b3-5176-4f71-9a52-9dbd6fEXAMPLE",
    ...
```

```
  }
}
```

以下は、Amazon ECS タスク状態変更イベントの EventBridge ルールを作成するために使用されるイベントパターンです。オプションで、detail フィールドでクラスターを指定できます。そうすると、そのクラスターのタスク状態変更イベントを受信することになります。EventBridge ルールの作成の詳細については、「Amazon EventBridge ユーザーガイド」の「Amazon EventBridge の開始方法」を参照してください。

```
{
    "source": [
        "aws.ecs"
    ],
    "detail-type": [
        "ECS Task State Change"
    ],
    "detail": {
        "clusterArn": [
            "arn:aws:ecs:us-west-2:111122223333:cluster/default"
        ]
    }
}
```

Fargate ワークロード用の Amazon ECS クラスターを作成する

クラスターを作成して、タスクとサービスを実行するインフラストラクチャを定義します。

これを開始する前に、「Amazon ECS を使用するようにセットアップする」 の手順を完了し、適切な IAM 許可を割り当てる必要があります。詳細については、「the section called “Amazon ECS クラスターの例”」を参照してください。Amazon ECS コンソールでは、CloudFormation スタックを作成することで、Amazon ECS クラスターに必要なリソースを作成します。

コンソールは、自動的に Fargate および Fargate Spot キャパシティープロバイダーをクラスターに関連付けます。

以下のオプションを変更できます。

- クラスターに名前空間を追加します。

名前空間を使用すると、クラスターで作成したサービスを、追加の設定なしで名前空間内の他のサービスに接続できます。詳細については、「Amazon ECS サービスを相互接続する」を参照してください。

- タスクイベントを有効にして、タスク状態の変更に関する EventBridge 通知を受信します。
- クラスターを識別しやすいようにタグを追加します。
- マネージドストレージに AWS KMS キーを割り当てます。キーの作成方法の詳細については、AWS Key Management Service ユーザーガイドの「KMS キーを作成する」を参照してください。
- Fargate エフェメラルストレージに AWS KMS キーを割り当てます。キーの作成方法の詳細については、AWS Key Management Service ユーザーガイドの「KMS キーを作成する」を参照してください。
- ECS Exec の AWS KMS キーとログ記録を設定します。

手順

新しいクラスターを作成するには (Amazon ECS コンソール)

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションバーから、使用するリージョンを選択します。
3. ナビゲーションペインで [Clusters] (クラスター) を選択します。
4. [Clusters] (クラスター) ページで、[Create Cluster] (クラスターの作成) を選択します。
5. [クラスター設定] で以下を設定します。

   - [クラスター名] に一意の名前を入力します。

     名前には、最大 255 文字 (大文字と小文字)、数字、およびハイフンを含めることができます。

   - (オプション) Service Connect に使用する名前空間をクラスター名と別のものにするには、[Service Connect のデフォルト] の [名前空間] で、名前空間名を選択または入力します。共有名前空間を使用するには、名前空間 ARN を選択または入力します。共有名前空間の使用について詳しくは、「共有 AWS Cloud Map 名前空間を使用した Amazon ECS Service Connect」を参照してください。
6. (オプション) Container Insights を使用して モニタリング を展開し、次のいずれかのオプションを選択します。

- オブザーバビリティが強化された推奨の Container Insights を使用するには、[オブザーバビリティが強化された Container Insights] を選択します。
- Container Insights を使用するには、[Container Insights] を選択します。

7. (オプション) タスクイベントを有効にするには、[タスクイベント] を展開し、[タスクイベントを有効化] をオンにします。

   タスクイベントを有効にすると、Amazon ECS はタスク状態変更イベントを EventBridge に送信します。これにより、タスクライフサイクルの変更を自動的にモニタリングして対応できます。

8. (オプション) ECS Exec を使用してクラスター内のタスクをデバッグするには、[トラブルシューティング設定] を展開し、以下を設定します。

   - (オプション) [ECS Exec の AWS KMS キー] には、ECS Exec セッションデータの暗号化に使用する AWS KMS キーの ARN を入力します。
   - (オプション) [ECS Exec ログ記録] で、ログの送信先を選択します。
     - CloudWatch Logs にログを送信するには、[Amazon CloudWatch] を選択します。
     - Amazon S3 にログを送信するには、[Amazon S3] を選択します。
     - ログ記録を無効にするには、[なし] を選択します。

9. (オプション) [暗号化] で以下を設定できます。

   - Fargate エフェメラルストレージ上のデータを暗号化します。[暗号化] の [Fargate エフェメラルストレージ] で、Fargate エフェメラルストレージデータの暗号化に使用する AWS KMS キーの ARN を入力します。
   - マネージドストレージのデータを暗号化します。[暗号化] の [マネージドストレージ] に、マネージドストレージデータの暗号化に使用する AWS KMS キーの ARN を入力します。

10. (オプション) クラスターを識別しやすくするには、[Tags] (タグ) を展開し、タグを設定します。

    [タグの追加] [タグの追加] を選択して、以下を実行します。

    - [キー] にはキー名を入力します。
    - [値] にキー値を入力します。

    [タグを削除] タグのキーと値の右側にある [削除] を選択します。

11. [作成] を選択します。

次のステップ

クラスターを作成したら、アプリケーションのタスク定義を作成し、スタンドアロンタスク、またはサービスの一部として実行できます。詳細については次を参照してください:

- Amazon ECSの タスク定義
- Amazon ECS タスクとしてのアプリケーションの実行
- Amazon ECS のローリング更新デプロイの作成

EC2 ワークロード用の Amazon ECS キャパシティプロバイダー

キャパシティーとして Amazon EC2 インスタンスを使用する場合は、Auto Scaling グループを使用し、クラスターに登録されている Amazon EC2 インスタンスを管理します。Auto Scaling により、アプリケーションの負荷を処理するために適切な数の使用可能な Amazon EC2 インスタンスを確保できるようになります。

マネージドスケーリング機能を使用して、Auto Scaling グループのスケールインおよびスケールアウトアクションを Amazon ECS に管理させることも、自分自身でスケーリングアクションを管理することもできます。詳細については、「クラスターの自動スケーリングで Amazon ECS キャパシティーを自動的に管理する」を参照してください。

新しい空の Auto Scaling グループを作成することをお勧めします。既存の Auto Scaling グループを使用する場合、キャパシティプロバイダーの作成に使用される Auto Scaling グループの前に、既に実行され、Amazon EC2 クラスターに登録されていたグループに関連付けられた Amazon ECS インスタンスが、キャパシティプロバイダーに正しく登録されないことがあります。これにより、キャパシティプロバイダー戦略でキャパシティプロバイダーを使用するときに問題が発生する可能性があります。`DescribeContainerInstances` を使用し、コンテナインスタンスがキャパシティープロバイダーに関連付けられているかどうかを確認できます。

> **Note**
>
> 空の Auto Scaling グループを作成するには、必要なカウントをゼロに設定します。キャパシティプロバイダーを作成してクラスターに関連付けた後、スケールアウトできます。Amazon ECS コンソールを使用する場合、Amazon ECS は、CloudFormation スタックの一部として、ユーザーの代わりに Amazon EC2 起動テンプレートと Auto Scaling グループを作成します。これらには、プレフィックスとして `EC2ContainerService-`

<*ClusterName*> が付きます。Auto Scaling グループは、そのクラスターのキャパシティープロバイダーとして使用できます。

マネージドインスタンスドレイニングを使用して、ワークロードを中断することなく Amazon EC2 インスタンスを正常終了できるようにすることをお勧めします。この機能は、デフォルトでオンになっています。詳細については、EC2 インスタンスで実行されている Amazon ECS ワークロードを安全に停止するを参照してください。

コンソール内の Auto Scaling グループキャパシティープロバイダーを使用する場合は、次の点を考慮する必要があります。

- Auto Scaling グループをスケールアウトするには、その `MaxSize` が 0 より大きくなければなりません。
- Auto Scaling グループは、インスタンスの重み付けを設定することはできません。
- Auto Scaling グループが実行されるタスクの数に合わせてスケールアウトできない場合、タスクは `PROVISIONING` の後の状態に移行できません。
- キャパシティプロバイダーによって管理される Auto Scaling グループに関連付けられているスケーリングポリシーリソースは、変更しないでください。
- キャパシティプロバイダーの作成時にマネージドスケーリングがオンになっている場合、Auto Scaling グループの希望するカウントを `0` に設定できます。マネージドスケーリングがオンになっている場合、Amazon ECS は Auto Scaling グループのスケールインアクションとスケールアウトアクションを管理します。
- キャパシティープロバイダーは、キャパシティープロバイダー戦略に関連付ける前に、クラスターに関連付ける必要があります。
- キャパシティープロバイダー戦略には、最大 20 のキャパシティープロバイダーを指定できます。
- Auto Scaling グループキャパシティプロバイダーを使用するサービスは、Fargate キャパシティプロバイダーを使用するように更新することはできません。逆の場合も同様です。
- キャパシティプロバイダー戦略では、コンソールでキャパシティプロバイダーに `weight` 値が指定されていない場合、1 のデフォルト値が使用されます。API または AWS CLI を使用する場合は、`0` のデフォルト値が使用されます。
- キャパシティプロバイダー戦略内で複数のキャパシティプロバイダーを指定する場合、少なくとも 1 つのキャパシティプロバイダーのウェイト値が 0 より大きい必要があります。ウェイトが 0 のキャパシティープロバイダーはタスクの配置に使用されません。戦略に複数のキャパシティプロバ

イダーを指定し、すべて同じウェイトを 0 にした場合、キャパシティプロバイダー戦略を使用する RunTask または CreateService のアクションは失敗します。

- キャパシティプロバイダー戦略では、1 つのキャパシティプロバイダーのみが定義されたベース値を持つことができます。ベース値を指定しない場合は、デフォルト値の 0 が使用されます。
- クラスターには、Auto Scaling グループキャパシティプロバイダーと Fargate キャパシティプロバイダーの両方を混在させることができます。ただし、キャパシティプロバイダー戦略に含めることができるのは Auto Scaling グループまたは Fargate キャパシティプロバイダーのみで、両方を含めることはできません。
- クラスターには、キャパシティプロバイダーと起動タイプの両方を使用するサービスとスタンドアロンタスクを混在させることができます。サービスは、起動タイプではなくキャパシティプロバイダー戦略を使用するように更新できます。ただし、その場合は強制的に新しいデプロイを行う必要があります。
- Amazon ECS では、Amazon EC2 Auto Scaling ウォームプールをサポートします。ウォームプールは、事前に初期化済みの Amazon EC2 インスタンスグループでサービス開始が準備されています。アプリケーションがスケールアウトする必要がある場合は、常に Amazon EC2 Auto Scaling はコールドインスタンスを起動するのではなく、ウォームプールから事前に初期化されたインスタンスを使用します。これにより、インスタンスがサービスを開始する前に、最終的な初期化プロセスを実行できるようになります。詳細については、「Amazon ECS Auto Scaling グループ用に事前初期化されたインスタンスを設定する」を参照してください。

Amazon EC2 Auto Scaling 起動テンプレートの作成についての詳細は、「Amazon EC2 Auto Scaling ユーザーガイド」の「Auto Scaling 起動テンプレート」を参照してください。Amazon EC2 Auto Scaling グループの作成についての詳細は、「Amazon EC2 Auto Scaling ユーザーガイド」の「Auto Scaling グループ」を参照してください。

Amazon ECS の Amazon EC2 コンテナインスタンスのセキュリティに関する考慮事項

脅威モデル内での単一のコンテナインスタンスとそのアクセスを考慮する必要があります。たとえば、影響を受けるひとつのタスクが、同じインスタンス上の感染していないタスクの IAM アクセス許可を利用できる場合があります。

これを防ぐには、次のことを行うことをお勧めします。

- タスクを実行するときは、管理者権限を使用しないでください。
- タスクに割り当てるタスクロールは最小特権にします。

コンテナエージェントは、Amazon ECS リソースへのアクセスに使用される固有の認証情報 ID を持つトークンを自動的に作成します。

- awsvpc ネットワークモードを使用するタスクで実行されたコンテナが、Amazon EC2 のインスタンスプロファイルに入力されている認証情報にアクセスするのを防止しつつ、タスクロールで指定されている許可を有効にするには、エージェントの設定ファイルで ECS_AWSVPC_BLOCK_IMDS エージェント設定変数を [true] に設定し、そのエージェントを再起動します。
- Amazon GuardDuty ランタイムモニタリングを使用して、AWS 環境内のクラスターとコンテナの脅威を検出します。Runtime Monitoring では、ファイルアクセス、プロセス実行、ネットワーク接続などの個々の Amazon ECS ワークロードを実行時に可視化する、GuardDuty セキュリティエージェントを使用します。詳細については、「GuardDuty ユーザーガイド」の「GuardDuty ランタイムモニタリング」を参照してください。

Amazon EC2 ワークロード用の Amazon ECS クラスターを作成する

クラスターを作成して、タスクとサービスを実行するインフラストラクチャを定義します。

これを開始する前に、「Amazon ECS を使用するようにセットアップする」 の手順を完了し、適切な IAM 許可を割り当てる必要があります。詳細については、「the section called “Amazon ECS クラスターの例”」を参照してください。Amazon ECS コンソールでは、CloudFormation スタックを作成することで、Amazon ECS クラスターに必要なリソースを簡単に作成できます。

クラスターの作成プロセスをできるだけ簡単にするために、コンソールには、次のセクションで説明する多くの選択肢に対するデフォルトの選択肢があります。コンソール内のほとんどのセクションには、詳細なコンテキストを提供するヘルプパネルもあります。

クラスターの作成時に Amazon EC2 インスタンスを登録することや、作成後のクラスターに対し、追加のインスタンスを登録することもできます。

以下のデフォルトオプションを変更できます。

- インスタンスを起動するサブネットを変更します。
- コンテナインスタンスへのトラフィックを制御するために使用されるセキュリティグループを変更します。
- クラスターに名前空間を追加します。

名前空間を使用すると、クラスターで作成したサービスを、追加の設定なしで名前空間内の他のサービスに接続できます。詳細については、「Amazon ECS サービスを相互接続する」を参照してください。

- タスクイベントを有効にして、タスク状態の変更に関する EventBridge 通知を受信します。
- マネージドストレージに AWS KMS キーを割り当てます。キーの作成方法の詳細については、AWS Key Management Service ユーザーガイドの「KMS キーを作成する」を参照してください。
- Fargate エフェメラルストレージに AWS KMS キーを割り当てます。キーの作成方法の詳細については、AWS Key Management Service ユーザーガイドの「KMS キーを作成する」を参照してください。
- ECS Exec の AWS KMS キーとログ記録を設定します。
- クラスターを識別しやすいようにタグを追加します。

Auto Scaling グループオプション

Amazon EC2 インスタンスを使用する場合は、タスクとサービスが実行されるインフラストラクチャを管理するために Auto Scaling グループを指定する必要があります。

新しい Auto Scaling グループの作成を選択すると、自動的に次の動作が設定されます。

- Amazon ECS は Auto Scaling グループのスケールインアクションとスケールアウトアクションを管理します。
- Amazon ECS は、タスクを含む Auto Scaling グループ内の Amazon EC2 インスタンスがスケールインアクション中に終了されるのを防ぎます。詳細については、AWS Auto Scaling ユーザーガイドの「インスタンスの保護」を参照してください。

次の Auto Scaling グループプロパティを構成し、グループに対して起動するインスタンスのタイプと数を決定します。

- Amazon ECS に最適化された AMI。
- インスタンスタイプ。
- インスタンスに接続するときに ID を証明する SSH キーペア。SSH キーの作成については、「Amazon EC2 ユーザーガイド」の「Amazon EC2 のキーペアと Linux インスタンス」を参照してください。
- Auto Scaling グループに対して起動するインスタンスの最小数。

- Auto Scaling グループに対して開始されるインスタンスの最大数。

  グループをスケールアウトするには、最大値を 0 より大きくする必要があります。

Amazon ECS は、CloudFormation スタックの一部としてユーザーに代わり Amazon EC2 Auto Scaling 起動テンプレートと Auto Scaling グループを作成します。AMI、インスタンスタイプ、SSH キーペアのために指定した値は、起動テンプレート内に保存されます。テンプレートは `EC2ContainerService-<ClusterName>` プレフィックスが付いているため、識別が容易になります。Auto Scaling グループのプレフィックスは `<ClusterName>-ECS-Infra-ECSAutoScalingGroup` です。

Auto Scaling グループに対して起動されたインスタンスは、起動テンプレートを使用します。

ネットワークのオプション

デフォルトでは、インスタンスはリージョンのデフォルトのサブネットで起動されます。コンテナインスタンスへのトラフィックを制御するセキュリティグループは、現在サブネットに関連付けられているものが使用されます。インスタンスのサブネットとセキュリティグループは変更できます。

既存のサブネットを選択できます。既存のセキュリティグループを使用するか、セキュリティグループを新規作成できます。IPv6 のみの設定でタスクを作成するには、IPv6 CIDR ブロックのみを含むサブネットを使用します。

セキュリティグループを新規作成する場合は、少なくとも 1 つのインバウンドルールを指定する必要があります。

インバウンドルールは、コンテナインスタンスに到達できるトラフィックを決定し、以下のプロパティを含みます。

- 許可するプロトコル
- 許可するポートの範囲
- インバウンドトラフィック (送信元)

特定のアドレスまたは CIDR ブロックからのインバウンドトラフィックを許可するには、許可された CIDR で [送信元] に [カスタム] を使用します。

すべての送信先からのインバウンドトラフィックを許可するには、[送信元] に [すべて] を使用してください。これにより、これにより、0.0.0.0/0 IPv4 CIDR ブロックと::/0 IPv6 CIDR ブロックが自動的に追加されます。

ローカルコンピューターからのインバウンドトラフィックを許可するには、[送信元] に [ソースグループ] を使用します。これにより、ローカルコンピュータの現在の IP アドレスが送信元アドレスとして自動的に追加されます。

新しいクラスターを作成するには (Amazon ECS コンソール)

開始する前に、適切な IAM アクセス許可を割り当ててください。詳細については、「the section called “Amazon ECS クラスターの例”」を参照してください。

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションバーから、使用するリージョンを選択します。
3. ナビゲーションペインで [Clusters] (クラスター) を選択します。
4. [Clusters] (クラスター) ページで、[Create Cluster] (クラスターの作成) を選択します。
5. [クラスター設定] で以下を設定します。
   - [クラスター名] に一意の名前を入力します。

     名前には、最大 255 文字 (大文字と小文字)、数字、およびハイフンを含めることができます。
   - (オプション) Service Connect に使用する名前空間をクラスター名と別のものにするには、[Service Connect のデフォルト] の [名前空間] で、名前空間名を選択または入力します。共有名前空間を使用するには、名前空間 ARN を選択または入力します。共有名前空間の使用について詳しくは、「共有 AWS Cloud Map 名前空間を使用した Amazon ECS Service Connect」を参照してください。
6. Amazon EC2 インスタンスをクラスターに追加するには、[インフラストラクチャ] を展開し、次に [Fargate インスタンスとセルフマネージドインスタンス] を選択します。

   次に、キャパシティープロバイダーとして機能する Auto Scaling グループを設定します。

   a. 既存の Auto Scaling グループを使用するには、[Auto Scaling group (ASG)] (Auto Scaling グループ) からグループを選択します。

   b. Auto Scaling グループを作成するには、Auto Scaling group(ASG) (Auto Scaling グループ) から、[Create new group] (新しいグループの作成) を選択し、グループに関する以下の詳細情報を入力します。

      - [プロビジョニングモデル] で、[オンデマンド]インスタンスと[スポット]インスタンスのどちらを使用するかを選択します。

- スポットインスタンスを使用する場合は、[割り当て戦略] で、インスタンスに使用するスポットキャパシティプール (インスタンスタイプおよびアベイラビリティーゾーン) を選択します。

  ほとんどのワークロードでは、[料金キャパシティ最適化] を選択できます。

  詳細については、「Amazon EC2 ユーザーガイド」の「スポットインスタンスの配分戦略」を参照してください。
- コンテナインスタンスの Amazon マシンイメージ (AMI) で、Auto Scaling グループインスタンスの [Amazon ECS 最適化 AMI] を選択します。
- [EC2 instance type] (EC2 インスタンスタイプ) を使用する場合、ワークロードのインスタンスタイプを選択します。

  マネージドスケーリングは、Auto Scaling グループが同じインスタンスタイプまたは類似のインスタンスタイプを使用している場合に最適です。
- [EC2 インスタンスロール]では、既存のコンテナインスタンスロールを選択するか、新しいコンテナインスタンスロールを作成できます。

  詳細については、「Amazon ECS コンテナインスタンスの IAM ロール」を参照してください。
- [Capacity] (キャパシティー) には、Auto Scaling グループで起動するインスタンスの最小数と最大数を入力します。
- [SSH key pair] (SSH キーペア) を使用する場合、インスタンスに接続する際に ID を証明するペアを選択してください。
- より大きなイメージとストレージを許容するには、[ルート EBS ボリュームサイズ] に単位 GiB で値を入力します。

7. (オプション) VPC とサブネットを変更するには、[Amazon EC2 インスタンスのネットワーキング] で、次のいずれかの操作を実行します。

   - サブネットを削除するには、[Subnets] (サブネット) で、削除するサブネットごとに [X] を選択します。
   - デフォルト VPC 以外の VPC に変更するには、[VPC] で既存の VPC を選択し、[サブネット] でサブネットを選択します。IPv6 のみの設定の場合は、IPv6 CIDR ブロックを持つ VPC と IPv6 CIDR ブロックのみを持つサブネットを選択します。
   - セキュリティグループを選択します。[セキュリティグループ] で、次のいずれかのオプションを選択します。

- 既存のセキュリティグループを選択するには、[既存のセキュリティグループの選択] を選択してから、セキュリティグループを選択します。
- 新しいセキュリティグループを作成するには、[新しいセキュリティグループの作成] を選択します。その後、各インバウンドルールについて [ルールの追加] を選択します。

  インバウンドルールの詳細については、「ネットワークのオプション」を参照してください。

- Amazon EC2 コンテナインスタンスにパブリック IP アドレスを自動的に割り当てるには、[パブリック IP の自動割り当て] で以下のいずれかのオプションを選択します。
  - サブネット設定を使用 — インスタンスが起動するサブネットがパブリックサブネットの場合、パブリック IP アドレスをインスタンスに割り当てます。
  - オンにする — インスタンスにパブリック IP アドレスを割り当てます。

8. (オプション) Container Insights を使用して モニタリング を展開し、次のいずれかのオプションを選択します。

   - オブザーバビリティが強化された推奨の Container Insights を使用するには、[オブザーバビリティが強化された Container Insights] を選択します。
   - Container Insights を使用するには、[Container Insights] を選択します。

9. (オプション) タスクイベントを有効にするには、[タスクイベント] を展開し、[タスクイベントを有効化] をオンにします。

   タスクイベントを有効にすると、Amazon ECS はタスク状態変更イベントを EventBridge に送信します。これにより、タスクライフサイクルの変更を自動的にモニタリングして対応できます。

10. (オプション) ECS Exec を使用してクラスター内のタスクをデバッグするには、[トラブルシューティング設定] を展開し、以下を設定します。

    - (オプション) [ECS Exec の AWS KMS キー] には、ECS Exec セッションデータの暗号化に使用する AWS KMS キーの ARN を入力します。
    - (オプション) [ECS Exec ログ記録] で、ログの送信先を選択します。
      - CloudWatch Logs にログを送信するには、[Amazon CloudWatch] を選択します。
      - Amazon S3 にログを送信するには、[Amazon S3] を選択します。
      - ログ記録を無効にするには、[なし] を選択します。

11. (オプション)

ランタイムのモニタリングを手動オプションで使用していて、このクラスターを GuardDuty でモニタリングしたい場合は、[タグを追加] を選択して次の操作を行います。

- [キー] で「**guardDutyRuntimeMonitoringManaged**」と入力します。
- [Value] (値) に「**true**」と入力します。

12. (オプション) マネージドストレージのデータを暗号化します。[暗号化] の [マネージドストレージ] に、マネージドストレージデータの暗号化に使用する AWS KMS キーの ARN を入力します。
13. (オプション) クラスタータグを管理するには、[Tags] (タグ) を展開し、次のいずれかのオペレーションを実行します。

    [タグの追加] [タグの追加] を選択して、以下を実行します。

    - [キー] にはキー名を入力します。
    - [値] にキー値を入力します。

    [タグを削除] タグのキーと値の右側にある [削除] を選択します。
14. [作成] を選択します。

次のステップ

クラスターを作成したら、アプリケーションのタスク定義を作成し、スタンドアロンタスク、またはサービスの一部として実行できます。詳細については次を参照してください:

- Amazon ECSの タスク定義
- Amazon ECS タスクとしてのアプリケーションの実行
- Amazon ECS のローリング更新デプロイの作成

クラスターの自動スケーリングで Amazon ECS キャパシティーを自動的に管理する

Amazon ECS では、クラスターに登録された Amazon EC2 インスタンスのスケーリングを管理できます。これは、Amazon ECS クラスターの自動スケーリングと呼ばれます。Amazon ECS Auto Scaling グループのキャパシティープロバイダーを作成するときに、マネージドスケーリングを有効にします。その後、この Auto Scaling グループのインスタンス使用率のターゲットパーセン

テージ (`targetCapacity`) を設定します。Amazon ECS は 2 つのカスタム CloudWatch メトリクス、および Auto Scaling グループに対するターゲット追跡スケーリングポリシーを作成します。また、Amazon ECS は、タスクが使用するリソース使用率に基づいて、スケールインアクションとスケールアウトアクションを管理します。

クラスターに関連付けられている各 Auto Scaling グループのキャパシティプロバイダーについて、Amazon ECS は次のリソースを作成し、管理します。

- 低いメトリクス値の CloudWatch アラーム
- 高いメトリクス値の CloudWatch アラーム
- ターゲットの追跡スケーリングポリシー

> **Note**
>
> Amazon ECS は、ターゲットの追跡スケーリングポリシーを作成し、Auto Scaling グループにアタッチします。ターゲットの追跡スケーリングポリシーを更新するには、スケーリングポリシーを直接更新するのではなく、キャパシティプロバイダーのマネージドスケーリング設定を更新します。

マネージドスケーリングを無効にするか、クラスターからキャパシティプロバイダーの関連付けを解除すると、Amazon ECS は CloudWatch メトリクスとターゲットの追跡スケーリングポリシーのリソースの両方を削除します。

Amazon ECS では、実行するアクションを決定するのに次のメトリクスを使用します。

`CapacityProviderReservation`

特定のキャパシティプロバイダーで使用されているコンテナインスタンスの割合。Amazon ECS はこのメトリクスを生成します。

Amazon ECS は、`CapacityProviderReservation` の値を 0 ~ 100 の数値に設定します。Amazon ECS は、次の式を使用して Auto Scaling グループに残っているキャパシティの割合を表します。その後、Amazon ECS は CloudWatch にメトリクスをパブリッシュします。メトリクスの算出方法の詳細については、「Amazon ECS クラスター自動スケーリングを詳しく知る」を参照してください。

```
CapacityProviderReservation = (number of instances needed) / (number of running
 instances) x 100
```

`DesiredCapacity`

Auto Scaling グループのキャパシティ量。このメトリクスは CloudWatch に公開されていません。

Amazon ECS は `AWS/ECS/ManagedScaling` 名前空間内の CloudWatch に `CapacityProviderReservation` メトリックを公開します。`CapacityProviderReservation` メトリクスは、次のいずれかのアクションを実行します。

`CapacityProviderReservation` の値は `targetCapacity` に等しいです

Auto Scaling グループはスケールインまたはスケールアウトする必要はありません。目標使用率に達しました。

`CapacityProviderReservation` の値は `targetCapacity` より大きいです

キャパシティのパーセンテージを使用しているタスクの数が、自分の `targetCapacity` のパーセンテージを上回っています。`CapacityProviderReservation` メトリクスの値が増加すると、関連する CloudWatch アラームが動作します。このアラームは Auto Scaling グループの `DesiredCapacity` 値を更新します。Auto Scaling グループはこの値を使用して EC2 インスタンスを起動し、クラスターに登録します。

`targetCapacity` がデフォルト値の 100% の場合、インスタンスにタスクを実行できる空き容量がないため、スケールアウト中は新しいタスクは `PENDING` 状態になります。新しいインスタンスが ECS に登録されると、これらのタスクは新しいインスタンスで開始されます。

`CapacityProviderReservation` の値は `targetCapacity` 未満です

キャパシティのパーセンテージを使用しているタスクが自分の `targetCapacity` のパーセンテージよりも少なく、終了できるインスタンスが少なくとも 1 つあります。`CapacityProviderReservation` メトリクスの値が減少すると、関連する CloudWatch アラームが動作します。このアラームは Auto Scaling グループの `DesiredCapacity` 値を更新します。Auto Scaling グループはこの値を使用して EC2 コンテナインスタンスを終了し、クラスターから登録解除します。

Auto Scaling グループは、終了ポリシーを使用して、スケールインイベント中に最初に終了するインスタンスを決定します。さらに、インスタンスのスケールイン保護の設定は、回避します。クラスター自動スケーリングでは、マネージドターミネーション保護を有効にすると、どのインスタンスにインスタンススケールイン保護が設定されているかを管理できます。終了保護の詳細については、「Amazon ECS が終了するインスタンスの制御」を参照してください。詳細につい

ては、Amazon EC2 Auto Scaling ユーザーガイドの「スケールイン時にどの自動スケーリングインスタンスを終了するかのコントロール」を参照してください。

クラスターの自動スケーリングを使用するときは、次の点を考慮してください。

- キャパシティプロバイダーに関連付けられている Auto Scaling グループが希望するキャパシティは、Amazon ECS が管理しているもの以外のスケーリングポリシーを変更または管理しないでください。
- Amazon ECS が 0 インスタンスからスケールアウトする場合、自動的に 2 つのインスタンスを起動します。
- Amazon ECS は、ユーザーに代わって AWS Auto Scaling を呼び出すために必要なアクセス許可として `AWSServiceRoleForECS` サービスにリンクされた IAM ロールを使用します。詳細については、「Amazon ECS のサービスリンクロールの使用」を参照してください。
- Auto Scaling グループでキャパシティプロバイダーを使用する場合、キャパシティプロバイダーを作成するユーザー、グループ、ロールには `autoscaling:CreateOrUpdateTags` アクセス許可が必要です。これは、Auto Scaling グループが、キャパシティープロバイダーに関連付けるときに、Amazon ECS が Auto Scaling グループにタグを追加するためです。

  > **Important**
  >
  > ツールの使用により `AmazonECSManaged` タグが Auto Scaling グループから削除されないようにしてください。このタグが削除されると、Amazon ECS はスケーリングを管理できません。

- クラスターの自動スケーリングは、グループの [MinimumCapacity] と [MaximumCapacity] を変更しません。グループをスケールアウトするには、[MaximumCapacity] を 0 より大きくする必要があります。
- 自動スケーリング(マネージドスケーリング) がオンになっている場合、キャパシティプロバイダーは、同時に 1 つのクラスターにしか接続できません。キャパシティープロバイダーがマネージドスケーリングをオフにしている場合は、複数のクラスターに関連付けることができます。
- マネージドスケーリングがオフの場合、キャパシティプロバイダーはスケールインまたはスケールアウトを実行しません。キャパシティプロバイダー戦略を使用して、キャパシティプロバイダー間でタスクのバランスを取ることができます。
- `binpack` 戦略は、キャパシティに関して最も効率的な戦略です。

- ターゲットキャパシティが 100% 未満の場合、配置戦略内では、binpack 戦略が spread 戦略よりも高い優先順位を持つ必要があります。こうすることで、各タスクにハードウェア専有インスタンスが割り当てられるか、上限に達するまで、キャパシティプロバイダーはスケールアウトできなくなります。

クラスターの自動スケーリングをオンにする

コンソールまたは AWS CLI を使用して、クラスターの自動スケーリングをオンにすることができます。

コンソールを使用して EC2 キャパシティプロバイダーを用いるクラスターを作成すると、Amazon ECS がユーザーに代わって Auto Scaling グループを作成し、ターゲット容量を設定します。詳細については、「Amazon EC2 ワークロード用の Amazon ECS クラスターを作成する」を参照してください。

ユーザーが Auto Scaling グループを作成してクラスターに割り当てることもできます。詳細については、「Amazon ECS キャパシティープロバイダーを更新する」を参照してください。

AWS CLI を使用する場合、クラスターを作成した後、次の手順に従います。

1. キャパシティプロバイダーを作成する前に、Auto Scaling グループを作成する必要があります。詳細については、「Amazon EC2 Auto Scaling ユーザーガイド」の「Auto Scaling グループ」を参照してください。
2. `put-cluster-capacity-providers` を使用して、クラスターキャパシティプロバイダーを変更します。詳細については、「Amazon ECS クラスターの自動スケーリングを有効にする」を参照してください。

Amazon ECS クラスターの自動スケーリングの最適化

Amazon EC2 で Amazon ECS を実行するお客様は、クラスターの自動スケーリングを利用して Amazon EC2 Auto Scaling グループのスケーリングを管理できます。クラスターの自動スケーリングを使用すると、Auto Scaling グループを自動的にスケーリングするように Amazon ECS を設定することができ、ユーザーはタスクの実行に集中できます。Amazon ECS では、追加の操作を必要とせずに、必要に応じて Auto Scaling グループがスケールインおよびスケールアウトします。Amazon ECS キャパシティープロバイダーは、アプリケーションの需要を満たすのに十分なコンテナインスタンスを確保することで、クラスター内のインフラストラクチャを管理するために使用されます。クラスターの自動スケーリングが内部でどのように機能するかについては、「Deep Dive on Amazon ECS Cluster Auto Scaling」を参照してください。

クラスターの自動スケーリングは、Auto Scaling グループとの CloudWatch ベースの 統合を利用して、クラスターキャパシティーを調整します。したがって、次に関連する固有のレイテンシーがあります。

- CloudWatch メトリクスの発行
- メトリクス `CapacityProviderReservation` が CloudWatch アラームに抵触 (高と低の両方) するまでにかかる時間
- 新しく起動した Amazon EC2 インスタンスがウォームアップするのにかかる時間 以下のアクションを実行して、クラスターの自動スケーリングの応答性を高め、デプロイを高速化できます。

キャパシティープロバイダーのステップスケーリングサイズ

Amazon ECS キャパシティープロバイダーは、アプリケーションの需要に合わせてコンテナインスタンスを拡大/縮小します。Amazon ECS が起動するインスタンスの最小数は、デフォルトでは 1 に設定されています。そのため、保留中のタスクを配置するために複数のインスタンスが必要な場合は、デプロイにさらに時間がかかることがあります。Amazon ECS API を介して `minimumScalingStepSize` を増やすことで、Amazon ECS が一度にスケールインまたはスケールアウトするインスタンスの最小数を増やすことができます。`maximumScalingStepSize` が小さすぎると、一度にスケールインまたはスケールアウトされるコンテナインスタンスの数が制限され、デプロイが遅くなる可能性があります。

> **Note**
>
> この設定は現在、`CreateCapacityProvider` または `UpdateCapacityProvider` API を介してのみ使用できます。

インスタンスのウォームアップ期間

インスタンスのウォームアップ期間は、新たに起動された Amazon EC2 インスタンスが Auto Scaling グループの CloudWatch メトリックスに反映されるまでの時間です。指定されたウォームアップ期間が終了すると、そのインスタンスは Auto Scaling グループの集計メトリクスにカウントされ、クラスターの自動スケーリングは、必要なインスタンス数を推定するための次の計算ループに進みます。

`instanceWarmupPeriod` のデフォルト値は 300 秒です。この値を `CreateCapacityProvider` または `UpdateCapacityProvider` API を使用して小さい値に設定することで、スケーリングの応

答性を向上させることができます。過剰なプロビジョニングを回避できるように、値を 60 秒以上に設定することをお勧めします。

予備のキャパシティー

キャパシティープロバイダーにタスクを配置するために使用できるコンテナインスタンスがない場合は、Amazon EC2 インスタンスをその場で起動してクラスターキャパシティーを増やし (スケールアウトし)、それらのインスタンスが起動するまで待ってからコンテナを起動する必要があります。これにより、タスクの起動レートが大幅に低下する可能性があります。ここでは 2 つのオプションがあります。

この場合、予備の Amazon EC2 キャパシティーを事前に起動してタスクを実行する準備をしておくことで、実質的なタスク起動レートを上げることができます。`Target Capacity` 設定を使用して、クラスターで予備のキャパシティーを保持するかどうかを指定できます。例えば、`Target Capacity` を 80 % に設定することで、クラスターに常に 20 % の予備のキャパシティーが必要であることを示します。この予備のキャパシティーにより、スタンドアロンタスクをすぐに起動できるようになり、タスクの起動がスロットリングされなくなります。このアプローチのトレードオフは、予備のクラスターキャパシティーを保持するコストが増加する可能性があることです。

検討できる代替アプローチは、キャパシティープロバイダーではなくサービスにヘッドルームを追加することです。つまり、予備のキャパシティーを起動するための `Target Capacity` 設定を小さくする代わりに、ターゲット追跡スケーリングメトリクス、またはサービス自動スケーリングのステップスケーリングのしきい値を変更することで、サービス内のレプリカの数を増やすことができます。このアプローチが有用なのはワークロードの急増に対してのみであり、新しいサービスをデプロイし、初めて 0 から N タスクに移行する場合には効果がないことに注意してください。関連するスケーリングポリシーの詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「ターゲット追跡スケーリングポリシー」または「ステップスケーリングポリシー」を参照してください。

Amazon ECS マネージドスケーリング動作

マネージドスケーリングを使用する Auto Scaling グループキャパシティプロバイダーがある場合、Amazon ECS はクラスターに追加する最適なインスタンス数を見積もり、この値を使用してリクエストまたはリリースするインスタンス数を決定します。

マネージドスケールアウト動作

Amazon ECS は、サービス、スタンドアロンタスク、またはクラスターのデフォルトからのキャパシティープロバイダー戦略に従って、各タスクのキャパシティープロバイダーを選択しま

す。Amazon ECS は、単一のキャパシティプロバイダーのために、これらの残りのステップに従います。

キャパシティプロバイダー戦略のないタスクはキャパシティプロバイダーによって無視されます。キャパシティプロバイダー戦略がない保留中のタスクによって、キャパシティプロバイダーがスケールアウトされることはありません。タスクまたはサービスが起動タイプを設定する場合、タスクまたはサービスはキャパシティプロバイダー戦略を設定できません。

以下では、スケールアウト動作について詳しく説明します。

- このキャパシティプロバイダーのすべてのプロビジョニングタスクをグループ化し、各グループが同じリソース要件を持つようにします。
- グループ内の複数のインスタンスタイプを使用する場合、Auto Scaling グループ内のインスタンスタイプはパラメータによってソートされます。これらのパラメータには vCPU、メモリ、Elastic Network Interface (ENI)、ポート、GPU が含まれます。各パラメータの最小インスタンスタイプと最大インスタンスタイプが選択されます。インスタンスタイプの選択方法の詳細については、「Amazon ECS 用の Amazon EC2 コンテナインスタンス」を参照してください。

  > **Important**
  >
  > タスクのグループに、Auto Scaling グループの最小のインスタンスタイプよりも大きなリソース要件がある場合、そのタスクのグループは、このキャパシティプロバイダーでは実行できません。キャパシティプロバイダーは、Auto Scaling グループをスケールしません。タスクは `PROVISIONING` 状態のままです。
  > タスクが `PROVISIONING` 状態にとどまらないようにするには、最小リソース要件ごとに個別の Auto Scaling グループとキャパシティプロバイダーを作成することをお勧めします。タスクを実行するか、サービスを作成するときは、Auto Scaling グループ内の最小インスタンスタイプでタスクを実行できるキャパシティプロバイダーのみをキャパシティプロバイダー戦略に追加します。他のパラメータでは、配置制約を使用できます

- タスクグループごとに、Amazon ECS は未配置タスクの実行に必要なインスタンス数を計算します。この計算には、`binpack` 戦略が用いられます。この戦略では、タスクの vCPU、メモリ、Elastic Network Interface (ENI)、ポートや GPU の要件を考慮します。また、Amazon EC2 インスタンスのリソースの可用性も考慮されます。最大インスタンスタイプの値は、計算された最大インスタンス数として扱われます。最小のインスタンスタイプの値は、保護として使用されます。最小インスタンスタイプでタスクの少なくとも 1 つのインスタンスを実行できない場合、計算ではタスクが互換性がないと見なされます。その結果、タスクはスケールアウト計算から除外されます。すべてのタスクに最小のインスタンスタイプとの互換性がないときは、クラスター 自動ス

ケーリングは停止し、CapacityProviderReservation 値は targetCapacity のままになります。

- Amazon ECS は、次のいずれかに該当する場合、minimumScalingStepSize に関連した CloudWatch に CapacityProviderReservation メトリクスをパブリッシュします。
  - 計算された最大インスタンス数が最小スケーリングステップサイズを下回っています。
  - maximumScalingStepSize か、計算された最大インスタンス数のどちらか低い方の値。
- CloudWatch アラームは、キャパシティプロバイダーの CapacityProviderReservation メトリクスを使用します。CapacityProviderReservation メトリクスが targetCapacity 値より大きい場合、アラームでは、Auto Scaling グループの DesiredCapacity も増加します。targetCapacity 値は、クラスターの自動スケーリングがアクティブ化されているフェーズ中に CloudWatch アラームに送信されるキャパシティプロバイダー設定です。

  デフォルトの targetCapacity は 100 % です。
- Auto Scaling グループは追加の EC2 インスタンスを起動します。オーバープロビジョニングを防ぐため、Auto Scaling では、最近起動された EC2 インスタンスのキャパシティが新たなインスタンスを起動する前に安定させています。自動スケーリングは、既存のすべてのインスタンスが instanceWarmupPeriod (現在はインスタンスの起動時間を減じたもの) を経過したかどうかを確認します。instanceWarmupPeriod 内にあるインスタンスのスケールアウトはブロックされます。

  新しく起動されたインスタンスがウォームアップに達するまでのデフォルト秒数は 300 です。

詳細については、「Amazon ECS のクラスター自動スケーリングに関する詳細な説明」を参照してください。

スケールアウトの考慮事項

スケールアウトのプロセスでは、次の点を考慮してください。

- 配置制約は複数ありますが、distinctInstance タスク配置の制約事項のみを使用することをお勧めします。サンプリングされたインスタンスと互換性がない配置制約を使用しているため、これにより、スケールアウトプロセスが停止するのを防ぐことができます。
- マネージドスケーリングは、Auto Scaling グループが同じインスタンスタイプまたは類似のインスタンスタイプを使用している場合に最適です。
- スケールアウトプロセスが必要で現在実行中のコンテナインスタンスがない場合は、Amazon ECS は常に 2 つのインスタンスにスケールアウトしてから追加のスケールアウト/インプロセスを実行

します。追加のスケールアウトは、インスタンスのウォームアップ期間を待ちます。スケールインプロセスの場合、Amazon ECS はスケールアウトプロセスの後 15 分待ってから、常にスケールインプロセスを開始します。

- 2 番目のスケールアウトのステップは `instanceWarmupPeriod` が期限切れになるまで待つ必要があるため、全体的なスケール制限に影響する可能性があります。この時間を短縮する必要がある場合は、`instanceWarmupPeriod` が EC2 インスタンスが Amazon ECS エージェントを起動して開始するのに十分な大きさであることを確認してください (オーバープロビジョニングを防げます)。
- クラスターの自動スケーリングは、キャパシティプロバイダーの Auto Scaling グループ内の起動設定、起動テンプレート、複数のインスタンスタイプをサポートします。複数のインスタンスタイプを使用せずに、属性ベースのインスタンスタイプの選択も使用できます。
- オンデマンドインスタンスと複数のインスタンスタイプ、またはスポットインスタンスを持つ Auto Scaling グループを使用する場合は、大きいインスタンスタイプを優先順位リストで上位に配置し、ウェイトを指定しないでください。現時点では、ウェイトの指定はサポートされていません。詳細については、AWS Auto Scalingユーザーガイドの「複数のインスタンスタイプと Auto Scaling グループ」を参照してください。
- Amazon ECS は、計算された最大インスタンス数が最小スケーリングステップサイズより小さい場合は `minimumScalingStepSize` を、あるいは `maximumScalingStepSize` または計算された最大インスタンスカウント値のいずれか小さい方を起動します。
- Amazon ECS サービスまたは `run-task` がタスクを起動し、キャパシティープロバイダーのコンテナインスタンスにタスクを開始するために十分なリソースがない場合、Amazon ECS は、各クラスターでこのステータスのタスク数を制限し、タスクがこの制限を超えることを防ぎます。詳細については、「Service Quotas」を参照してください。

マネージドスケールイン動作

Amazon ECS は、クラスター内の各キャパシティプロバイダーのコンテナインスタンスをモニタリングします。コンテナインスタンスがタスクを実行していない場合、コンテナインスタンスは空であるとみなされ、Amazon ECS はスケールインプロセスを開始します。

CloudWatch スケールインアラームは、Auto Scaling グループのスケールインプロセスが開始する前に 15 データポイント (15 分) を必要とします。スケールインプロセスがスタートされた後から Amazon ECS が登録されたコンテナインスタンス数を減らす必要があるまで、Auto Scaling グループは `DesireCapacity` 値を 1 つのインスタンスより大きく、かつ毎分 50% 未満に設定します。

Amazon ECS がスケールアウトをリクエストしたときに (CapacityProviderReservation が 100 より大きいとき) スケールインプロセスが進行中の場合、スケールインプロセスは停止して、必要に応じて最初から開始されます。

次では、スケールイン動作について詳しく説明します。

1. Amazon ECS は、空のコンテナインスタンスの数を計算します。デーモンタスクが実行されている場合でも、コンテナインスタンスは空であるとみなされます。
2. Amazon ECS では、CapacityProviderReservation 値を 0 ~ 100 の数値に設定します。この数値は、Auto Scaling グループが必要とする規模と実際の規模との比率をパーセンテージで表す次の式を使用します。その後、Amazon ECS は CloudWatch にメトリクスをパブリッシュします。メトリクスの算出方法の詳細については、「Amazon ECS クラスター自動スケーリングの Deep Dive」を参照してください。

   ```
   CapacityProviderReservation = (number of instances needed) / (number of running
    instances) x 100
   ```

3. CapacityProviderReservation メトリクスは CloudWatch アラームを生成します。このアラームは Auto Scaling グループの DesiredCapacity 値を更新します。すると、以下のいずれかのアクションが発生します。
   - キャパシティプロバイダーによるマネージド終了を使用しない場合、Auto Scaling グループは、Auto Scaling グループ終了ポリシーを使用して EC2 インスタンスを選択し、EC2 インスタンス数が DesiredCapacity に達するまでインスタンスを終了します。その後、コンテナインスタンスがクラスターから登録解除されます。
   - すべてのコンテナインスタンスがマネージド型の終了保護を使用している場合、Amazon ECS は空のコンテナインスタンスのスケールイン保護を削除します。Auto Scaling グループは EC2 インスタンスを終了できるようになります。その後、コンテナインスタンスがクラスターから登録解除されます。

Amazon ECS が終了するインスタンスの制御

> **Important**
>
> クラスター自動スケーリングのマネージドターミネーション保護機能を使用するには、Auto Scaling グループで自動スケーリングインスタンスのスケールイン保護を有効にする必要があります。

マネージド終了保護を使用すると、クラスターの自動スケーリングで、どのインスタンスを終了するかを制御できます。マネージド終了保護を使用した場合、Amazon ECS は、実行中の Amazon ECS タスクがない EC2 インスタンスのみを終了します。DAEMON スケジューリング戦略を使用するサービスによって実行されるタスクは無視され、インスタンスがこれらのタスクを実行している場合でも、クラスター自動スケーリングによってインスタンスを終了できます。これは、クラスター内のすべてのインスタンスがこれらのタスクを実行しているためです。

Amazon ECS は最初に Auto Scaling グループの EC2 インスタンスに対するインスタンススケールイン保護オプションを有効にします。次に、Amazon ECS がインスタンスにタスクを配置します。デーモン以外のすべてのタスクがインスタンスで停止すると、Amazon ECS はスケールインプロセスを開始し、EC2 インスタンスのスケールイン保護をオフにします。Auto Scaling グループはインスタンスを終了できます。

自動スケーリングインスタンスのスケールイン保護は、終了できる EC2 インスタンスを制御します。スケールイン機能がオンになっているインスタンスは、スケールインプロセス中に終了できません。自動スケーリングのインスタンススケールイン保護についての詳細は、「Amazon EC2 Auto Scaling ユーザーガイド」の「インスタンススケールイン保護を使用する」を参照してください。

キャパシティに余裕を持たせるように `targetCapacity` の割合を設定できます。こうすると、Auto Scaling グループが起動するインスタンスが増えないため、以降のタスクがより速く起動します。Amazon ECS では、ターゲットキャパシティー値を使用して、サービスによって作成される CloudWatch メトリクスを管理します。Amazon ECS は CloudWatch メトリクスを管理します。Auto Scaling グループが定常状態として扱われるため、スケーリングアクションが必要なくなります。値は 0-100% の範囲で指定できます。例えば、Amazon ECS タスクで使用されるキャパシティーに加えて 10% の空き容量を維持するように Amazon ECS を設定するには、ターゲットキャパシティー値を 90% に設定します。キャパシティプロバイダーで `targetCapacity` 値を設定する際には、次の点を考慮します。

- 100% 未満の `targetCapacity` 値は、クラスター内に必要な空き容量 (Amazon EC2 インスタンス) を表します。空き容量とは、実行中のタスクがないことを意味します。
- アベイラビリティーゾーンなどの配置制約は、追加の `binpack` がなければ、Amazon ECS が最終的にインスタンスごとに 1 つのタスクを実行するように強制しますが、これは望ましい動作ではない可能性があります。

マネージド終了保護を使用するには、Auto Scaling グループで 自動スケーリングインスタンスのスケールイン保護を有効にする必要があります。スケールイン保護を有効にしていない場合、マネージド終了保護を有効にすると、望ましくない動作が発生する可能性があります。例えば、インスタン

スのドレイン状態のままになっている場合を考えてみます。詳細については、「Amazon EC2 Auto Scaling ユーザーガイド」の「インスタンスのスケールイン保護の使用」を参照してください。

キャパシティープロバイダーで終了保護を使用するときは、キャパシティープロバイダーに関連付けられた Auto Scaling グループで、インスタンスのデタッチなどの手動アクションを実行しないでください。手動操作を行うと、キャパシティープロバイダーのスケールイン操作が中断される可能性があります。Auto Scaling グループからインスタンスをデタッチする場合は、Amazon ECS クラスターからもデタッチしたインスタンスを登録解除する必要があります。

Amazon ECS キャパシティープロバイダーのマネージド終了保護の更新

マネージド終了保護を使用する場合は、既存のキャパシティープロバイダーの設定を更新する必要があります。

コンソール

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
2. [クラスター] ページで、クラスターを選択します。
3. クラスターのページで、[インフラストラクチャ] タブを選択します。
4. キャパシティプロバイダーを選択します。
5. [更新] を選択して、キャパシティープロバイダーの設定を変更します。
6. [Auto Scaling グループ設定]で、[マネージド終了保護]を切り替えて機能を有効または無効にします。
7. [更新] を選択します。

AWS CLI

キャパシティープロバイダーのマネージド終了保護設定は、`update-capacity-provider` コマンドを使用して更新できます。

マネージド終了保護を有効にするには:

```
aws ecs update-capacity-provider \
  --name CapacityProviderName \
  --auto-scaling-group-provider
 "managedScaling={status=ENABLED,targetCapacity=70,minimumScalingStepSize=1,maximumScalingStepS
```

マネージド終了保護を無効にするには:

```
aws ecs update-capacity-provider \
  --name CapacityProviderName \
  --auto-scaling-group-provider
 "managedScaling={status=ENABLED,targetCapacity=70,minimumScalingStepSize=1,maximumScalingStepS
```

> **Note**
>
> クラスター全体で変更が有効になるまでに数分かかる場合があります。マネージド終了保護を有効にすると、すでにタスクを実行しているインスタンスはスケールインイベントから保護されます。マネージド終了保護を無効にすると、次の ECS キャパシティープロバイダー管理サイクル中に保護フラグがインスタンスから削除されます。

タスクを実行するコンソール

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
2. [クラスター] ページで、クラスターを選択します。
3. クラスターのページで、[タスク] タブを選択します。
4. タスクを選択します。
5. [設定] で、[マネージド終了保護] を切り替えて機能を有効または無効にします。
6. [タスクのスケールイン保護を設定する] を選択します。

   [タスクのスケールイン保護を設定する] ダイアログボックスが表示されます。

   a. [タスクのスケールイン保護] で、[オンにする] に切り替えます。
   b. [分単位で期限切れになる] に設定する場合、タスクのスケールイン保護が終了するまでの分数を入力します。
   c. [Update] (更新) を選択する

Amazon ECS クラスターの自動スケーリングを有効にする

クラスターの自動スケーリングを有効にして、Amazon ECS がクラスターに登録された Amazon EC2 インスタンスのスケーリングを管理するようにします。

コンソールを使用してクラスターの自動スケーリングを有効にする場合は、「Amazon ECS のキャパシティープロバイダーを作成する」を参照してください。

開始する前に、Auto Scaling グループとキャパシティープロバイダーを作成します。詳細については、「the section called “EC2 ワークロード用の Amazon ECS キャパシティプロバイダー”」を参照してください。

クラスターの自動スケーリングを有効にする場合は、キャパシティープロバイダーをクラスターに関連付けてから、クラスターの自動スケーリングを有効にします。

1. put-cluster-capacity-providers コマンドを使用して、1 つ以上のキャパシティープロバイダーをクラスターに関連付けます。

   AWS Fargate キャパシティプロバイダーを追加するには、リクエストに FARGATE および FARGATE_SPOT キャパシティプロバイダーを入れます。詳細については、AWS CLI コマンドリファレンスの「put-cluster-capacity-providers」を参照してください。

   ```
   aws ecs put-cluster-capacity-providers \
     --cluster ClusterName \
     --capacity-providers CapacityProviderName FARGATE FARGATE_SPOT \
     --default-capacity-provider-strategy capacityProvider=CapacityProvider,weight=1
   ```

   EC2 の Auto Scaling グループを追加するには、リクエストに Auto Scaling グループ名を含めます。詳細については、AWS CLI コマンドリファレンスの「put-cluster-capacity-providers」を参照してください。

   ```
   aws ecs put-cluster-capacity-providers \
     --cluster ClusterName \
     --capacity-providers CapacityProviderName \
     --default-capacity-provider-strategy capacityProvider=CapacityProvider,weight=1
   ```

2. describe-clusters コマンドを使用して、関連付けが成功したことを確認します。詳細については、AWS CLI コマンドリファレンスの「describe-clusters」を参照してください。

   ```
   aws ecs describe-clusters \
     --cluster ClusterName \
     --include ATTACHMENTS
   ```

3. キャパシティープロバイダーのマネージド自動スケーリングを有効にするには、update-capacity-provider コマンドを使用します。詳細については、AWS CLI コマンドリファレンスの「update-capacity-provider」を参照してください。

   ```
   aws ecs update-capacity-provider \
   ```

```
  --name CapacityProviderName \
  --auto-scaling-group-provider "managedScaling={status=ENABLED}"
```

Amazon ECS クラスターの自動スケーリングを無効にする

クラスターに登録されている EC2 インスタンスをよりきめ細かく制御する必要がある場合は、クラスターの自動スケーリングを無効にします。

クラスターの自動スケーリングを無効にするには、キャパシティープロバイダーとクラスターから有効にしたマネージドスケーリングの関連付けを解除するか、キャパシティープロバイダーを更新してマネージドスケーリングを無効にします。

キャパシティープロバイダーの関連付けを解除する

キャパシティープロバイダーとクラスターとの関連付けを解除するには、次の手順を実行します。

1. put-cluster-capacity-providers コマンドを使用して、Auto Scaling グループのキャパシティープロバイダーとクラスターとの関連付けを解除します。クラスターは、AWS Fargate キャパシティープロバイダーとの関連付けを保持できます。詳細については、AWS CLI コマンドリファレンスの「put-cluster-capacity-providers」を参照してください。

   ```
   aws ecs put-cluster-capacity-providers \
     --cluster ClusterName \
     --capacity-providers FARGATE FARGATE_SPOT \
     --default-capacity-provider-strategy '[]'
   ```

   put-cluster-capacity-providers コマンドを使用して、Auto Scaling グループのキャパシティープロバイダーとクラスターとの関連付けを解除します。詳細については、AWS CLI コマンドリファレンスの「put-cluster-capacity-providers」を参照してください。

   ```
   aws ecs put-cluster-capacity-providers \
     --cluster ClusterName \
     --capacity-providers [] \
     --default-capacity-provider-strategy '[]'
   ```

2. describe-clusters コマンドを使用して、関連付けの解除が成功したことを確認します。詳細については、AWS CLI コマンドリファレンスの「describe-clusters」を参照してください。

   ```
   aws ecs describe-clusters \
   ```

```
  --cluster ClusterName \
  --include ATTACHMENTS
```

キャパシティープロバイダーのマネージドスケーリングを無効にする

キャパシティープロバイダーのマネージドスケーリングを無効にするには、次のステップに従います。

- キャパシティープロバイダーのマネージド自動スケーリングを無効にするには、update-capacity-provider コマンドを使用します。詳細については、AWS CLI コマンドリファレンスの「update-capacity-provider」を参照してください。

```
aws ecs update-capacity-provider \
  --name CapacityProviderName \
  --auto-scaling-group-provider "managedScaling={status=DISABLED}"
```

Amazon ECS のキャパシティープロバイダーを作成する

クラスターの作成が完了したら、EC2 の新しいキャパシティプロバイダー (Auto Scaling グループ) を作成できます。キャパシティープロバイダーは、アプリケーションのインフラストラクチャの管理とスケーリングに役立ちます。

キャパシティプロバイダーを作成する前に、Auto Scaling グループを作成する必要があります。詳細については、「Amazon EC2 Auto Scaling ユーザーガイド」の「Auto Scaling グループ」を参照してください。

クラスターのキャパシティプロバイダーを作成するには (Amazon ECS コンソール)

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。
3. [Clusters] (クラスター) ページで、クラスターを選択します。
4. [Cluster : *name*] (クラスター : 名) ページで、[Infrastructure] (インフラストラクチャ) を選択し、[Create] (作成) を選択します。
5. [Create capacity providers] (キャパシティプロバイダーの作成) ページで、次のオプションを設定します。

a. [Basic details] (基本的な詳細) の下の [Capacity provider name] (キャパシティプロバイダー名) に、一意のキャパシティプロバイダー名を入力します。

b. [Auto Scaling group] (Auto Scaling グループ) の [Use an existing Auto Scaling group] (既存の Auto Scaling グループを使用する) で、Auto Scaling グループを選択します。

c. (オプション) スケーリングポリシーを設定するには、[Scaling policies] (スケーリングポリシー) で次のオプションを設定します。

- スケールインおよびスケールアウトアクションを Amazon ECS に管理させるには、[Turn on managed scaling] (マネージドスケーリングをオンにする) を選択します。
- Amazon ECS タスクが実行されている EC2 インスタンスが終了しないようにするには、[Turn on scaling protection] (スケーリング保護を有効にする) を選択します。
- [Set target capacity] (ターゲットキャパシティの設定) に、Amazon ECS マネージド対象ターゲット追跡スケーリングポリシーで使用される CloudWatch メトリクスのターゲット値を入力します。

6. [作成] を選択します。

Amazon ECS キャパシティープロバイダーを更新する

Auto Scaling グループをキャパシティプロバイダーとして使用する場合、グループのスケーリングポリシーを変更できます。

クラスターのキャパシティプロバイダーを更新するには (Amazon ECS コンソール)

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。
3. [Clusters] (クラスター) ページで、クラスターを選択します。
4. [Cluster : ***name***] (クラスター : 名) ページで、[Infrastructure] (インフラストラクチャ) を選択し、[Update] (更新) を選択します。
5. [Create capacity providers] (キャパシティプロバイダーの作成) ページで、次のオプションを設定します。

- [Auto Scaling グループ] の [スケーリングポリシー] で、以下のオプションを設定します。
  - スケールインおよびスケールアウトアクションを Amazon ECS に管理させるには、[Turn on managed scaling] (マネージドスケーリングをオンにする) を選択します。

- Amazon ECS タスクが実行されている EC2 インスタンスが終了しないようにするには、[スケーリング保護をオンにする] を選択します。
- [Set target capacity] (ターゲットキャパシティの設定) に、Amazon ECS マネージド対象ターゲット追跡スケーリングポリシーで使用される CloudWatch メトリクスのターゲット値を入力します。

6. [更新] を選択します。

Amazon ECS キャパシティープロバイダーを削除する

Auto Scaling グループキャパシティープロバイダーは、使い終わったら削除できます。グループが削除されると、Auto Scaling グループのキャパシティープロバイダーは `INACTIVE` 状態に移行します。`INACTIVE` ステータスのキャパシティープロバイダーは、一定期間アカウント内で検出可能になる場合があります。ただし、この動作は今後変更される予定であり、`INACTIVE` のキャパシティープロバイダーを永続的に使用するのは避けてください。Auto Scaling グループキャパシティプロバイダーを削除する前に、すべてのサービスのキャパシティプロバイダー戦略からキャパシティプロバイダーを削除する必要があります。サービスのキャパシティプロバイダー戦略からキャパシティプロバイダーを削除するには、`UpdateService` API または Amazon ECS コンソールのサービス更新ワークフローを使用できます。[新しいデプロイの強制] オプションを使用すると、キャパシティプロバイダーが提供する Amazon EC2 インスタンスキャパシティを使用している任意のタスクを、残りのキャパシティプロバイダーのキャパシティを使用するように移行できます。

クラスターのキャパシティプロバイダーを削除するには (Amazon ECS コンソール)

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。
3. [Clusters] (クラスター) ページで、クラスターを選択します。
4. [Cluster : ***name***] (クラスター : 名) ページで、[Infrastructure] (インフラストラクチャ) を選択し、Auto Scaling グループを選択して、[Delete] (削除) を選択します。
5. 確認ボックスに、「delete ***Auto Scaling #######***」と入力します。
6. [削除] を選択します。

EC2 インスタンスで実行されている Amazon ECS ワークロードを安全に停止する

マネージドインスタンスドレイニングを使用すると、Amazon EC2 インスタンスの正常な終了が容易になります。この結果、ワークロードを安全に停止し、終了しないインスタンスに再スケジュール

できます。インフラストラクチャのメンテナンスと更新は、ワークロードの中断を心配することなく実行できます。マネージドインスタンスドレイニングを使用することで、Amazon EC2 インスタンスの置換を必要とするインフラストラクチャ管理ワークフローを簡素化しつつ、アプリケーションの耐障害性と可用性を確保できます。

Amazon ECS マネージドインスタンスドレイニングは、Auto Scaling グループのインスタンス置換と連動します。インスタンスの更新とインスタンスの最大有効期間に基づいて、お客様は最新の OS およびキャパシティに関するセキュリティ要件に常に準拠できます。

マネージドインスタンスドレイニングは、Amazon ECS キャパシティープロバイダーでのみ使用できます。Amazon ECS コンソール、AWS CLI、または SDK を使用して Auto Scaling グループのキャパシティープロバイダーを作成または更新するときに、マネージドインスタンスドレイニングを有効にできます。

以下のイベントは Amazon ECS マネージドインスタンスドレイニングの対象となります。

- Auto Scaling グループのインスタンス更新 - インスタンス更新を使用すると、バッチによる手動ではなく、Auto Scaling グループ内の Amazon EC2 インスタンスのローリング置換を実行できます。これは、多数のインスタンスを置換する必要がある場合に便利です。インスタンス更新は、Amazon EC2 コンソールまたは `StartInstanceRefresh` API により開始されます。マネージドターミネーション保護を使用している場合は、`StartInstanceRefresh` を呼び出す際に必ず `Replace` を選択してスケールイン保護をしてください。
- インスタンスの最大有効期間 - Auto Scaling グループインスタンスの置換に至るまでの最大有効期間を定義できます。これは、内部のセキュリティポリシーやコンプライアンスに基づいて置換インスタンスをスケジュールするのに役立ちます。
- Auto Scaling グループのスケールイン - スケーリングポリシーとスケジュールされたスケーリングアクションに基づいて、Auto Scaling グループはインスタンスの自動スケーリングをサポートします。Amazon ECS キャパシティープロバイダーとして Auto Scaling グループを使用することで、タスクが実行されていないときに Auto Scaling グループのインスタンスをスケールインすることができます。
- Auto Scaling グループのヘルスチェック - Auto Scaling グループは、異常のあるインスタンスの終了を管理するための多くのヘルスチェックをサポートしています。
- CloudFormation スタックの更新 - CloudFormation スタックに `UpdatePolicy` 属性を追加することで、グループが変更されたときにローリング更新を実行できます。
- スポットキャパシティーの再調整 - Auto Scaling グループは、Amazon EC2 のキャパシティー再調整通知に基づいて、中断のリスクが高いスポットインスタンスを事前対応的に置換しようとします。Auto Scaling グループは、置換インスタンスが起動して正常になると、古いインスタンスを終

了します。Amazon ECS マネージドインスタンスドレイニングでは、非スポットインスタンスをドレインするのと同じ方法でスポットインスタンスをドレインします。

- スポット中断 - スポットインスタンスは 2 分間の通知の後に終了します。Amazon ECS マネージドインスタンスドレイニングでは、それに応じてインスタンスがドレイニング状態になります。

マネージドインスタンスドレイニングがある Amazon EC2 Auto Scaling のライフサイクルフック

Auto Scaling グループのライフサイクルフックを使用すると、お客様はインスタンスライフサイクルの特定のイベントによってトリガーされるソリューションを作成し、その特定のイベントが発生したときにカスタムアクションを実行できます。Auto Scaling グループでは、最大 50 個のフックを使用できます。複数の終了フックが存在する場合があり、それらは同時に実行されるため、Auto Scaling グループはすべてのフックが終了するのを待ってからインスタンスを終了します。

Amazon ECS マネージド型のフック終了に加えて、独自のライフサイクル終了フックを設定することもできます。ライフサイクルフックには `default action` があります。Amazon ECS マネージドフックなどの他のフックがカスタムフックによるエラーの影響を受けないようにするために、`continue` をデフォルトとして設定することをお勧めします。

Auto Scaling グループの終了ライフサイクルフックを既に設定していて、Amazon ECS マネージドインスタンスドレイニングも有効にしている場合、両方のライフサイクルフックが実行されます。ただし、相対的なタイミングは保証されません。ライフサイクルフックには、タイムアウトが終了したときに実行するアクションを指定する `default action` 設定があります。障害が発生した場合は、`continue` をカスタムフックのデフォルト結果として使用することをお勧めします。このようにすることで、他のフック、特に Amazon ECS マネージドフックは、カスタムライフサイクルフックのエラーの影響を受けなくなります。`abandon` による別の結果では、ほかのすべてのフックがスキップされるので、避けるべきです。Auto Scaling グループのライフサイクルフックの詳細については、「Amazon EC2 Auto Scaling ユーザーガイド」の「Amazon EC2 Auto Scaling のライフサイクルフック」を参照してください。

タスクとマネージドインスタンスドレイニング

Amazon ECS マネージドインスタンスドレイニングでは、コンテナインスタンスにある既存のドレイニング機能を使用します。コンテナインスタンスドレイニング機能は、Amazon ECS サービスに属するレプリカタスクの置換と停止を実行します。スタンドアロンタスク (`RunTask` によって呼び出されたタスクなど) は、`PENDING` または `RUNNING` の状態にある場合には影響を受けません。そのようなタスクは、完了するまで待つか、手動で停止する必要があります。コンテナインスタンスは、すべてのタスクが停止されるか、48 時間が経過するまで `DRAINING` 状態のままになります。デーモンタスクは、すべてのレプリカタスクが停止した後に、最後に停止します。

マネージドインスタンスドレイニングとマネージドターミネーション保護

マネージドインスタンスドレイニングは、マネージド終了が無効になっている場合でも機能します。マネージド終了保護の詳細については、「Amazon ECS が終了するインスタンスの制御」を参照してください。

次の表は、マネージドターミネーションとマネージドドドレイニングのさまざまな組み合わせの動作をまとめたものです。

マネージドターミネーション	マネージドドドレイニング	結果
有効	有効	Amazon ECS は、タスクを実行している Amazon EC2 インスタンスがスケールインイベントによって終了するのを防ぎます。終了保護が設定されていないインスタンス、スポット中断を受けたインスタンス、インスタンス更新によって強制されたインスタンスなど、終了中のインス

マネージドターミネーション	マネージドドレイニング	結果
		タンスがすべて正常にドレインされます。
無効	有効	Amazon ECS は、タスクを実行している Amazon EC2 インスタンスがスケールインされないように保護することはしません。ただし、終了されるインスタンスはすべて正常にドレインされます。

マネージドターミネーション	マネージドドレイニング	結果
有効	無効	Amazon ECS は、タスクを実行している Amazon EC2 インスタンスがスケールインイベントによって終了するのを防ぎます。ただし、スポット中断や強制的なインスタンス更新によって、またはタスクがまったく実行されていない場合に、インスタンスは終了する可能性があります。Amazon ECS はこれらのインスタンスに対して正常なドレインを実行せ

マネージドターミネーション	マネージドドレイニング	結果
		ず、停止後に置換サービスタスクを起動します。
無効	無効	Amazon EC2 インスタンスは、Amazon ECS タスクを実行している場合でも、いつでもスケールインまたは終了できます。Amazon ECS は、インスタンスが停止した後に置換サービスタスクを起動します。

マネージドインスタンスドレイニングとスポットインスタンスドレイニング

スポットインスタンスドレイニングでは、Amazon ECS エージェントに環境変数 `ECS_ENABLE_SPOT_INSTANCE_DRAINING` を設定できます。これにより、Amazon ECS は 2 分間のスポット中断に応答して、インスタンスをドレイン中ステータスに移行できます。Amazon ECS マネージドインスタンスドレイニングを使用すると、スポット中断だけでなく、さまざまな理由で

終了中の Amazon EC2 インスタンスを正常にシャットダウンできます。例えば、Amazon EC2 Auto Scaling のキャパシティー再調整を使用して、中断のリスクが高まっているスポットインスタンスを事前対応的に置換できます。そして、マネージドインスタンスドレイニングにより、置換対象のスポットインスタンスの正常シャットダウンが実行されます。マネージドインスタンスドレイニングを使用する場合、スポットインスタンスドレイニングを個別に有効にする必要がないため、Auto Scaling グループのユーザーデータの `ECS_ENABLE_SPOT_INSTANCE_DRAINING` が冗長になります。スポットインスタンスドレイニングの詳細については、「スポットインスタンス」を参照してください。

マネージドインスタンスドレイニングと EventBridge の連携

Amazon ECS マネージドインスタンスドレイニングイベントは Amazon EventBridge に発行され、Amazon ECS はマネージドインスタンスドレイニングをサポートするために、アカウントのデフォルトバスに EventBridge マネージドルールを作成します。これらのイベントを Lambda、Amazon SNS、および Amazon SQS などのほかの AWS サービスにフィルタリングして、モニタリングおよびトラブルシューティングできます。

- Amazon EC2 Auto Scaling は、ライフサイクルフックを呼び出すときに EventBridge にイベントを送信します。
- スポット中断通知は EventBridge に発行されます。
- Amazon ECS は、エラーメッセージを生成します。このエラーメッセージは、Amazon ECS コンソールおよび API を介して取得することができます。
- EventBridge には、一時的な障害を軽減するための再試行メカニズムが組み込まれています。

インスタンスを安全にシャットダウンするように Amazon ECS キャパシティープロバイダーを設定する

Amazon ECS コンソールと AWS CLI を使用して Auto Scaling グループのキャパシティープロバイダーを作成または更新するときに、マネージドインスタンスドレイニングを有効にできます。

> **Note**
>
> キャパシティープロバイダーを作成すると、マネージドインスタンスドレイニングはデフォルトで有効になります。

以下に、AWS CLI を使用してマネージドインスタンスドレイニングを有効にしたキャパシティプロバイダーを作成し、クラスターの既存のキャパシティプロバイダーのマネージドインスタンスドレイニングを有効にする例を示します。

マネージドインスタンスドレイニングを有効にしたキャパシティプロバイダーを作成する

マネージドインスタンスドレイニングを有効にしたキャパシティプロバイダーを作成するには、`create-capacity-provider` コマンドを使用します。`managedDraining` パラメータを `ENABLED` に設定します。

```
aws ecs create-capacity-provider \
--name capacity-provider \
--auto-scaling-group-provider '{
  "autoScalingGroupArn": "asg-arn",
  "managedScaling": {
    "status": "ENABLED",
    "targetCapacity": 100,
    "minimumScalingStepSize": 1,
    "maximumScalingStepSize": 1
  },
  "managedDraining": "ENABLED",
  "managedTerminationProtection": "ENABLED",
}'
```

レスポンス:

```
{
    "capacityProvider": {
        "capacityProviderArn": "capacity-provider-arn",
        "name": "capacity-provider",
        "status": "ACTIVE",
        "autoScalingGroupProvider": {
            "autoScalingGroupArn": "asg-arn",
            "managedScaling": {
                "status": "ENABLED",
                "targetCapacity": 100,
                "minimumScalingStepSize": 1,
                "maximumScalingStepSize": 1
            },
            "managedTerminationProtection": "ENABLED"
            "managedDraining": "ENABLED"
        }
```

```
    }
}
```

クラスターの既存のキャパシティプロバイダーに対してマネージドインスタンスドレイニングを有効にします。

update-capacity-provider コマンドを使用して、クラスターの既存のキャパシティプロバイダーに対してマネージドインスタンスドレイニングを有効にします。現在 managedDraining が DISABLED で、updateStatus が UPDATE_IN_PROGRESS になっているのがわかります。

```
aws ecs update-capacity-provider \
--name cp-draining \
--auto-scaling-group-provider '{
  "managedDraining": "ENABLED"
}
```

レスポンス:

```
{
    "capacityProvider": {
        "capacityProviderArn": "cp-draining-arn",
        "name": "cp-draining",
        "status": "ACTIVE",
        "autoScalingGroupProvider": {
            "autoScalingGroupArn": "asg-draining-arn",
            "managedScaling": {
                "status": "ENABLED",
                "targetCapacity": 100,
                "minimumScalingStepSize": 1,
                "maximumScalingStepSize": 1,
                "instanceWarmupPeriod": 300
            },
            "managedTerminationProtection": "DISABLED",
            "managedDraining": "DISABLED" // before update
        },
        "updateStatus": "UPDATE_IN_PROGRESS", // in progress and need describe again to
 find out the result
        "tags": [
        ]
    }
}
```

describe-clusters コマンドを使用して ATTACHMENTS を含めます。マネージドインスタンスドレイニングアタッチメントの status は PRECREATED で、全体の attachmentsStatus は UPDATING です。

```
aws ecs describe-clusters --clusters cluster-name --include ATTACHMENTS
```

レスポンス:

```
{
    "clusters": [
        {
            ...

            "capacityProviders": [
                "cp-draining"
            ],
            "defaultCapacityProviderStrategy": [],
            "attachments": [
                # new precreated managed draining attachment
                {
                    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
                    "type": "managed_draining",
                    "status": "PRECREATED",
                    "details": [
                        {
                            "name": "capacityProviderName",
                            "value": "cp-draining"
                        },
                        {
                            "name": "autoScalingLifecycleHookName",
                            "value": "ecs-managed-draining-termination-hook"
                        }
                    ]
                },

                ...

            ],
            "attachmentsStatus": "UPDATING"
        }
    ],
    "failures": []
```

```
}
```

更新が終了したら、describe-capacity-providers を使用します。すると、managedDraining が ENABLED になったのがわかります。

```
aws ecs describe-capacity-providers --capacity-providers cp-draining
```

レスポンス:

```
{
    "capacityProviders": [
        {
            "capacityProviderArn": "cp-draining-arn",
            "name": "cp-draining",
            "status": "ACTIVE",
            "autoScalingGroupProvider": {
                "autoScalingGroupArn": "asg-draning-arn",
                "managedScaling": {
                    "status": "ENABLED",
                    "targetCapacity": 100,
                    "minimumScalingStepSize": 1,
                    "maximumScalingStepSize": 1,
                    "instanceWarmupPeriod": 300
                },
                "managedTerminationProtection": "DISABLED",
                "managedDraining": "ENABLED" // successfully update
            },
            "updateStatus": "UPDATE_COMPLETE",
            "tags": []
        }
    ]
}
```

Amazon ECS マネージドインスタンスドレイニングのトラブルシューティング

マネージドインスタンスドレイニングに関する問題のトラブルシューティングが必要になる場合があります。以下は、使用中に発生する可能性のある問題と解決方法の例です。

自動スケーリングの使用時に、最大インスタンス有効期間を超えてもインスタンスが終了しません。

Auto Scaling グループの使用中に最大インスタンス有効期間に到達および超過した後もインスタンスが終了しない場合、スケールインから保護されていることが原因である可能性があります。マネージ

ド終了を無効にし、マネージドドレイニングがインスタンスのリサイクルを処理できるようにすることができます。

Amazon ECS マネージドインスタンスのドレイニング動作

Amazon ECS マネージドインスタンスの終了は、コストを最適化し、システムの健全性を維持しながら、ワークロードを正常に移行します。終了システムには、インスタンスの終了に関する 3 つの異なる決定パスがあり、それぞれのタイミングの特性やお客様に影響する事項が異なります。

終了決定パス

お客様が開始した終了

コンテナインスタンスをすぐにサービスから削除する必要がある場合に、インスタンスの削除を直接制御できます。強制フラグを true に設定して DeregisterContainerInstance API を呼び出します。これは、ワークロードが実行中にもかかわらず即時終了が必要であることを示します。

システムによるアイドル終了

Amazon ECS マネージドインスタンスは、タスクを実行していないアイドル状態の Amazon ECS コンテナインスタンスを終了することで、コストを継続的にモニタリングし、プロアクティブに最適化します。ECS はヒューリスティック遅延を使用して、コンテナインスタンスが終了する前に新しく起動されたタスクを取得する時間を提供します。これは、`scaleInAfter` Amazon ECS マネージドインスタンスのキャパシティープロバイダー設定パラメータを使用してカスタマイズできます。

インフラストラクチャの更新の終了

Amazon ECS マネージドインスタンスでは、ワークロードの可用性を維持しながら、マネージドコンテナインスタンスのソフトウェアを自動的に管理および更新することで、セキュリティとコンプライアンスを確実にします。詳細については、Amazon ECS マネージドインスタンスでのパッチ適用を参照してください。

正常なドレイニングとワークロードの移行

正常なドレイニングシステムは、Amazon ECS サービス管理との高度な調整を実装し、サービスマネージドタスクが、終了予定のインスタンスから適切に移行されるようにします。

サービスタスクドレイニングの調整

インスタンスが DRAINING 状態に移行すると、Amazon ECS スケジューラは、既存のサービスタスクに正常なシャットダウン手順を実装しながら、インスタンスへの新しいタスクの配置を自動的に

停止します。サービスタスクドレイニングには、最適な移行タイミングと成功率を確保するために、サービスデプロイ戦略、ヘルスチェック要件、ドレイニング設定の調整が含まれます。

スタンドアロンタスク処理

スタンドアロンタスクは、自動サービス管理の恩恵を受けないため、さまざまな処理が必要です。システムは、スタンドアロンのタスク特性 (タスク期間の見積もり、完了確率分析、顧客への影響評価など) を評価します。正常な完了戦略により、スタンドアロンタスクは猶予期間の延長中に自然に完了できます。また、強制終了により、タスクが自然に完了していない場合、許容期間内にインフラストラクチャの更新を実行するようになります。

2 フェーズ完了戦略

終了システムは、ワークロードの継続性とインフラストラクチャ管理要件のバランスを取る、2 段階のアプローチを実行します。

フェーズ 1: 正常な完了期間

このフェーズでは、システムはワークロードの継続性を優先する、正常なドレイニング戦略を実行します。サービスタスクは通常の Amazon ECS スケジューリングプロセスによって正常にドレインされます。スタンドアロンタスクは実行を継続しますが、自然に完了する可能性があります。システムはすべてのタスクが自然な完了プロセスを通じて停止状態に到達しているかどうかをモニタリングします。

フェーズ 2: 厳格な期限の適用

正常な完了が、許容可能な期間内に終了目標を達成しない場合、システムは厳格な期限の適用を実行します。厳格な期限では、通常ドレイン開始日時に 7 日を加えて設定されるため、運用要件は維持されますが、正常な完了にはかなりの時間がかかります。強制には、強制登録解除手順の自動呼び出しと、完了ステータスを考慮しない残りのすべてのタスクの即時終了が含まれます。

AWS マネジメントコンソール を使用した Amazon ECS クラスターの自動スケーリング用のリソースの作成

AWS マネジメントコンソール を使用してクラスターの自動スケーリング用のリソースを作成する方法について説明します。リソースに名前が必要な場合は、プレフィックス `ConsoleTutorial` を使用して、すべてのリソースに一意の名前があることを確認し、見つけやすくします。

トピック

- 前提条件
- ステップ 1: Amazon ECS クラスターを作成する

- ステップ 2: タスク定義を登録する
- ステップ 3: タスクを実行する
- ステップ 4: 確認する
- ステップ 5: クリーンアップ

前提条件

このチュートリアルでは、以下の前提条件が完了済みであることを前提としています。

- 「Amazon ECS を使用するようにセットアップする」のステップを完了していること。
- IAM ユーザーに AmazonECS_FullAccess IAM ポリシー例で指定されている必要なアクセス権限があること。
- Amazon ECS コンテナインスタンス IAM ロールが作成されます。詳細については、「Amazon ECS コンテナインスタンスの IAM ロール」を参照してください。
- Amazon ECS サービスにリンクされた IAM ロールが作成されます。詳細については、「Amazon ECS のサービスリンクロールの使用」を参照してください。
- 自動スケーリングのサービスにリンクされたIAMロールを作成する 詳細については、Amazon EC2 Auto Scaling ユーザーガイドの「Amazon EC2 Auto Scaling のサービスにリンクされたロール」を参照してください。
- VPC およびセキュリティグループが使用できるように作成されていること。詳細については、「the section called “仮想プライベートクラウドを作成する”」を参照してください。

ステップ 1: Amazon ECS クラスターを作成する

次の手順に従って Amazon ECS クラスターを作成します。

Amazon ECS は、CloudFormation スタックの一部としてユーザーに代わり Amazon EC2 Auto Scaling 起動テンプレートと Auto Scaling グループを作成します。

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで、[クラスター] を選択し、[クラスターの作成] を選択します。
3. [クラスター設定] の [クラスター名] に、「`ConsoleTutorial-cluster`」と入力します。
4. [インフラストラクチャ] で、[AWS Fargate (サーバーレス)] を選択解除し、[Amazon EC2 インスタンス] を選択します。次に、キャパシティプロバイダーとして動作する、Auto Scaling グループを設定します。

- [自動スケーリンググループ (ASG)] の下です。[新しい ASG を作成] を選択し、次に、そのグループに関する詳細を以下のように入力します。
  - [オペレーティングシステム/アーキテクチャ] で、[Amazon Linux 2] を選択します。
  - [EC2 インスタンスタイプ] で [t3.nano] を選択します。
  - [Capacity] (キャパシティー) には、Auto Scaling グループで起動するインスタンスの最小数と最大数を入力します。

5. (オプション) クラスタータグを管理するには、[Tags] (タグ) を展開し、次のいずれかのオペレーションを実行します。

   [タグの追加] [タグの追加] を選択して、以下を実行します。

   - [キー] にはキー名を入力します。
   - [値] にキー値を入力します。

   [タグを削除] タグのキーと値の右側にある [削除] を選択します。

6. [作成] を選択します。

ステップ 2: タスク定義を登録する

クラスターでタスクを実行する前に、タスク定義を登録する必要があります。タスク定義とは、1 つにグループ化されたコンテナのリストです。次の例は、Docker Hub から取得した `amazonlinux` イメージを使用し、スリープ状態になるシンプルなタスク定義です。使用できるタスク定義パラメータの詳細については、「Amazon ECSの タスク定義」を参照してください。

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで、タスク定義 を選択します。
3. [Create new task definition] (新しいタスク定義の作成)、[Create new task definition with JSON] (JSON で新しいタスク定義を作成) の順に選択します。
4. [JSON エディタ] ボックスには、以下の内容を貼り付けます。

```
{
    "family": "ConsoleTutorial-taskdef",
    "containerDefinitions": [
        {
            "name": "sleep",
```

```
            "image": "public.ecr.aws/amazonlinux/amazonlinux:latest",
            "memory": 20,
            "essential": true,
            "command": [
                "sh",
                "-c",
                "sleep infinity"
            ]
        }
    ],
    "requiresCompatibilities": [
        "EC2"
    ]
}
```

5. [作成] を選択します。

ステップ 3: タスクを実行する

アカウントのタスク定義を登録したら、クラスターでタスクを実行できます。このチュートリアルでは、ConsoleTutorial-taskdef クラスターで ConsoleTutorial-cluster タスク定義のインスタンスを 5 つ実行します。

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. [クラスター] ページで、[コンソールチュートリアル – クラスター] を選択します。
3. [タスク] で、[新しいタスクの実行] を選択します
4. [環境] セクションの [コンピューティングオプション] で、[キャパシティープロバイダー戦略] を選択します。
5. [デプロイメント構成] の [アプリケーション タイプ] で、[タスク] を選択します。
6. [ファミリー] ドロップダウン リストから [ConsoleTutorial-taskdef] を選択します。
7. [希望するタスク] で、「5」と入力します。
8. [作成] を選択します。

ステップ 4: 確認する

チュートリアルのこの時点で、5 つのタスクが実行されているクラスターと、キャパシティープロバイダーを備えた Auto Scaling グループができているはずです。キャパシティープロバイダーが Amazon ECS マネージドスケーリングを有効にしています。

CloudWatch メトリクス、Auto Scaling グループ設定、最後に Amazon ECS クラスタータスク数を表示することで、すべてが正常に機能していることを確認できます。

クラスターの CloudWatch メトリクスを表示するには

1. CloudWatch コンソールの https://console.aws.amazon.com/cloudwatch/ を開いてください。
2. 画面上部のナビゲーションバーで、 リージョンを選択します。
3. ナビゲーションペインで、[メトリクス] から [すべてのメトリクス] を選択します。
4. [すべてのメトリクス] ページの [参照] タブで、`AWS/ECS/ManagedScaling` を選択します。
5. [CapacityProviderName, ClusterName] を選択します。
6. `ConsoleTutorial-cluster` [クラスター名] に対応するチェックボックスを選択します。
7. [グラフ化メトリクス] タブで、[期間] を [30 秒]、[統計] を [最大] に変更します。

   グラフに表示される値は、キャパシティープロバイダーのターゲットキャパシティー値を示します。これは、設定したターゲットキャパシティーパーセントである `100` から開始する必要があります。`200` までスケールアップすると、ターゲット追跡スケーリングポリシーのアラームがトリガーされます。その後、アラームが発生し Auto Scaling グループがスケールアウトします。

次のステップに従って、Auto Scaling グループの詳細を表示し、スケールアウトアクションが発生したことを確認します。

Auto Scaling グループがスケールアウトされたことを確認するには

1. Amazon EC2 コンソールの https://console.aws.amazon.com/ec2/ を開いてください。
2. 画面上部のナビゲーションバーで、 リージョンを選択します。
3. ナビゲーションペインの 自動スケーリング で、[Auto Scaling Groups] (Auto Scaling グループ) を選択します。
4. このチュートリアルで作成した `ConsoleTutorial-cluster` Auto Scaling グループを選択します。[必要なキャパシティー] の値を表示し、[インスタンス管理] タブでインスタンスを表示して、グループが 2 つのインスタンスにスケールアウトされていることを確認します。

次のステップを使用して Amazon ECS クラスターを表示し、Amazon EC2 インスタンスがクラスターに登録され、タスクが `RUNNING` ステータスに移行したことを確認します。

Auto Scaling グループに含まれるインスタンスを確認する方法

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。
3. [クラスター] ページで、`ConsoleTutorial-cluster` クラスターを選択します。
4. [タスク] タブで、5 つのタスクが `RUNNING` ステータスになっていることを確認します。

ステップ 5: クリーンアップ

このチュートリアルが終了したら、使用していないリソースに対する料金が発生しないように、チュートリアルに関連付けられたリソースをクリーンアップします。キャパシティープロバイダーとタスク定義の削除はサポートされていませんが、これらのリソースに関連するコストは発生しません。

チュートリアルリソースをクリーンアップするには

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。
3. [クラスター] ページで、[コンソールチュートリアル – クラスター] を選択します。
4. [ConsoleTutorial-cluster] ページで、[タスク] タブを選択し、[停止]、[すべて停止] の順に選択します。
5. ナビゲーションペインで [Clusters] (クラスター) を選択します。
6. [クラスター] ページで、[コンソールチュートリアル – クラスター] を選択します。
7. [クラスターの削除] を選択します。
8. 確認ボックスに [ConsoleTutorial-cluster を削除] と入力し、[削除] を選択します。
9. 以下のステップに従って Auto Scaling グループを削除します。
    a. Amazon EC2 コンソールの https://console.aws.amazon.com/ec2/ を開いてください。
    b. 画面上部のナビゲーションバーで、 リージョンを選択します。
    c. ナビゲーションペインの 自動スケーリング で、[Auto Scaling Groups] (Auto Scaling グループ) を選択します。
    d. `ConsoleTutorial-cluster` Auto Scaling グループ > [アクション] の順に選択します。
    e. [アクション] メニューから、[削除] を選択します。確認ボックスに [削除] と入力し、[削除] を選択します。

Amazon ECS 用の Amazon EC2 コンテナインスタンス

Amazon ECS コンテナインスタンスは、Amazon ECS コンテナエージェントを実行し、クラスターに登録されている Amazon EC2 インスタンスです。キャパシティプロバイダー、外部キャパシティプロバイダー、または Auto Scaling グループキャパシティプロバイダーを使用して Amazon ECS でタスクを実行すると、タスクはアクティブなコンテナインスタンス上に配置されます。コンテナインスタンスの管理とメンテナンスはお客様の責任となります。

Amazon ECS で一元化されたワークロードを実行するために必要な基本的な仕様を満たす独自の Amazon EC2 インスタンス AMI を作成することはできますが、Amazon ECS に最適化された AMI は事前設定され、AWS エンジニアにより Amazon ECS でテストされています。これは最も簡単に開始できる方法であり、AWS でコンピューティングリソースをすばやく実行できます。

コンソールを使用してクラスターを作成すると、Amazon ECS は選択したオペレーティングシステムに関連付けられた最新の AMI を使用してインスタンスの起動テンプレートを作成します。

CloudFormation を使用してクラスターを作成する場合、SSM パラメータは Auto Scaling グループインスタンスの Amazon EC2 起動テンプレートの一部となります。動的な Systems Manager パラメータを使用して、デプロイする Amazon ECS Optimized AMI を決定するようにテンプレートを設定できます。このパラメータにより、スタックをデプロイするたびに、EC2 インスタンスに適用する必要がある利用可能な更新があるかどうかがチェックされます。Systems Manager パラメータの使用方法の例については、「AWS CloudFormation ユーザーガイド」の「Amazon ECS に最適化された Amazon Linux 2023 AMI を使用して Amazon ECS クラスターを作成する」を参照してください。

- Amazon ECS に最適化された Linux AMI メタデータを取得する
- Amazon ECS に最適化された Bottlerocket AMI メタデータを取得する
- Amazon ECS に最適化された Windows AMI メタデータを取得する

アプリケーションと互換性のあるインスタンスタイプから選択できます。大きいインスタンスでは、同時に多くのタスクを起動できます。インスタンスが小さい場合は、よりきめ細かくスケールアウトしてコストを節約できます。クラスター内のすべてのアプリケーションに対応する Amazon EC2 インスタンスタイプを 1 つ選択する必要はありません。代わりに、複数の Auto Scaling グループを作成し、各グループが異なるインスタンスタイプを持つこともできます。次に、これらのグループごとに Amazon EC2 キャパシティープロバイダーを作成できます。

使用するインスタンスファミリータイプとインスタンスタイプを決定するには、以下のガイドラインを使用します。

- アプリケーションの特定の要件を満たしていないインスタンスタイプまたはインスタンスファミリーを除外します。例えば、アプリケーションに GPU が必要な場合は、GPU がないインスタンスタイプをすべて除外できます。
- ネットワークスループットやストレージなどの要件を検討します。
- CPU とメモリを検討します。原則として、CPU とメモリは、実行するタスクのレプリカを少なくとも 1 つ保存できる大きさである必要があります。

スポットインスタンス

Spot キャパシティーは、オンデマンドインスタンスに比べて大幅なコスト削減が可能です。Spot キャパシティーとは、オンデマンドまたはリザーブドキャパシティーよりも大幅に低価格の余剰容量です。Spot キャパシティーは、バッチ処理や機械学習のワークロード、開発環境やステージング環境に適しています。より一般的には、一時的なダウンタイムを許容するあらゆるワークロードに適しています。

Spot キャパシティーが常に利用できるわけではないため、次のような結果が生じることを理解してください。

- 需要が非常に多い時期には、Spot キャパシティーが利用できない場合があります。これにより、Amazon EC2 スポットインスタンスの起動が遅れる場合があります。このようなイベントで、Amazon ECS サービスはタスクの起動を再試行し、Amazon EC2 Auto Scaling グループも、必要なキャパシティーが利用可能になるまでインスタンスの起動を再試行します。Amazon EC2 は Spot キャパシティーをオンデマンドキャパシティーに置き換えません。
- キャパシティーに対する全体的な需要が高まると、スポットインスタンスとタスクは 2 分間の警告だけで終了する場合があります。警告が送信されたら、インスタンスが完全に終了する前に、必要に応じてタスクを順番にシャットダウンする必要があります。これにより、エラーの可能性を最小限に抑えられます。正常なシャットダウンの詳細については、「ECS による正常なシャットダウン」を参照してください。

スポットのキャパシティー不足を最小限に抑えるために、次の推奨事項を検討してください。

- 複数のリージョンとアベイラビリティーゾーンを使用する - Spot キャパシティーはリージョンとアベイラビリティーゾーンによって異なります。複数のリージョンとアベイラビリティーゾーンでワークロードを実行することで、スポットの可用性を向上できます。可能な場合は、タスクとインスタンスを実行するリージョンのすべてのアベイラビリティーゾーンのサブネットを指定してください。

- 複数の Amazon EC2 インスタンスタイプを使用する- Amazon EC2 Auto Scaling で混合インスタンスポリシーを使用すると、複数のインスタンスタイプが Auto Scaling グループに起動されます。これにより、Spot キャパシティーのリクエストを必要なときに確実に処理できます。信頼性を最大化し、複雑さを最小限に抑えるには、混合インスタンスポリシーでほぼ同じ量の CPU とメモリを備えたインスタンスタイプを使用してください。これらのインスタンスは、異なる世代のものでも、同じ基本インスタンスタイプのバリアントのものでもかまいません。不要な追加機能が付属している場合があることに注意してください。このようなリストの例としては、m4.large、m5.large、m5a.large、m5d.large、m5n.large、m5dn.large、m5ad.large などがあります。詳細については「Amazon EC2 Auto Scaling ユーザーガイド」の「複数のインスタンスタイプと購入オプションを使用する Auto Scaling グループ」を参照してください。
- キャパシティーが最適化されたスポット割り当て戦略を使用する - Amazon EC2 スポットでは、容量を最適化する割り当て戦略とコストを最適化する割り当て戦略のどちらかを選択できます。新しいインスタンスを起動するときにキャパシティー最適化戦略を選択した場合、Amazon EC2 スポットは選択したアベイラビリティーゾーンで最も可用性の高いインスタンスタイプを選択します。これにより、インスタンスが起動後すぐに終了する可能性が低くなります。

コンテナインスタンスでスポット終了通知を設定する方法については、以下を参照してください。

- スポットインスタンス通知を受信するように Amazon ECS Linux コンテナインスタンスを設定する
- スポットインスタンス通知を受信するように Amazon ECS Windows コンテナインスタンスを設定する

Amazon ECS に最適化された Linux AMI

> **Important**
>
> Amazon ECS 向けに最適化された Amazon Linux 2 AMI は 2026 年 6 月 30 日にサポート終了となり、上流の Amazon Linux 2 オペレーティングシステムと同じ終了日が反映されます (詳細については、「Amazon Linux 2 に関するよくある質問」を参照してください)。アプリケーションをアップグレードして、2028 年までの長期サポートが提供されている Amazon Linux 2023 を使用することをお勧めします。Amazon Linux 2 から Amazon Linux 2023 への移行については、「Migrating from the Amazon Linux 2 Amazon ECS-optimized AMI to the Amazon Linux 2023 Amazon ECS-optimized AMI」を参照してください。

Amazon ECS に最適化された AMI はすべて、非推奨になる日が AMI 作成日から 2 年後にデフォルト設定されています。Amazon EC2 `DescribeImages` API を使用して、AMI の非推奨化ステータスと日付を確認できます。詳細については、「Amazon Elastic Compute Cloud API リファレンス」の「DescribeImages」を参照してください。

Amazon ECS は、コンテナワークロードを実行する要件と推奨事項で事前設定された Amazon ECS に最適化された AMI を備えています。Amazon EC2 インスタンスには、Amazon ECS 向けに最適化された Amazon Linux 2023 AMI を使用することをお勧めします。最新の Amazon ECS に最適化された AMI からコンテナインスタンスを起動することで、最新のセキュリティアップデートや、現行バージョンのコンテナエージェントを確実に取得できます。インスタンスを起動する方法についての詳細は、Amazon ECS Linux コンテナインスタンスの起動 を参照してください。

コンソールを使用してクラスターを作成すると、Amazon ECS は選択したオペレーティングシステムに関連付けられた最新の AMI を使用してインスタンスの起動テンプレートを作成します。

CloudFormation を使用してクラスターを作成する場合、SSM パラメータは Auto Scaling グループインスタンスの Amazon EC2 起動テンプレートの一部となります。動的な Systems Manager パラメータを使用して、デプロイする Amazon ECS Optimized AMI を決定するようにテンプレートを設定できます。このパラメータにより、スタックをデプロイするたびに、EC2 インスタンスに適用する必要がある利用可能な更新があるかどうかがチェックされます。Systems Manager パラメータの使用方法の例については、「AWS CloudFormation ユーザーガイド」の「Amazon ECS に最適化された Amazon Linux 2023 AMI を使用して Amazon ECS クラスターを作成する」を参照してください。

Amazon ECS に最適化された AMI をカスタマイズする必要がある場合は、GitHub の「Amazon ECS Optimized AMI Build Recipes」を参照してください。

Amazon Linux 2023 オペレーティングシステムを実行する Amazon EC2 インスタンスでは、Amazon ECS に最適化された AMI の次のバリアントを使用できます。

オペレーティングシステム	AMI	説明	ストレージ設定
Amazon Linux 2023	Amazon ECS 最適化 Amazon Linux 2023 AMI	Amazon Linux 2023 は、AWS の次世代 Amazon Linux です。ほとんどの場合、Amazon ECS ワークロードのため	デフォルトでは、Amazon ECS 最適化 Amazon Linux 2023 AMI は 単一の 30 GiB ルートボリュームが付属しています。30

オペレーティングシステム	AMI	説明	ストレージ設定
		に Amazon EC2 インスタンスを起動するためにお勧めします。詳細については、「Amazon Linux 2023 ユーザーガイド」の「Amazon Linux 2023 とは」を参照してください。	GiB ルートボリュームのサイズを起動時に変更して、コンテナインスタンスで使用可能なストレージを増やすことができます。このストレージは、オペレーティングシステム用と Docker イメージおよびメタデータ用に使用されます。 Amazon ECS 最適化 Amazon Linux 2023 AMI のデフォルトファイルシステムは`xfs`を使用しており、Docker は `overlay2` ストレージドライバーを使用しています。詳細については、Docker ドキュメントの「OverlayFS ストレージドライバーを使用する」を参照してください。

オペレーティングシステム	AMI	説明	ストレージ設定
Amazon Linux 2023 (arm64)	Amazon ECS 最適化 Amazon Linux 2023 (arm64) AMI	この AMI は Amazon Linux 2023 に基づいており、ARM ベースの AWS Graviton/Graviton 2/Graviton 3/Graviton 4 プロセッサを搭載した Amazon EC2 インスタンスを、Amazon ECS ワークロードのために起動する場合に使用することが推奨されています。詳細については、「Amazon EC2 インスタンスタイプガイド」の「Amazon EC2 汎用インスタンスの仕様」を参照してください。	デフォルトでは、Amazon ECS 最適化 Amazon Linux 2023 AMI は 単一の 30 GiB ルートボリュームが付属しています。30 GiB ルートボリュームのサイズを起動時に変更して、コンテナインスタンスで使用可能なストレージを増やすことができます。このストレージは、オペレーティングシステム用と Docker イメージおよびメタデータ用に使用されます。 Amazon ECS 最適化 Amazon Linux 2023 AMI のデフォルトファイルシステムは`xfs`を使用しており、Docker は `overlay2` ストレージドライバーを使用しています。詳細については、Docker ドキュメントの「OverlayFS ストレージドライバーを

オペレーティングシステム	AMI	説明	ストレージ設定
			使用する」を参照してください。

オペレーティングシステム	AMI	説明	ストレージ設定
Amazon Linux 2023 (Neuron)	Amazon ECS 最適化 Amazon Linux 2023 AMI	これは Amazon Linux 2023 をベースにした、Amazon EC2 Inf1、Trn1、または Inf2 インスタンス用の AMI です。AWS Inferentia および AWS Trainium ドライバーと Docker 用の AWS Neuron ランタイムが事前設定されており、Amazon ECS での機械学習推論ワークロードの実行が容易になります。詳細については、「AWS Neuron 機械学習ワークロードでの Amazon ECS タスク定義」を参照してください。 Amazon ECS に最適化された Amazon Linux 2023 (Neuron) AMI には、AWS CLI はプリインストールされていません。	デフォルトでは、Amazon ECS 最適化 Amazon Linux 2023 AMI は 単一の 30 GiB ルートボリュームが付属しています。30 GiB ルートボリュームのサイズを起動時に変更して、コンテナインスタンスで使用可能なストレージを増やすことができます。このストレージは、オペレーティングシステム用と Docker イメージおよびメタデータ用に使用されます。 Amazon ECS 最適化 Amazon Linux 2023 AMI のデフォルトファイルシステムは`xfs`を使用しており、Docker は `overlay2` ストレージドライバーを使用しています。詳細については、Docker ドキュメントの「OverlayFS ストレージドライバーを

オペレーティングシステム	AMI	説明	ストレージ設定
			使用する」を参照してください。

オペレーティングシステム	AMI	説明	ストレージ設定
Amazon Linux 2023 GPU	Amazon ECS 最適化 Amazon Linux 2023 GPU AMI	この AMI は Amazon Linux 2023 に基づいており、Amazon EC2 GPU ベースのインスタンスを、Amazon ECS ワークロードのために起動する場合に使用することが推奨されています。NVIDIA カーネルドライバーと Docker GPU ランタイムが事前に構成されており、Amazon ECS で GPU を利用する実行中のワークロードになります。詳細については、「GPU ワークロード向けの Amazon ECS タスク定義」を参照してください。	デフォルトでは、Amazon ECS 最適化 Amazon Linux 2023 AMI は 単一の 30 GiB ルートボリュームが付属しています。30 GiB ルートボリュームのサイズを起動時に変更して、コンテナインスタンスで使用可能なストレージを増やすことができます。このストレージは、オペレーティングシステム用と Docker イメージおよびメタデータ用に使用されます。 Amazon ECS 最適化 Amazon Linux 2023 AMI のデフォルトファイルシステムは`xfs`を使用しており、Docker は `overlay2` ストレージドライバーを使用しています。詳細については、Docker ドキュメントの「OverlayFS ストレージドライバーを

オペレーティングシステム	AMI	説明	ストレージ設定
			使用する」を参照してください。

Amazon Linux 2 オペレーティングシステムを実行する Amazon EC2 インスタンスでは、Amazon ECS に最適化された AMI の次のバリアントを使用できます。

オペレーティングシステム	AMI	説明	ストレージ設定
Amazon Linux 2	Amazon ECS に最適化された Amazon Linux 2 カーネル 5.10 AMI	この AMI は Amazon Linux 2 をベースにしており、Amazon ECS ワークロードに Linux カーネル 4.14 ではなく Linux カーネル 5.10 を使用して、Amazon EC2 インスタンスを起動したい場合に使用します。Amazon ECS に最適化された Amazon Linux 2 カーネル 5.10 AMI では、AWS CLI は事前インストールされていません。	デフォルトでは、Amazon Linux 2 ベースの Amazon ECS に最適化された AMI (Amazon ECS に最適化された Amazon Linux 2 AMI、Amazon ECS に最適化された Amazon Linux 2 (arm64) AMI、および Amazon ECS GPU に最適化された AMI) には、1 つの 30 GiB のルートボリュームが付属しています。30 GiB ルートボリュームのサイズを起動時に変更して、コンテナインスタンスで使用可能なストレージを増やすことができます。このストレージは、オペ

オペレーティングシステム	AMI	説明	ストレージ設定
			レーティングシステム用と Docker イメージおよびメタデータ用に使用されます。 Amazon ECS に最適化された Amazon Linux 2 AMI のデフォルトファイルシステムは`xfs`を使用しており、Docker は `overlay2` ストレージドライバーを使用しています。詳細については、Docker ドキュメントの「OverlayFS ストレージドライバーを使用する」を参照してください。

オペレーティングシステム	AMI	説明	ストレージ設定
Amazon Linux 2	Amazon ECS に最適化された Amazon Linux 2 AMI	これは Amazon ECS のワークロード用です。Amazon ECS に最適化された Amazon Linux 2 AMI には、AWS CLI はプリインストールされていません。	デフォルトでは、Amazon Linux 2 ベースの Amazon ECS に最適化された AMI (Amazon ECS に最適化された Amazon Linux 2 AMI、Amazon ECS に最適化された Amazon Linux 2 (arm64) AMI、および Amazon ECS GPU に最適化された AMI) には、1 つの 30 GiB のルートボリュームが付属しています。30 GiB ルートボリュームのサイズを起動時に変更して、コンテナインスタンスで使用可能なストレージを増やすことができます。このストレージは、オペレーティングシステム用と Docker イメージおよびメタデータ用に使用されます。 Amazon ECS に最適化された Amazon Linux 2 AMI のデフォルトファイルシステムは`xfs`を使用

オペレーティングシステム	AMI	説明	ストレージ設定
			しており、Docker は `overlay2` ストレージドライバーを使用しています。詳細については、Docker ドキュメントの「OverlayFS ストレージドライバーを使用する」を参照してください。

オペレーティングシステム	AMI	説明	ストレージ設定
Amazon Linux 2 (arm64)	Amazon ECS に最適化された Amazon Linux 2 カーネル 5.10 (arm64) AMI	この AMI は Amazon Linux 2 をベースにし、ARM ベースの AWS Graviton/Graviton 2/Graviton 3/Graviton 4 プロセッサを搭載したAmazon EC2 インスタンス用であり、Amazon EC2 S ワークロードに Linux カーネル 4.14 ではなく Linux カーネル 5.10を使用する場合に使用します。詳細については、「Amazon EC2 インスタンスタイプガイド」の「Amazon EC2 汎用インスタンスの仕様」を参照してください。 Amazon ECS に最適化された Amazon Linux 2 (arm64) AMI には、AWS CLI はプリインストールされていません。	デフォルトでは、Amazon Linux 2 ベースの Amazon ECS に最適化された AMI (Amazon ECS に最適化された Amazon Linux 2 AMI、Amazon ECS に最適化された Amazon Linux 2 (arm64) AMI、および Amazon ECS GPU に最適化された AMI) には、1 つの 30 GiB のルートボリュームが付属しています。30 GiB ルートボリュームのサイズを起動時に変更して、コンテナインスタンスで使用可能なストレージを増やすことができます。このストレージは、オペレーティングシステム用と Docker イメージおよびメタデータ用に使用されます。 Amazon ECS に最適化された Amazon Linux 2 AMI のデフォルトファイルシステムは`xfs`を使用

オペレーティングシステム	AMI	説明	ストレージ設定
			しており、Docker は `overlay2` ストレージドライバーを使用しています。詳細については、Docker ドキュメントの「OverlayFS ストレージドライバーを使用する」を参照してください。

オペレーティングシステム	AMI	説明	ストレージ設定
Amazon Linux 2 (arm64)	Amazon ECS に最適化された Amazon Linux 2 (arm64) AMI	この AMI は Amazon Linux 2 をベースにしており、Amazon ECS ワークロードに ARM ベースの AWS Graviton/Graviton 2/Graviton 3/Graviton 4 プロセッサを搭載した Amazon EC2 インスタンスを起動する場合に使用します。 Amazon ECS に最適化された Amazon Linux 2 (arm64) AMI には、AWS CLI はプリインストールされていません。	デフォルトでは、Amazon Linux 2 ベースの Amazon ECS に最適化された AMI (Amazon ECS に最適化された Amazon Linux 2 AMI、Amazon ECS に最適化された Amazon Linux 2 (arm64) AMI、および Amazon ECS GPU に最適化された AMI) には、1 つの 30 GiB のルートボリュームが付属しています。30 GiB ルートボリュームのサイズを起動時に変更して、コンテナインスタンスで使用可能なストレージを増やすことができます。このストレージは、オペレーティングシステム用と Docker イメージおよびメタデータ用に使用されます。 Amazon ECS に最適化された Amazon Linux 2 AMI のデフォルトファイルシステムは`xfs`を使用

オペレーティングシステム	AMI	説明	ストレージ設定
			しており、Docker は `overlay2` ストレージドライバーを使用しています。詳細については、Docker ドキュメントの「OverlayFS ストレージドライバーを使用する」を参照してください。

オペレーティングシステム	AMI	説明	ストレージ設定
Amazon Linux 2 (GPU)	Amazon ECS GPU に最適化されたカーネル 5.10 AMI	この AMI は Amazon Linux 2 に基づいており、Amazon ECS ワークロード用に Linux カーネル 5.10 で Amazon EC2 GPU ベースのインスタンスを起動する場合に使用することをお勧めします。NVIDIA カーネルドライバーと Docker GPU ランタイムが事前に構成されており、Amazon ECS で GPU を利用する実行中のワークロードになります。詳細については、「GPU ワークロード向けの Amazon ECS タスク定義」を参照してください。	デフォルトでは、Amazon Linux 2 ベースの Amazon ECS に最適化された AMI (Amazon ECS に最適化された Amazon Linux 2 AMI、Amazon ECS に最適化された Amazon Linux 2 (arm64) AMI、および Amazon ECS GPU に最適化された AMI) には、1 つの 30 GiB のルートボリュームが付属しています。30 GiB ルートボリュームのサイズを起動時に変更して、コンテナインスタンスで使用可能なストレージを増やすことができます。このストレージは、オペレーティングシステム用と Docker イメージおよびメタデータ用に使用されます。 Amazon ECS に最適化された Amazon Linux 2 AMI のデフォルトファイルシステムは`xfs`を使用

オペレーティングシステム	AMI	説明	ストレージ設定
			しており、Docker は `overlay2` ストレージドライバーを使用しています。詳細については、Docker ドキュメントの「OverlayFS ストレージドライバーを使用する」を参照してください。

オペレーティングシステム	AMI	説明	ストレージ設定
Amazon Linux 2 (GPU)	Amazon ECS GPU に最適化された AMI	この AMI は Amazon Linux 2 に基づいており、Amazon ECS ワークロード用に Linux カーネル 4.14 で Amazon EC2 GPU ベースのインスタンスを起動する場合に使用することをお勧めします。NVIDIA カーネルドライバーと Docker GPU ランタイムが事前に構成されており、Amazon ECS で GPU を利用する実行中のワークロードになります。詳細については、「GPU ワークロード向けの Amazon ECS タスク定義」を参照してください。	デフォルトでは、Amazon Linux 2 ベースの Amazon ECS に最適化された AMI (Amazon ECS に最適化された Amazon Linux 2 AMI、Amazon ECS に最適化された Amazon Linux 2 (arm64) AMI、および Amazon ECS GPU に最適化された AMI) には、1 つの 30 GiB のルートボリュームが付属しています。30 GiB ルートボリュームのサイズを起動時に変更して、コンテナインスタンスで使用可能なストレージを増やすことができます。このストレージは、オペレーティングシステム用と Docker イメージおよびメタデータ用に使用されます。 Amazon ECS に最適化された Amazon Linux 2 AMI のデフォルトファイルシステムは`xfs`を使用

オペレーティングシステム	AMI	説明	ストレージ設定
			しており、Docker は `overlay2` ストレージドライバーを使用しています。詳細については、Docker ドキュメントの「OverlayFS ストレージドライバーを使用する」を参照してください。

オペレーティングシステム	AMI	説明	ストレージ設定
Amazon Linux 2 (Neuron)	Amazon ECS に最適化された Amazon Linux 2 (Neuron) カーネル 5.10 AMI	これは Amazon Linux 2 をベースにした、Amazon EC2 Inf1、Trn1、または Inf2 インスタンス用の AMI です。Linux カーネル 5.10 搭載 AWS Inferentia および AWS Trainium ドライバーと Docker 用の AWS Neuron ランタイムが事前設定されており、Amazon ECS での機械学習推論ワークロードの実行が容易になります。詳細については、「AWS Neuron 機械学習ワークロードでの Amazon ECS タスク定義」を参照してください。Amazon ECS 最適化 Amazon Linux 2 (Neuron) AMI には、AWS CLI はプリインストールされていません。	デフォルトでは、Amazon Linux 2 ベースの Amazon ECS に最適化された AMI (Amazon ECS に最適化された Amazon Linux 2 AMI、Amazon ECS に最適化された Amazon Linux 2 (arm64) AMI、および Amazon ECS GPU に最適化された AMI) には、1 つの 30 GiB のルートボリュームが付属しています。30 GiB ルートボリュームのサイズを起動時に変更して、コンテナインスタンスで使用可能なストレージを増やすことができます。このストレージは、オペレーティングシステム用と Docker イメージおよびメタデータ用に使用されます。 Amazon ECS に最適化された Amazon Linux 2 AMI のデフォルトファイルシステムは`xfs`を使用

オペレーティングシステム	AMI	説明	ストレージ設定
			しており、Docker は `overlay2` ストレージドライバーを使用しています。詳細については、Docker ドキュメントの「OverlayFS ストレージドライバーを使用する」を参照してください。

オペレーティングシステム	AMI	説明	ストレージ設定
Amazon Linux 2 (Neuron)	Amazon ECS 最適化 Amazon Linux 2 (Neuron) AMI	これは Amazon Linux 2 をベースにした、Amazon EC2 Inf1、Trn1、または Inf2 インスタンス用の AMI です。AWS Inferentia および AWS Trainium ドライバーと Docker 用の AWS Neuron ランタイムが事前設定されており、Amazon ECS での機械学習推論ワークロードの実行が容易になります。詳細については、「AWS Neuron 機械学習ワークロードでの Amazon ECS タスク定義」を参照してください。Amazon ECS 最適化 Amazon Linux 2 (Neuron) AMI には、AWS CLI はプリインストールされていません。	デフォルトでは、Amazon Linux 2 ベースの Amazon ECS に最適化された AMI (Amazon ECS に最適化された Amazon Linux 2 AMI、Amazon ECS に最適化された Amazon Linux 2 (arm64) AMI、および Amazon ECS GPU に最適化された AMI) には、1 つの 30 GiB のルートボリュームが付属しています。30 GiB ルートボリュームのサイズを起動時に変更して、コンテナインスタンスで使用可能なストレージを増やすことができます。このストレージは、オペレーティングシステム用と Docker イメージおよびメタデータ用に使用されます。 Amazon ECS に最適化された Amazon Linux 2 AMI のデフォルトファイルシステムは`xfs`を使用

オペレーティングシステム	AMI	説明	ストレージ設定
			しており、Docker は overlay2 ストレージドライバーを使用しています。詳細については、Docker ドキュメントの「OverlayFS ストレージドライバーを使用する」を参照してください。

Amazon ECS は、GitHub で Amazon ECS を使用して最適化した AMI の Linux バリアントの変更ログを提供します。詳細については、「Changelog」を参照してください。

Amazon ECS 最適化 AMI の Linux バリエーションは、Amazon Linux 2 AMI または Amazon Linux 2023 AMI をベースとして使用します。Systems Manager Parameter Store API をクエリすることで、各バリアントの AMI 名を取得できます。詳細については、「Amazon ECS に最適化された Linux AMI メタデータを取得する」を参照してください。Amazon Linux 2 AMI リリースノートも公開されています。詳細については、「Amazon Linux 2 リリースノート」を参照してください。Amazon Linux 2023 リリースノートも公開されています。詳細については、「Amazon Linux 2023 リリースノート」を参照してください。

次のページでは、変更に関する追加情報を説明します。

- GitHub のソース AMI リリースノート
- Docker ドキュメントの「Docker Engine リリースノート」
- NVIDIA ドキュメントの「NVIDIA ドライバードキュメント」
- GitHub での Amazon ECS エージェントの変更ログ

  ecs-init アプリケーションのソースコード、エージェントをパッケージ化するためのスクリプトと構成は、エージェントリポジトリの一部になりました。ecs-init の古いバージョンおよびパッケージについては、GitHub で「Amazon ecs-init の変更ログ」を参照してください。

Amazon ECS に最適化された AMI へのセキュリティ更新の適用

Amazon Linux に基づく Amazon ECS に最適化された AMI には、cloud-init のカスタマイズされたバージョンが含まれています。Cloud-init は、Linux イメージをクラウドコンピューティング環境でブートストラップし、インスタンスの起動時に必要なアクションを実行するために使用されるパッケージです。デフォルトで、2024 年 6 月 12 日より前にリリースされた Amazon Linux に基づくすべての Amazon ECS に最適化された AMI では、インスタンスの起動時にすべての「重大」および「重要」なセキュリティ更新が適用されます。

2024 年 6 月 12 日のリリース以降、Amazon Linux 2 に基づく Amazon ECS に最適化された AMI の デフォルトの動作には、起動時のパッケージの更新が含まれなくなります。代わりに、リリースが利用可能になりしだい、Amazon ECS に最適化された新しい AMI に更新することをお勧めします。Amazon ECS に最適化された AMI は、利用可能なセキュリティ更新またはベース AMI の変更があったときにリリースされます。この結果、最新のパッケージバージョンとセキュリティ更新が確実に適用され、パッケージバージョンはインスタンスの起動を通じて不変です。Amazon ECS に最適化された最新の AMI の取得方法の詳細については、「Amazon ECS に最適化された Linux AMI メタデータを取得する」を参照してください。

新しい AMI が利用可能になりしだい更新するように環境を自動化することをお勧めします。利用可能なオプションの詳細については、「Amazon ECS のマネージドインスタンスドレインにより Amazon EC2 キャパシティの管理が容易に」を参照してください。

特定の AMI バージョンに「重大」および「重要」のセキュリティ更新を引き続き手動で適用する場合は、Amazon EC2 インスタンスで次のコマンドを実行します。

```
yum update --security
```

> **Warning**
>
> Docker または containerd パッケージを更新すると、ホストで実行中のすべてのコンテナが停止するため、実行中のすべての Amazon ECS タスクも停止されることになります。サービスの中断を最小限に抑えるため、適切に計画してください。

起動時にセキュリティ更新を再度有効にする場合は、Amazon EC2 インスタンスの起動時に cloud-init ユーザーデータの `#cloud-config` セクションに次の行を追加できます。詳細については、「Amazon Linux ユーザーガイド」の「Amazon Linux 2 で cloud–init を使用する」を参照してください。

```
#cloud-config
repo_upgrade: security
```

Amazon ECS 最適化 AL2023 GPU AMI のバージョンロックパッケージ

Amazon ECS 最適化 AL2023 GPU AMI で GPU 機能の正確かつ高性能な動作を確保するには、特定のパッケージが不可欠です。具体的には次のとおりです。

- NVIDIA ドライバー (`nvidia*`)
- カーネルモジュール (`kmod*`)
- NVIDIA ライブラリ (`libnvidia*`)
- カーネルパッケージ (`kernel*`)
- Xorg サーバーパッケージ (`xorg*`)
- NVIDIA Data Center GPU Manager (DCGM) (`datacenter-gpu-manager-*`)

> **Note**
>
> これは網羅的なリストではありません。ロックされたパッケージの完全なリストは、`dnf versionlock list` を使用することで入手できます。

これらのパッケージは、安定性を確保し、GPU ワークロードを中断する可能性のある意図しない変更を防ぐためにバージョンロックされています。このため、これらのパッケージの変更は、潜在的な問題を適切に処理し、GPU の機能性を維持するマネージドプロセスの範囲内で行う必要があるのが一般的です。

意図しない変更を防ぐため、これらのパッケージには `dnf versionlock` プラグインが使用されています。

ロックされたパッケージを変更したい場合は、以下を実行できます。

```
# unlock a single package
sudo dnf versionlock delete $PACKAGE_NAME

# unlock all packages
sudo dnf versionlock clear
```

> **Important**
>
> これらのパッケージを更新する必要があるお客様は、必要な更新が含まれる最新の AMI バージョンの使用を検討するようにしてください。既存のインスタンスを更新する必要がある場合は、パッケージのロック解除、更新、再ロックを含めた慎重なアプローチを用いることで、プロセスの全体を通じて GPU の機能性を常に確保するようにしてください。

Amazon ECS に最適化された Linux AMI メタデータを取得する

Amazon ECS に最適化された AMI のメタデータをプログラムで取得できます。メタデータには、AMI 名、Amazon ECS コンテナエージェントバージョン、Docker バージョンを含む ECS ランタイムバージョンなどが入っています。

コンソールを使用してクラスターを作成すると、Amazon ECS は選択したオペレーティングシステムに関連付けられた最新の AMI を使用してインスタンスの起動テンプレートを作成します。

CloudFormation を使用してクラスターを作成する場合、SSM パラメータは Auto Scaling グループインスタンスの Amazon EC2 起動テンプレートの一部となります。動的な Systems Manager パラメータを使用して、デプロイする Amazon ECS Optimized AMI を決定するようにテンプレートを設定できます。このパラメータにより、スタックをデプロイするたびに、EC2 インスタンスに適用する必要がある利用可能な更新があるかどうかがチェックされます。Systems Manager パラメータの使用方法の例については、「AWS CloudFormation ユーザーガイド」の「Amazon ECS に最適化された Amazon Linux 2023 AMI を使用して Amazon ECS クラスターを作成する」を参照してください。

Amazon ECS に最適化された AMI の各バリアントの AMI ID、イメージ名、オペレーティングシステム、コンテナエージェントバージョン、ソースイメージ名、ランタイムバージョンは、Systems Manager Parameter Store API にクエリすることでプログラムで取得できます。Systems Manager パラメータストア API の詳細については、「GetParameters」および「GetParametersByPath」を参照してください。

> **Note**
>
> Amazon ECS に最適化された AMI メタデータを取得するには、管理ユーザーに次の IAM アクセス権限が必要です。`AmazonECS_FullAccess` IAM ポリシーには、次の許可が追加されています。
>
> - ssm:GetParameters

- ssm:GetParameter
- ssm:GetParametersByPath

Systems Manager パラメータストアのパラメータフォーマット

以下は、各 Amazon ECS に最適化された AMI バリアントのパラメータ名のフォーマットです。

Linux Amazon ECS に最適化された AMI

- Amazon Linux 2023 AMI メタデータ:

```
/aws/service/ecs/optimized-ami/amazon-linux-2023/<version>
```

- Amazon Linux 2023 (arm64) AMI メタデータ:

```
/aws/service/ecs/optimized-ami/amazon-linux-2023/arm64/<version>
```

- Amazon Linux 2023 (Neuron) AMI メタデータ:

```
/aws/service/ecs/optimized-ami/amazon-linux-2023/neuron/<version>
```

- Amazon Linux 2023 (GPU) AMI メタデータ:

```
/aws/service/ecs/optimized-ami/amazon-linux-2023/gpu/<version>
```

  Amazon Linux 2 AMI メタデータ:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/<version>
```

- Amazon Linux 2 カーネル 5.10 AMI メタデータ:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/kernel-5.10/<version>
```

- Amazon Linux 2 (arm64) AMI メタデータ:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/arm64/<version>
```

- Amazon Linux 2 カーネル 5.10 (arm64) AMI メタデータ:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/kernel-5.10/arm64/<version>
```

- Amazon ECS GPU に最適化されたカーネル 5.10 AMI のメタデータ:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/kernel-5.10/gpu/<version>
```

- Amazon Linux 2 (GPU) AMI メタデータ:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/<version>
```

- Amazon ECS に最適化された Amazon Linux 2 (Neuron) カーネル 5.10 AMI のメタデータ:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/kernel-5.10/inf/<version>
```

- Amazon Linux 2 (Neuron) AMI メタデータ:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/inf/<version>
```

以下のパラメータ名の形式は、サブパラメータ `image_id` を使用して、推奨される最新の Amazon ECS に最適化された Amazon Linux 2 AMI のイメージ ID を取得します。

```
/aws/service/ecs/optimized-ami/amazon-linux-2/recommended/image_id
```

以下のパラメータ名の形式は、AMI 名を指定して、特定の Amazon ECS に最適化された AMI バージョンのメタデータを取得します。

- Amazon ECS に最適化された Amazon Linux 2 AMI メタデータ:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/amzn2-ami-ecs-hvm-2.0.20181112-x86_64-ebs
```

> **Note**
>
> Amazon ECS に最適化された Amazon Linux 2 AMI のすべてのバージョンを取得できます。Amazon ECS に最適化された AMI バージョン `amzn-ami-2017.09.l-amazon-ecs-optimized` (Linux) 以降のみ取得できます。

例

以下の例は、それぞれの Amazon ECS に最適化された AMI バリアントのメタデータを取得する方法を示しています。

推奨される最新の Amazon ECS に最適化された AMI メタデータを取得する

推奨される最新の Amazon ECS に最適化された AMI を取得するには、AWS CLI で次の AWS CLI コマンドを使用します。

Linux Amazon ECS に最適化された AMI

- Amazon ECS 最適化 Amazon Linux 2023 AMI の場合:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2023/
recommended --region us-east-1
```

- Amazon ECS 最適化 Amazon Linux 2023 (arm64) AMI の場合:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2023/
arm64/recommended --region us-east-1
```

- Amazon ECS に最適化された Amazon Linux 2023 (Neuron) AMI の場合:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2023/
neuron/recommended --region us-east-1
```

- Amazon ECS に最適化された Amazon Linux 2023 GPU AMI の場合:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2023/gpu/
recommended --region us-east-1
```

- Amazon ECS に最適化された Amazon Linux 2 カーネル 5.10 AMI の場合:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/
kernel-5.10/recommended --region us-east-1
```

- Amazon ECS に最適化された Amazon Linux 2 AMI の場合:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/
recommended --region us-east-1
```

- Amazon ECS に最適化された Amazon Linux 2 カーネル 5.10 (arm64) AMI の場合:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/
kernel-5.10/arm64/recommended --region us-east-1
```

- Amazon ECS に最適化された Amazon Linux 2 (arm64) AMI の場合:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/arm64/
recommended --region us-east-1
```

- Amazon ECS GPU に最適化されたカーネル 5.10 AMI の場合:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/
kernel-5.10/gpu/recommended --region us-east-1
```

- Amazon ECS GPU に最適化された AMI の場合:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/gpu/
recommended --region us-east-1
```

- Amazon ECS に最適化された Amazon Linux 2 (Neuron) カーネル 5.10 AMI の場合:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/
kernel-5.10/inf/recommended --region us-east-1
```

- Amazon ECS 最適化 Amazon Linux 2 (Neuron) AMI の場合:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/inf/
recommended --region us-east-1
```

推奨される最新の Amazon ECS に最適化された Amazon Linux 2023 AMI のイメージ ID を取得する

推奨される最新の Amazon ECS に最適化された Amazon Linux 2023 AMI ID のイメージ ID を取得するには、サブパラメータ `image_id` を使用します。

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-
linux-2023/recommended/image_id --region us-east-1
```

`image_id` 値のみを取得するには、特定のパラメータ値のみのクエリを行うことができます。次に例を示します。

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2023/
recommended/image_id --region us-east-1 --query "Parameters[0].Value"
```

Amazon ECS に最適化された特定の Amazon Linux 2 AMI バージョンのメタデータを取得する

Amazon ECS に最適化された特定の Amazon Linux AMI バージョンのメタデータを取得するには、AWS CLI で次の AWS CLI コマンドを使用します。AMI 名を、Amazon ECS に最適化された Amazon Linux AMI の名前と置き換えます。

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/amzn2-ami-
ecs-hvm-2.0.20200928-x86_64-ebs --region us-east-1
```

Systems Manager の GetParametersByPath API を使用した Amazon ECS に最適化された Amazon Linux 2 カーネル 5.10 AMI メタデータの取得

Systems Manager の GetParametersByPath API を使用して、Amazon ECS に最適化された Amazon Linux 2 AMI メタデータを取得します。AWS CLI で次のコマンドを使用します。

```
aws ssm get-parameters-by-path --path /aws/service/ecs/optimized-ami/amazon-linux-2/
kernel-5.10/ --region us-east-1
```

推奨される最新の Amazon ECS に最適化された Amazon Linux 2 カーネル 5.10 AMI のイメージ ID を取得する

サブパラメータ `image_id` を使用することで、推奨される最新の Amazon ECS に最適化された Amazon Linux 2 カーネル 5.10 AMI ID のイメージ ID を取得できます。

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/
kernel-5.10/recommended/image_id --region us-east-1
```

`image_id` 値のみを取得するには、特定のパラメータ値のみのクエリを行うことができます。次に例を示します。

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/
recommended/image_id --region us-east-1 --query "Parameters[0].Value"
```

CloudFormation テンプレートの推奨される最新の Amazon ECS に最適化された AMI を使用する

Systems Manager パラメータストア名を参照することにより、CloudFormation テンプレートで推奨される最新の Amazon ECS に最適化された AMI を参照できます。

Linux の例:

```
Parameters:kernel-5.10
  LatestECSOptimizedAMI:
    Description: AMI ID
    Type: AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>
    Default: /aws/service/ecs/optimized-ami/amazon-linux-2/kernel-5.10/recommended/
image_id
```

Amazon Linux 2 から Amazon Linux 2023 Amazon ECS 最適化 AMI への移行

Amazon Linux に続いて、Amazon ECS は、2026 年 6 月 30 日をもって Amazon Linux 2 Amazon ECS 最適化 AMI の標準サポートを終了します。この日以降は、Amazon ECS エージェントバージョンは固定され、新しい Amazon Linux 2 Amazon ECS 最適化 AMI は、ソースとなる Amazon Linux 2 AMI が更新された場合にのみ公開されます。完全なサポート終了 (EOL) は 2026 年 6 月 30 日に発生します。その後は、ソースとなる AMI が更新されても、Amazon ECS 向けに最適化された Amazon Linux 2 AMI は公開されません。

Amazon Linux 2023 は、デフォルトで安全性を確保するアプローチを採用し、事前設定されたセキュリティポリシー、Permissive モードの SELinux、デフォルトで有効になっている IMDSv2-only モード、最適化された起動時間、セキュリティとパフォーマンスの向上を目的として改善されたパッケージ管理機能を提供します。

Amazon Linux 2 と Amazon Linux 2023 の Amazon ECS 最適化 AMI は互換性が高く、ほとんどのお客様が感じる 2 つのオペレーティングシステム間でのワークロードの変化は最小限となるか、全く感じられないでしょう。

AL2 および AL2023 間の主な相違点の詳細については、「Amazon Linux 2023 ユーザーガイド」の「Amazon Linux 2 と Amazon Linux 2023 の比較」と「AL 2023 に関するよくある質問」を参照してください。

互換性に関する考慮事項

パッケージ管理と OS の更新

Amazon Linux の以前のバージョンとは異なり、Amazon ECS 向けに最適化された Linux 2023 AMI の Amazon Linux リポジトリは特定のバージョンにロックされています。このため、不要な変更や互換性を破る変更を引き起こす可能性のあるパッケージを誤って更新することがありません。詳細については、「Amazon Linux 2023 ユーザーガイド」の「Managing repositories and OS updates in Amazon Linux 2023」を参照してください。

Linux カーネルバージョン

Amazon Linux 2 AMI は Linux カーネル 4.14 および 5.10 をベースにしており、Amazon Linux 2023 は Linux カーネル 6.1 および 6.12 を使用しています。Amazon Linux 2023 の詳細については、「Amazon Linux 2023 ユーザーガイド」の「Comparing Amazon Linux 2 and Amazon Linux 2023 kernels」を参照してください。

パッケージアベイラビリティの変更

Amazon Linux 2023 でのパッケージにおける重要な変更点は次のとおりです。

- Amazon Linux 2 の一部のソースバイナリパッケージは、Amazon Linux 2023 では利用できなくなりました。詳細については、「Amazon Linux 2023 リリースノート」の「Packages removed from Amazon Linux 2023」を参照してください。
- Amazon Linux がさまざまなバージョンのパッケージをサポートする方法が変更されました。Amazon Linux 2 で使用される `amazon-linux-extras` システムは、Amazon Linux 2023 には存在しません。すべてのパッケージは、単純に「コア」リポジトリから使用できます。
- Enterprise Linux (EPEL) の追加パッケージは Amazon Linux 2023 ではサポートされません。詳細については、「Amazon Linux 2023 ユーザーガイド」の「EPEL compatibility in Amazon Linux 2023」を参照してください。
- 32-bit アプリケーションは Amazon Linux 2023 ではサポートされていません。詳細については、「Amazon Linux 2023 ユーザーガイド」の「Deprecated features from Amazon Linux 2」を参照してください。

コントロールグループ (cgroups) の変更点

コントロールグループ (cgroup) は、プロセスを階層的に整理し、システムリソースをプロセス間で分散するための Linux カーネルの機能です。コントロールグループは、コンテナランタイムの実装や、さまざまな用途に `systemd` によって使用されています。

Amazon Linux 2023 はデフォルトで cgroupv2 を使用し、Amazon Linux 2 は cgroupv1 を使用します。Amazon ECS エージェント、Docker、および containerd はすべて cgroupv1 と cgroupv2 の両方をサポートしています。cgroupv2 の詳細については、「Amazon Linux 2023 ユーザーガイド」の「Control groups v2 in Amazon Linux 2023」を参照してください。

メモリ使用量レポート

cgroupv2 は、コンテナメモリ使用量の計算方法を変更します。cgroupv1 (Amazon Linux 2) では、コンテナランタイムによってレポートされたコンテナのメモリ使用率には通常、ページキャッシュは

含まれません。cgroupv2 (Amazon Linux 2023) では、ページキャッシュはレポートされたメモリ使用率に含まれます。実際のアプリケーションのメモリ消費量が変更されていない場合でも、Amazon Linux 2023 では、Amazon Linux 2 と比較して、同じワークロードで高いメモリ使用率がレポートされる可能性があります。

本番ワークロードを移行する前に、Amazon Linux 2023 インスタンスのメモリ使用率をベンチマークし、必要に応じてタスクとコンテナのメモリ制限を調整することをお勧めします。Container Insights を使用して、Amazon Linux 2 と Amazon Linux 2023 のメモリ使用率を比較できます。

タスクメモリの可視性制限

コンテナレベルのメモリ制限を設定せずにタスクレベルでメモリを設定すると、Amazon ECS はその制限をタスクの cgroup に適用します。これは、タスクに属するすべてのコンテナの実効メモリ制限として機能します。cgroupv1 (Amazon Linux 2) では、子 cgroup は独自のレベルで有効なメモリ制限を読み取ることができます。cgroup 階層をトラバースする必要はありません。cgroupv2 (Amazon Linux 2023) では、有効な制限はコンテナに表示されません。さらに、Docker はデフォルトでプライベート cgroup 名前空間でコンテナを実行します。これにより、コンテナは cgroup 階層を通過して親の制限を検出できなくなります。

cgroup メモリ制限を読み取り、使用可能なメモリを決定するコンテナ内のプロセスには、タスクレベルの制限ではなくホストメモリ全体が表示されます。例えば、Java 仮想マシン (JVM) はこの問題の影響を受けることが知られています。cgroup 制限を使用してヒープを自動的にサイズ設定します。真の制限を検出できない場合、メモリを過剰に割り当ててしまい、メモリ不足 (OOM) による強制終了のリスクが高まります。リソース決定の cgroup 制限を読み取る他のランタイムまたはフレームワークも同様に影響を受ける可能性があります。

これは、タスクレベルのメモリが設定されているが、タスク定義でコンテナレベル `memory` が設定されていない場合に発生します。

この問題を解決するには、以下のいずれかの回避策を選択してください。

コンテナレベルのメモリ制限の設定 (推奨)

タスク定義のコンテナレベルで、`memory` パラメータをタスクレベルのメモリ値と等しく明示的に設定します。このアプローチは安全で信頼性が高く、インフラの変更も不要です。

エージェントレベルのタスクメモリ制限の伝播を有効にする

Amazon ECS エージェントで `ECS_PROPAGATE_TASK_MEMORY_LIMIT_CGROUPV2` 環境変数を `true` に設定します。有効化されていて cgroupv2 の場合、コンテナに明示的なコンテナレベルのメ

モリ制限が設定されていない場合、エージェントは各コンテナのメモリ制限をタスクレベルの値に自動的に設定します。これにより、タスク定義を変更することなく、コンテナ内でタスクのメモリ制限を可視化できます。

これは、Amazon EC2 インスタンスの Amazon ECS エージェント設定ファイル (`/etc/ecs/ecs.config`) で設定できます。

```
ECS_PROPAGATE_TASK_MEMORY_LIMIT_CGROUPV2=true
```

このオプションを使用するには、Amazon ECS エージェントのバージョン 1.104.0 以降が必要です。cgroupv2 を使用する Linux インスタンスにのみ適用されます。Windows ではサポートされていません。

インスタンスメタデータサービス (IMDS) の変更点

Amazon Linux 2023 では、デフォルトではインスタンスメタデータサービスのバージョン 2 (IMDSv2) が必要になります。IMDSv2 には、セキュリティ体制の改善に役立ついくつかの利点があります。セッション指向の認証方式を使用しており、セッションを開始するためにシンプルな HTTP PUT リクエストでシークレットトークンを作成する必要があります。セッショントークンの有効期間は 1 秒 ~ 6 時間です。

IMDSv1 から IMDSv2 への移行の詳細については、「Amazon EC2 ユーザーガイド」の「インスタンスメタデータサービスバージョン 2 の使用への移行」を参照してください。

IMDSv1 を使用する場合は、インスタンスのメタデータオプションの起動プロパティで設定を手動で上書きすることで使用可能になります。

メモリスワップの変更点

コンテナごとのメモリスワップは、Amazon Linux 2023 および cgroups v2 ではサポートされていません。詳細については、「Amazon ECS のコンテナスワップメモリ空間の管理」を参照してください。

FIPS 検証の変更点

Amazon Linux 2 は FIPS 140-2 に基づいて認定されており、Amazon Linux 2023 は FIPS 140-3 に基づいて認定されています。

Amazon Linux 2023 で FIPS モードを有効にするには、Amazon EC2 インスタンスに必要なパッケージをインストールし、「Amazon Linux 2023 ユーザーガイド」の「Enable FIPS Mode on Amazon Linux 2023」に記載された手順に従って設定を行います。

インスタンスのサポートの高速化

Amazon ECS 向けに最適化された Amazon Linux 2023 AMI は、Neuron と GPU の両方の高速化インスタンスタイプをサポートしています。詳細については、「Amazon ECS に最適化された Linux AMI」を参照してください。

カスタム AMI の構築

Amazon Linux 2023 では、正式にサポートおよび公開されている Amazon ECS 最適化 AMI に移行することをお勧めしますが、Amazon ECS 最適化 AMI の Linux バリアントの構築に使用されているオープンソースのビルドスクリプトを使用することで、引き続きカスタム Amazon Linux 2 Amazon ECS 最適化 AMI を構築できます。詳細については、「Amazon ECS 最適化 Linux AMI のビルドスクリプト」を参照してください。

移行戦略

徹底的なアプリケーションテストを含む移行計画を作成して実装することをお勧めします。以下のセクションでは、Amazon ECS インフラストラクチャの管理方法に基づくさまざまな移行戦略についての概要を説明します。

Amazon ECS キャパシティプロバイダーを利用した移行

1. 新しい起動テンプレートを使用して、新しいキャパシティプロバイダーを作成します。これは、既存の起動テンプレートと類似した起動テンプレートを持つ Auto Scaling グループを参照する必要がありますが、Amazon Linux 2 Amazon ECS 最適化 AMI ではなく、Amazon Linux 2023 バリアントのいずれかを指定する必要があります。この新しいキャパシティプロバイダーを既存の Amazon ECS クラスターに追加します。
2. クラスターのデフォルトのキャパシティプロバイダー戦略を更新して、既存の Amazon Linux 2 キャパシティプロバイダーと新しい Amazon Linux 2023 キャパシティプロバイダーの両方が含まれるようにします。Amazon Linux 2 プロバイダーの重みを大きく、Amazon Linux 2023 プロバイダーの重みを小さく設定します (Amazon Linux 2: 重み 80、Amazon Linux 2023: 重み 20 など)。これにより、新しいタスクがスケジュールされると、Amazon ECS は Amazon Linux 2023 インスタンスのプロビジョニングを開始します。インスタンスが正しく登録されていること、タスクが新しいインスタンスで正常に実行できることを確認します。
3. クラスターのデフォルト戦略におけるキャパシティプロバイダーの重みを徐々に調整し、Amazon Linux 2023 プロバイダーの重みを増加させつつ、Amazon Linux 2 プロバイダーの重みを時間の経過とともに減らします (例: 60/40、40/60、20/80)。個々のサービスプロバイダー戦略を更新して、Amazon Linux 2023 インスタンスを優先させることもできます。タスクの配置を監視して、Amazon Linux 2023 インスタンスで正常に実行されていることを確認します。

4. 必要に応じて Amazon Linux 2 コンテナインスタンスをドレインして、タスクの移行を高速化します。Amazon Linux 2023 に置き換える十分な容量がある場合は、Amazon ECS コンソールまたは AWS CLI を使用して Amazon Linux 2 コンテナインスタンスを手動でドレインすることで、Amazon Linux 2 から Amazon Linux 2023 へのタスクの移行を高速化できます。移行が完了したら、クラスターから Amazon Linux 2 キャパシティプロバイダーを削除し、関連する Auto Scaling グループを削除します。

Amazon EC2 Auto Scaling グループを使用した移行

1. 新しい起動テンプレートを使用して、新しい Amazon EC2 Auto Scaling グループを作成します。これは既存の起動テンプレートと類似しているはずですが、Amazon Linux 2 Amazon ECS 最適化 AMI の代わりに、Amazon Linux 2023 バリアントのいずれかを指定する必要があります。この新しい Auto Scaling グループは、既存のクラスターにインスタンスを起動できます。
2. Auto Scaling グループをスケールアップして、Amazon Linux 2023 インスタンスがクラスターに登録されるようにします。インスタンスが正しく登録されていること、タスクが新しいインスタンスで正常に実行できることを確認します。
3. タスクが Amazon Linux 2023 で動作することを検証したら、Amazon Linux 2023 Auto Scaling グループをスケールアップしつつ、Amazon Linux 2 Auto Scaling グループを徐々にスケールダウンし、すべての Amazon Linux 2 インスタンスが完全に置き換えられるまで継続します。
4. Amazon Linux 2023 に置き換える十分な容量がある場合は、コンテナインスタンスを明示的にドレインすることで、Amazon Linux 2 から Amazon Linux 2023 へのタスクの移行を高速化できます。詳細については、「Amazon ECS コンテナインスタンスをドレインする」を参照してください。

手動マネージドインスタンスを使用した移行

1. Amazon Linux 2 の代わりに Amazon ECS に最適化された Amazon Linux 2023 AMI を使用して、新しい Amazon EC2 インスタンスを手動で起動 (または起動するスクリプトを調整) します。これらのインスタンスが既存の Amazon Linux 2 インスタンスと同じセキュリティグループ、サブネット、IAM ロール、クラスター設定を使用していることを確認します。インスタンスは、起動時に既存の Amazon ECS クラスターに自動的に登録される必要があります。
2. 新しい Amazon Linux 2023 インスタンスが Amazon ECS クラスターに正常に登録され、`ACTIVE` 状態であることを確認します。自然にタスクが配置されるのを待つか、一部のタスクを手動で停止/開始してスケジュール変更を開始することで、これらの新しいインスタンスでタスクを適切にスケジュールして実行できることをテストします。

3. 必要に応じて追加の Amazon Linux 2023 インスタンスを起動し、Amazon Linux 2 インスタンスを手動でドレインして 1 つずつ終了することで、Amazon Linux 2 インスタンスを段階的に置き換えます。インスタンスを DRAINING ステータスに設定することで、Amazon ECS コンソールからインスタンスをドレインできます。これにより、インスタンスへの新しいタスクの配置が停止し、既存のタスクの終了や別の場所への再スケジュールが可能になります。

Amazon ECS 最適化 Linux AMI のビルドスクリプト

Amazon ECS は、Amazon ECS に最適化された AMI の Linux バリアントの構築に使用するビルドスクリプトをオープンソース化しました。これらのビルドスクリプトは、現在 GitHub で入手できます。詳細については、GitHub の「amazon-ecs-ami」を参照してください。

Amazon ECS に最適化された AMI をカスタマイズする必要がある場合は、GitHub の「Amazon ECS Optimized AMI Build Recipies」を参照してください。

ビルドスクリプトレポジトリには、HashiCorp Packer テンプレートと、Amazon ECS に最適化された AMI の各 Linux バリアントを生成するためのビルドスクリプトが含まれています。ここに置かれているものは、Amazon ECS 最適化 AMI のビルド用として最も信頼できるソーススクリプトであるため、GitHub リポジトリの状態をフォローすることで、AMI に対し実施された変更をモニタリングできます。例えば、独自の AMI で、Amazon ECS チームが正式な AMI に使用するのと同じバージョンの Docker を使用できます。

詳細については、GitHub の Amazon ECS AMI リポジトリにある「aws/amazon-ecs-ami」を参照してください。

Amazon ECS に最適化された Linux AMI を構築するには

1. aws/amazon-ecs-ami GitHub リポジトリのクローンを作成する

```
git clone https://github.com/aws/amazon-ecs-ami.git
```

2. AMI の作成時に使用する AWS リージョンの環境変数を追加します。us-west-2 値を使用するリージョンに置き換えます。

```
export REGION=us-west-2
```

3. AMI を構築するための Makefile が提供されます。クローンされたリポジトリのルートディレクトリから、構築する Amazon ECS 最適化 AMI の Linux バリアントに対応する次のコマンドのいずれかを使用します。

- Amazon ECS に最適化された Amazon Linux 2 AMI

```
make al2
```

- Amazon ECS に最適化された Amazon Linux 2(arm64) AMI

```
make al2arm
```

- Amazon ECS GPU に最適化された AMI

```
make al2gpu
```

- Amazon ECS 最適化 Amazon Linux 2 (Neuron) AMI

```
make al2inf
```

- Amazon ECS 最適化 Amazon Linux 2023 AMI

```
make al2023
```

- Amazon ECS 最適化 Amazon Linux 2023 (arm64) AMI

```
make al2023arm
```

- Amazon ECS 最適化 Amazon Linux 2023 GPU AMI

```
make al2023gpu
```

- Amazon ECS 最適化 Amazon Linux 2023 (Neuron) AMI

```
make al2023neu
```

Amazon ECS 最適化 Bottlerocket AMI

Bottlerocket は Linux ベースのオープンソースのオペレーティングシステムで、仮想マシンやベアメタルホスト上でコンテナを実行するために AWS によって専用に構築されています。Amazon ECS 最適化 Bottlerocket AMI は安全で、コンテナの実行に必要な最小数のパッケージのみが含まれています。これは、リソースの使用量を改善し、セキュリティのアタックサーフェスを縮小し、管理オー

バーヘッドを削減するのに役立ちます。Bottlerocket AMI は Amazon ECS とも統合するため、クラスター内のコンテナインスタンスの更新に伴う運用上のオーバーヘッドを削減するのに役立ちます。

Bottlerocket は、次の点で Amazon Linux とは異なります。

- Bottlerocket にはパッケージマネージャーが含まれておらず、そのソフトウェアはコンテナとしてのみ実行できます。Bottlerocket に対する更新は、1 つのステップで適用され、1 つのステップでロールバックできるため、更新エラーの可能性が低くなります。
- Bottlerocket ホストを管理する主なメカニズムでは、コンテナスケジューラを使用します。Amazon Linux とは異なり、個別の Bottlerocket インスタンスへのログインは、高度なデバッグとトラブルシューティングのみを目的として、低頻度で実行される操作であることを想定しています。

Bottlerocket の詳細については、「GitHub」の「ドキュメント」と「リリース」を参照してください。

Amazon ECS に最適化された Bottlerocket AMI には、カーネル 6.1 用とカーネル 5.10 用のバリアントがあります。

以下のバリアントはカーネル 6.1 を使用します。

- `aws-ecs-2`
- `aws-ecs-2-nvidia`

以下のバリアントはカーネル 5.10 を使用します。

- `aws-ecs-1`
- `aws-ecs-1-nvidia`

  `aws-ecs-1-nvidia` バリアントの詳細については、「Amazon ECS での Bottlerocket 向け NVIDIA GPU サポートの発表」を参照してください。

考慮事項

Amazon ECS で Bottlerocket AMI を使用する場合は、次の点を考慮してください。

- Bottlerocket は、`x86_64` と `arm64` プロセッサを搭載した Amazon EC2 インスタンスをサポートします。Bottlerocket AMI を Inferentia チップを搭載した Amazon EC2 インスタンスで使用することはお勧めしません。
- Bottlerocket イメージには、SSH サーバーまたはシェルは含まれません。ただし、帯域外管理ツールを使用して、SSH 管理者アクセスを取得し、ブートストラップを実行できます。

  詳細については、「GitHub」の「bottlerocket README.md」のセクションを参照してください。

  - 探査
  - 管理者コンテナ
- デフォルトで、Bottlerocket には有効になっているコントロールコンテナがあります。このコンテナは、Amazon EC2 Bottlerocket インスタンス上でのコマンドの実行やシェルセッションの開始に使用できる AWS Systems Manager エージェントを実行します。詳細については、AWS Systems Manager ユーザーガイドの Session Manager のセットアップを参照してください。

- Bottlerocketはコンテナのワークロード向けに最適化されており、セキュリティに重点を置いています。Bottlerocket にはパッケージマネージャーが含まれておらず、イミュータブルです。

  セキュリティ機能とガイダンスについては、GitHub の「セキュリティ機能」および「セキュリティガイダンス」を参照してください。
- `awsvpc` ネットワークモードは、`1.1.0` 以降の Bottlerocket AMI バージョンでサポートされています。
- タスク定義の App Mesh は Bottlerocket AMI バージョン `1.15.0` 以降でサポートされています。
- タスク定義パラメータ `initProcessEnabled` は Bottlerocket AMI バージョン `1.19.0` 以降でサポートされています。
- Bottlerocket AMI は、次のサービスと機能もサポートしていません。
  - ECS Anywhere
  - Service Connect
  - 暗号化モードの Amazon EFS
  - `awsvpc` ネットワークモードの Amazon EFS
  - Amazon EBS ボリュームをマウントできない
  - Elastic Inference アクセラレーター

Amazon ECS に最適化された Bottlerocket AMI メタデータを取得する

AWS Systems Manager パラメータストア API をクエリすることで、Amazon ECS に最適化された AMI の Amazon マシンイメージ (AMI) ID を取得できます。このパラメータを使用することで、Amazon ECS 最適化 AMI ID を手動で検索する必要がなくなります。Systems Manager Parameter Store API の詳細については、GetParameter を参照してください。使用するユーザーには、Amazon ECS 最適化 AMI メタデータを取得するための `ssm:GetParameter` IAM 許可が必要です。

aws-ecs-2 Bottlerocket AMI バリアント

AWS CLI または AWS マネジメントコンソール を使用して、AWS リージョン およびアーキテクチャ別に、最新の安定した `aws-ecs-2` Bottlerocket AMI バリアントを取得できます。

- AWS CLI – サブパラメータ `image_id` を指定しながら次の AWS CLI コマンドを使用することで、推奨される最新の Amazon ECS 最適化 Bottlerocket AMI のイメージ ID を取得できます。*`region`* を、AMI ID が必要な地域コードに置き換えます。

  サポートされている AWS リージョン については、GitHub での「AMI の検索」を参照してください。最新バージョン以外を取得するには、`latest` をバージョン番号に置き換えてください。

  - 64 ビット (x86_64) アーキテクチャの場合:

    ```
    aws ssm get-parameter --region us-east-2 --name "/aws/service/bottlerocket/aws-ecs-2/x86_64/latest/image_id" --query Parameter.Value --output text
    ```

  - 64 ビット Arm (`arm64`) アーキテクチャの場合:

    ```
    aws ssm get-parameter --region us-east-2 --name "/aws/service/bottlerocket/aws-ecs-2/arm64/latest/image_id" --query Parameter.Value --output text
    ```

- AWS マネジメントコンソール –で URL を使用して、推奨される Amazon ECS 最適化 AMI ID をクエリできます。。AWS マネジメントコンソールこの URL により、Amazon EC2 Systems Manager コンソールが開き、パラメータに対応した ID 値が表示されます 次の URL で、*`region`* を、AMI ID が必要なリージョンコードに置き換えます。

  サポートされている AWS リージョン については、GitHub での「AMI の検索」を参照してください。

  - 64 ビット (x86_64) アーキテクチャの場合:

```
https://console.aws.amazon.com/systems-manager/parameters/aws/service/bottlerocket/
aws-ecs-2/x86_64/latest/image_id/description?region=region#
```

- 64 ビット Arm (arm64) アーキテクチャの場合:

```
https://console.aws.amazon.com/systems-manager/parameters/aws/service/bottlerocket/
aws-ecs-2/arm64/latest/image_id/description?region=region#
```

aws-ecs-2-nvidia Bottlerocket AMI バリアント

AWS CLI または AWS マネジメントコンソール を使用して、リージョンおよびアーキテクチャ別に、最新の安定した aws-ecs-2-nvdia Bottlerocket AMI バリアントを取得できます。

- AWS CLI – サブパラメータ image_id を指定しながら次の AWS CLI コマンドを使用することで、推奨される最新の Amazon ECS 最適化 Bottlerocket AMI のイメージ ID を取得できます。*region* を、AMI ID が必要な地域コードに置き換えます。

  サポートされている AWS リージョン については、GitHub での「AMI の検索」を参照してください。最新バージョン以外を取得するには、latest をバージョン番号に置き換えてください。

  - 64 ビット (x86_64) アーキテクチャの場合:

```
aws ssm get-parameter --region us-east-1 --name "/aws/service/bottlerocket/aws-
ecs-2-nvidia/x86_64/latest/image_id" --query Parameter.Value --output text
```

  - 64 ビット Arm (arm64) アーキテクチャの場合:

```
aws ssm get-parameter --region us-east-1 --name "/aws/service/bottlerocket/aws-
ecs-2-nvidia/arm64/latest/image_id" --query Parameter.Value --output text
```

- AWS マネジメントコンソール –で URL を使用して、推奨される Amazon ECS 最適化 AMI ID をクエリできます。。AWS マネジメントコンソールこの URL により、Amazon EC2 Systems Manager コンソールが開き、パラメータに対応した ID 値が表示されます 次の URL で、*region* を、AMI ID が必要なリージョンコードに置き換えます。

  サポートされている AWS リージョン については、GitHub での「AMI の検索」を参照してください。

  - 64 ビット (x86_64) アーキテクチャの場合:

```
https://regionconsole.aws.amazon.com/systems-manager/parameters/aws/service/
bottlerocket/aws-ecs-2-nvidia/x86_64/latest/image_id/description?region=region#
```

- 64 ビット Arm (arm64) アーキテクチャの場合:

```
https://regionconsole.aws.amazon.com/systems-manager/parameters/aws/service/
bottlerocket/aws-ecs-2-nvidia/arm64/latest/image_id/description?region=region#
```

aws-ecs-1 Bottlerocket AMI バリアント

AWS CLI または AWS マネジメントコンソール を使用して、AWS リージョン およびアーキテクチャ別に、最新の安定した aws-ecs-1 Bottlerocket AMI バリアントを取得できます。

- AWS CLI – サブパラメータ image_id を指定しながら次の AWS CLI コマンドを使用することで、推奨される最新の Amazon ECS 最適化 Bottlerocket AMI のイメージ ID を取得できます。*region* を、AMI ID が必要な地域コードに置き換えます。

  サポートされている AWS リージョン については、GitHub での「AMI の検索」を参照してください。最新バージョン以外を取得するには、latest をバージョン番号に置き換えてください。

  - 64 ビット (x86_64) アーキテクチャの場合:

```
aws ssm get-parameter --region us-east-1 --name "/aws/service/bottlerocket/aws-
ecs-1/x86_64/latest/image_id" --query Parameter.Value --output text
```

  - 64 ビット Arm (arm64) アーキテクチャの場合:

```
aws ssm get-parameter --region us-east-1 --name "/aws/service/bottlerocket/aws-
ecs-1/arm64/latest/image_id" --query Parameter.Value --output text
```

- AWS マネジメントコンソール –で URL を使用して、推奨される Amazon ECS 最適化 AMI ID をクエリできます。。AWS マネジメントコンソールこの URL により、Amazon EC2 Systems Manager コンソールが開き、パラメータに対応した ID 値が表示されます 次の URL で、*region* を、AMI ID が必要なリージョンコードに置き換えます。

  サポートされている AWS リージョン については、GitHub での「AMI の検索」を参照してください。

  - 64 ビット (x86_64) アーキテクチャの場合:

```
https://region.console.aws.amazon.com/systems-manager/parameters/aws/service/
bottlerocket/aws-ecs-1/x86_64/latest/image_id/description
```

- 64 ビット Arm (arm64) アーキテクチャの場合:

```
https://region.console.aws.amazon.com/systems-manager/parameters/aws/service/
bottlerocket/aws-ecs-1/arm64/latest/image_id/description
```

aws-ecs-1-nvidia Bottlerocket AMI バリアント

AWS CLI または AWS マネジメントコンソール を使用して、リージョンおよびアーキテクチャ別に、最新の安定した aws-ecs-1-nvdia Bottlerocket AMI バリアントを取得できます。

- AWS CLI – サブパラメータ image_id を指定しながら次の AWS CLI コマンドを使用することで、推奨される最新の Amazon ECS 最適化 Bottlerocket AMI のイメージ ID を取得できます。*region* を、AMI ID が必要な地域コードに置き換えます。

  サポートされている AWS リージョン については、GitHub での「AMI の検索」を参照してください。

  - 64 ビット (x86_64) アーキテクチャの場合:

```
aws ssm get-parameter --region us-east-1 --name "/aws/service/bottlerocket/aws-
ecs-1-nvidia/x86_64/latest/image_id" --query Parameter.Value --output text
```

  - 64 ビット Arm (arm64) アーキテクチャの場合:

```
aws ssm get-parameter --region us-east-1 --name "/aws/service/bottlerocket/aws-
ecs-1-nvidia/arm64/latest/image_id" --query Parameter.Value --output text
```

- AWS マネジメントコンソール –で URL を使用して、推奨される Amazon ECS 最適化 AMI ID をクエリできます。。AWS マネジメントコンソールこの URL により、Amazon EC2 Systems Manager コンソールが開き、パラメータに対応した ID 値が表示されます 次の URL で、*region* を、AMI ID が必要なリージョンコードに置き換えます。

  サポートされている AWS リージョン については、GitHub での「AMI の検索」を参照してください。

  - 64 ビット (x86_64) アーキテクチャの場合:

```
https://console.aws.amazon.com/systems-manager/parameters/aws/service/bottlerocket/
aws-ecs-1-nvidia/x86_64/latest/image_id/description?region=region#
```

- 64 ビット Arm (arm64) アーキテクチャの場合:

```
https://console.aws.amazon.com/systems-manager/parameters/aws/service/bottlerocket/
aws-ecs-1-nvidia/arm64/latest/image_id/description?region=region#
```

次のステップ

Amazon ECS で Bottlerocket オペレーティングシステムの使用を開始する方法の詳細なチュートリアルについては、GitHub の「Using a Bottlerocket AMI with Amazon ECS」および AWS ブログサイトの「Getting started with Bottlerocket and Amazon ECS」を参照してください。

Bottlerocket インスタンスを起動する方法については、「Amazon ECS の Bottlerocket インスタンスの起動」を参照してください。

Amazon ECS の Bottlerocket インスタンスの起動

Bottlerocket インスタンスを起動することで、コンテナワークロードを実行することができます。

AWS CLI を使用して Bottlerocket インスタンスを起動できます。

1. `userdata.toml` というファイルを作成します。このファイルは、インスタンスのユーザーデータに使用されます。*`cluster-name`* をクラスターの名前に置き換えます。

```
[settings.ecs]
cluster = "cluster-name"
```

2. the section called “Amazon ECS に最適化された Bottlerocket AMI メタデータを取得する” に含まれているコマンドのいずれかを使用して、Bottlerocket AMI ID を取得します。これは次のステップで使用します。
3. 次のコマンドを実行して、Bottlerocket インスタンスを起動します。次のパラメータを必ず置き換えてください。
   - *`subnet`* を、インスタンスを起動するプライベートまたはパブリックサブネットの ID に置き換えます。
   - *`bottlerocket_ami`* を、前のステップの AMI ID に置き換えます。

- *t3.large* を、使用するインスタンスタイプに置き換えます。
- *region* を、リージョンコードに置き換えます。

```
aws ec2 run-instances --key-name ecs-bottlerocket-example \
   --subnet-id subnet \
   --image-id bottlerocket_ami \
   --instance-type t3.large \
   --region region \
   --tag-specifications
 'ResourceType=instance,Tags=[{Key=bottlerocket,Value=example}]' \
   --user-data file://userdata.toml \
   --iam-instance-profile Name=ecsInstanceRole
```

4. 次のコマンドを実行して、コンテナインスタンスがクラスターに登録されていることを検証します。このコマンドを実行するときは、次のパラメータを必ず置き換えてください。

   - *cluster* を、自分のクラスター名に置き換えます。
   - *region* を、リージョンコードに置き換えます。

```
aws ecs list-container-instances --cluster cluster-name --region region
```

Amazon ECS で Bottlerocket オペレーティングシステムの使用を開始する方法の詳細なチュートリアルについては、GitHub の「Using a Bottlerocket AMI with Amazon ECS」および AWS ブログサイトの「Getting started with Bottlerocket and Amazon ECS」を参照してください。

Amazon ECS Linux コンテナインスタンスの管理

Amazon ECS ワークロードに EC2 インスタンスを使用する場合、インスタンスの維持はユーザーの責任となります。

管理手順

- Amazon ECS Linux コンテナインスタンスの起動
- Amazon ECS Linux コンテナインスタンスをブートストラップしてデータを渡す
- スポットインスタンス通知を受信するように Amazon ECS Linux コンテナインスタンスを設定する
- Amazon ECS Linux コンテナインスタンスの起動時にスクリプトを実行する

- Amazon ECS Linux コンテナインスタンスのネットワークインターフェイスを増やす
- Amazon ECS Linux コンテナインスタンスのメモリを予約する
- AWS Systems Manager を使用して Amazon ECS コンテナインスタンスをリモートで管理する
- Amazon ECS Linux コンテナインスタンスに HTTP プロキシを使用する
- Amazon ECS Auto Scaling グループ用に事前初期化されたインスタンスを設定する
- Amazon ECS コンテナエージェントをアップデートする

Amazon ECSコンテナエージェントの各バージョンは、異なる機能セットをサポートし、以前のバージョンのバグ修正を提供します。可能であれば、最新バージョンの Amazon ECSコンテナエージェントを使用することを常にお勧めします。コンテナエージェントを最新バージョンに更新するには、「Amazon ECS コンテナエージェントをアップデートする」を参照してください。

どの機能と拡張が各エージェントリリースに含まれているか確認するには、https://github.com/aws/amazon-ecs-agent/releases を参照してください。

> **Important**
>
> 信頼できるメトリクスの最小 Docker バージョンは、Amazon ECS 最適化 AMI 20220607 以降に含まれる Docker バージョン v20.10.13 以降です。
> バージョン 1.20.0 以降のAmazon ECSエージェントでは、18.01.0 より前のバージョンの Docker のサポートが廃止されました。

Amazon ECS Linux コンテナインスタンスの起動

Amazon EC2 コンソールを使用して Amazon ECS コンテナインスタンスを作成できます。

インスタンスは、Amazon EC2 コンソール、AWS CLI、SDK など、さまざまな方法で起動できます。インスタンスを起動する他の方法については、「Amazon EC2 ユーザーガイド」の「インスタンスを起動する」を参照してください。

起動ウィザードの詳細については、「Amazon EC2 ユーザーガイド」の「新しいインスタンス起動ウィザードを使用してインスタンスを起動する」を参照してください。

開始する前に、「Amazon ECS を使用するようにセットアップする」のステップを完了します。

新しい Amazon EC2 ウィザードを使用してインスタンスを起動できます。インスタンス起動ウィザードでは、インスタンスの起動に必要な起動パラメータを指定します。

インスタンス設定のパラメータ

- 手順
- 名前とタグ
- アプリケーションと OS イメージ (Amazon マシンイメージ)
- インスタンスタイプ
- キーペア (ログイン)
- ネットワーク設定
- ストレージの設定
- 高度な詳細

手順

開始する前に、「Amazon ECS を使用するようにセットアップする」のステップを完了します。

1. Amazon EC2 コンソールの https://console.aws.amazon.com/ec2/ を開いてください。
2. 画面の上のナビゲーションバーで、現在の AWS リージョンが表示されます (例: 米国東部 (オハイオ))。インスタンスを起動するリージョンを選択します。
3. Amazon EC2 コンソールダッシュボードで、[インスタンスを起動] を選択してください。

名前とタグ

インスタンス名はタグで、キーは [Name] (名前)、値は指定した名前です。インスタンス、ボリューム、および伸縮自在なグラフィックスにタグ付けできます。スポットインスタンスの場合、スポットインスタンスリクエストにのみタグを付けることができます。

インスタンス名と追加のタグを指定することはオプションです。

- [Name] (名前) に、インスタンスのわかりやすい名前を入力します。名前を指定しない場合は、インスタンスをその ID で識別できます。ID は、インスタンスの起動時に自動的に生成されます。
- タグを追加するには、[Add additional tag] (追加のタグを追加) を選択します。[Add tag] (タグを追加) を選択し、キーと値を入力し、タグ付けするリソースタイプを選択します。追加するタグごとに [Add tag] (タグの追加) を選択します。

アプリケーションと OS イメージ (Amazon マシンイメージ)

Amazon マシンイメージ (AMI) には、インスタンスの作成に必要な情報が含まれています。例えば、ある AMI には、ウェブサーバーとして動作するために必要なソフトウェア (Apache やウェブサイトなど) が含まれています。

[Search] (検索) バーを使用して、AWS によって発行された適切な Amazon ECS 最適化 AMI を検索します。

1. [Search] (検索) バーに次のいずれかの用語を入力します。
   - **`ami-ecs`**
   - Amazon ECS 最適化 AMI の [Value] (値)。

     最新の Amazon ECS 最適化 AMI とその値については、「Linux Amazon ECS に最適化された AMI」を参照してください。
2. [Enter] キーを押します。
3. [Choose an Amazon Machine Image (AMI)] (Amazon マシンイメージ (AMI) を選択) ページで、[AWS Marketplace AMIs] タブを選択します。
4. 左側の [結果を絞り込む] ペインから、[Amazon Web Services] を [パブリッシャー] として選択します。
5. 使用する AMI の行で [Select] (選択) を選択します。

   または、[Cancel] (キャンセル) を選択することで、AMI を選択せずにインスタンス起動ウィザードに戻ります。デフォルトの AMI が選択されます。AMI が、「Amazon ECS に最適化された Linux AMI」で概説されている要件を満たしていることを確認します。

インスタンスタイプ

インスタンスタイプは、インスタンスのハードウェア設定とサイズを定義します。インスタンスタイプが大きくなると、CPU およびメモリも増えます。詳細については、「Amazon EC2 ユーザーガイド」の「インスタンスタイプ」を参照してください。IPv6 のみのワークロードを実行する場合、一部のインスタンスタイプでは IPv6 アドレスがサポートされません。詳細については、Amazon EC2 ユーザーガイドの IPv6 アドレスを参照してください。

- [Instance type] (インスタンスタイプ) で、インスタンスのインスタンスタイプを選択します。

  選択したインスタンスタイプによって、タスクの実行に使用できるリソースが決まります。

キーペア (ログイン)

[Key pair name] (キーペア名) には、既存のキーペアを選択するか、[Create new key pair] (新しいキーペアを作成) を選択して新しいキーペアを作成します。

> **Important**
>
> [Proceed without key pair] (キーペアなしで進む) オプションを選択した場合 (非推奨)、ユーザーが別の方法でログインすることを許可するように設定された AMI を選択した場合でなければ、インスタンスに接続できなくなります。

ネットワーク設定

フォームのネットワーク設定セクションの [編集] ボタンを選択した後、必要に応じて [ネットワーク設定] を構成します。

- [VPC] で、DB インスタンスを起動する先の VPC を選択します。IPv6 のみのワークロードを実行するには、IPv4 と IPv6 CIDR ブロックの両方を含むデュアルスタック VPC を選択します。
- [サブネット] で、インスタンスを起動するサブネットを選択します。インスタンスは、アベイラビリティーゾーン、ローカルゾーン、Wavelength Zone、Outpost のいずれかに関連付けられたサブネットで起動できます。

  アベイラビリティーゾーンでインスタンスを起動するには、インスタンスを起動するサブネットを選択します。新しいサブネットを作成するには、[Create new subnet] を選択して Amazon VPC コンソールに移動します。終了したらインスタンス起動ウィザードに戻り、[Refresh] (更新) アイコンを選択して一覧にサブネットを読み込みます。

  ローカルゾーンでインスタンスを起動するには、ローカルゾーン内に作成したサブネットを選択します。

  アウトポストでインスタンスを起動するには、アウトポストに関連付けられた VPC 内のサブネットを選択します。

  IPv6 のみのワークロードを実行するには、IPv6 CIDR ブロックのみを含むサブネットを選択します。
- [Auto-assign Public IP] (パブリック IP の自動割り当て): インスタンスをインターネットからアクセス可能にする場合は、[Auto-assign Public IP] (パブリック IP の自動割り当て) フィールドが

[Enable] (有効) に設定されていることを確認します。可能にしない場合には、このフィールドを [無効] に設定します。

> **Note**
>
> コンテナインスタンスには、Amazon ECS サービスエンドポイントと通信するためのアクセスが必要です。これは、インターフェイス VPC エンドポイントを介して、またはパブリック IP アドレスを持つコンテナインスタンスを通じて可能になります。
> インターフェイス VPC エンドポイントについての詳細は、「Amazon ECS とインターフェイス VPC エンドポイント (AWS PrivateLink)」を参照してください。
> インターフェイス VPC エンドポイントが設定されておらず、コンテナインスタンスがパブリック IP アドレスを持たない場合、ネットワークアドレス変換 (NAT) を使用してこのアクセスを提供する必要があります。詳細については、「Amazon VPC ユーザーガイド」の「NAT ゲートウェイ」、およびこのガイドの「Amazon ECS Linux コンテナインスタンスに HTTP プロキシを使用する」を参照してください。

- デュアルスタック VPC と IPv6 のみのサブネットを選択した場合、[自動割り当て IPv6 IP] で [有効化] を選択します。
- [Firewall (security groups)] (ファイアウォール (セキュリティグループ)): セキュリティグループを使用して、コンテナインスタンスのファイアウォールルールを定義します。このルールでは、どの着信ネットワークトラフィックをコンテナインスタンスに配信するかを指定します。他のトラフィックはすべて無視されます。
  - 既存のセキュリティグループを選択するには、[Select existing security group] (既存のセキュリティグループを選択) を選択し、Amazon ECS を使用するようにセットアップする で作成したセキュリティグループを選択します。
- IPv6 のみのワークロードのインスタンスを起動する場合は、[高度なネットワーク設定] を選択し、[プライマリ IPv6 IP の割り当て] に [はい] を選択します。

> **Note**
>
> プライマリ IPv6 アドレスがない場合、ホストまたはブリッジネットワークモードのコンテナインスタンスで実行されているタスクは、ロードバランサーまたは AWS Cloud Map に登録されません。

ストレージの設定

選択した AMI には、ルートボリュームを含む、1 つまたは複数のストレージボリュームが含まれます。インスタンスにアタッチする追加のボリュームを指定できます。

[Simple] (シンプル) ビューを使用できます。

- [Storage type] (ストレージタイプ): コンテナインスタンスのストレージを設定します。

  Amazon ECS に最適化された Amazon Linux 2 AMI を使用している場合、1 つの 30 GiB ボリュームがインスタンスに設定されます。これは、オペレーティングシステムと Docker の間で共有されます。

  Amazon ECS に最適化された AMI を使用している場合、インスタンスには 2 つのボリュームが設定されます。[Root] ボリュームはオペレーティングシステム用で、2 番目の Amazon EBS ボリューム (`/dev/xvdcz` にアタッチ) は Docker 用です。

  オプションで、アプリケーションのニーズに合わせてインスタンスのボリュームサイズを増減できます。

高度な詳細

[Advanced details] で、セクションを開いてフィールドを表示し、インスタンスの追加パラメータを指定します。

- [Purchasing option] (購入のオプション): [Request Spot Instances] (スポットインスタンスのリクエスト) を選択して、スポットインスタンスをリクエストします。また、スポットインスタンスに関連する他のフィールドも設定する必要があります。詳細については、「スポットインスタンスのリクエスト」を参照してください。

  > **Note**
  >
  > スポットインスタンスを使用していて、"`Not available`" メッセージが表示される場合は、別のインスタンスタイプを選択する必要があります。

- [IAM instance profile] (IAM インスタンスプロフィール): コンテナインスタンス IAM ロールを選択します。通常、これは `ecsInstanceRole` という名前です。

> **Important**
>
> 適切な IAM アクセス許可を使用してコンテナインスタンスを起動しないと、Amazon ECS エージェントはクラスターに接続できません。詳細については、「Amazon ECS コンテナインスタンスの IAM ロール」を参照してください。

- [ユーザーデータ]: Amazon ECS コンテナエージェントの設定からのエージェント環境変数のようなユーザーデータを使用して、Amazon ECS コンテナインスタンスを設定します。Amazon EC2 ユーザーデータスクリプトはインスタンスの初回起動時に 1 回のみ実行されます。以下に、ユーザーデータを使用する目的の一般的な例を紹介します。
  - デフォルトでは、コンテナインスタンスはデフォルトのクラスターで起動されます。デフォルト以外のクラスターで起動するには、[Advanced Details] (高度な詳細) リストを選択します。次に、[User data] フィールドに以下のスクリプトを貼り付け、*your_cluster_name* を使用するクラスターの名前に置き換えます。

```
#!/bin/bash
echo ECS_CLUSTER=your_cluster_name >> /etc/ecs/ecs.config
```

  - Amazon S3 に ecs.config ファイルがあり、コンテナインスタンスロールへの Amazon S3 読み取り専用アクセスを有効にしている場合は、[高度な詳細] リストを選択します。次に、[User data (ユーザーデータ)] フィールドに以下のスクリプトを貼り付け、*your_bucket_name* を、AWS CLI をインストールして起動時に設定ファイルを書き込むバケットに置き換えます。

> **Note**
>
> この設定の詳細については、「Amazon S3 に Amazon ECS コンテナインスタンスの設定を保存する」を参照してください。

```
#!/bin/bash
yum install -y aws-cli
aws s3 cp s3://your_bucket_name/ecs.config /etc/ecs/ecs.config
```

  - ECS_CONTAINER_INSTANCE_TAGS 設定パラメータを使用して、コンテナインスタンスのタグを指定します。これにより Amazon ECS にのみ関連付けられているタグが作成され、そのタグは Amazon EC2 API ではリスト取得できません。

> **Important**
>
> Amazon EC2 Auto Scaling グループを使用してコンテナインスタンスを起動する場合は、ECS_CONTAINER_INSTANCE_TAGS エージェント設定パラメータを使用してタグを追加する必要があります。これは、Auto Scaling グループを使用して起動される Amazon EC2 インスタンスにタグを追加する方法によるものです。

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=your_cluster_name
ECS_CONTAINER_INSTANCE_TAGS={"tag_key": "tag_value"}
EOF
```

- コンテナインスタンスのタグを指定してから、ECS_CONTAINER_INSTANCE_PROPAGATE_TAGS_FROM 設定パラメータを使用してそれらを Amazon EC2 から Amazon ECS に伝播します。

  次に、コンテナインスタンスに関連付けられたタグを伝達し、さらに your_cluster_name という名前のクラスターでコンテナインスタンスを登録するユーザーデータスクリプトの例を示します。

  ```
  #!/bin/bash
  cat <<'EOF' >> /etc/ecs/ecs.config
  ECS_CLUSTER=your_cluster_name
  ECS_CONTAINER_INSTANCE_PROPAGATE_TAGS_FROM=ec2_instance
  EOF
  ```

- デフォルトでは、Amazon ECS コンテナエージェントは、インスタンスのデフォルト IPv4 および IPv6 ルートを確認することで、そのコンテナインスタンスが IPv6-only 設定に対応しているかどうかを検出します。この動作を上書きするには、インスタンスの /etc/ecs/ecs.config ファイルで ECS_INSTANCE_IP_COMPATIBILITY パラメータを ipv4 または ipv6 に設定してください。

  ```
  #!/bin/bash
  cat <<'EOF' >> /etc/ecs/ecs.config
  ECS_CLUSTER=your_cluster_name
  ECS_INSTANCE_IP_COMPATIBILITY=ipv6
  ```

```
EOF
```

詳細については、「Amazon ECS Linux コンテナインスタンスをブートストラップしてデータを渡す」を参照してください。

Amazon ECS Linux コンテナインスタンスをブートストラップしてデータを渡す

Amazon EC2 インスタンスを起動するときに、ユーザーデータを EC2 インスタンスに渡すことができます。インスタンスの起動時に、データを使って、一般的な自動設定タスクを実行したり、スクリプトを実行したりできます。Amazon ECS では、ユーザーデータの最も一般的なユースケースは、設定情報を Docker デーモンと Amazon ECS コンテナエージェントに渡すことです。

クラウドブートフック、シェルスクリプト、cloud-init ディレクティブなど、複数タイプのユーザーデータを Amazon EC2 に渡すことができます。これらおよびその他の形式の種類の詳細については、「Cloud-Init のドキュメント」を参照してください。

Amazon EC2 起動ウィザードの使用時にユーザーデータを渡すには、「Amazon ECS Linux コンテナインスタンスの起動」を参照してください。

コンテナインスタンスがデータを渡すようにするための設定を、コンテナエージェント設定または Docker デーモン設定で行うことができます。

Amazon ECS コンテナエージェント

Amazon ECS に最適化された AMI の Linux バリアントは、コンテナエージェントの開始時に /etc/ecs/ecs.config ファイルでエージェント設定データを検索します。Amazon EC2 ユーザーデータを使用して、起動時に、この設定データを指定することができます。利用可能な Amazon ECS コンテナエージェントの設定変数の詳細については、「Amazon ECS コンテナエージェントの設定」を参照してください。

クラスター名など、単一エージェントの設定変数のみを設定するには、echo を使用して、変数を設定ファイルにコピーします。

```
#!/bin/bash
echo "ECS_CLUSTER=MyCluster" >> /etc/ecs/ecs.config
```

/etc/ecs/ecs.config に書き込む変数が複数ある場合は、以下の heredoc 形式を使用します。この形式は cat で始まる行と EOF の間のすべてを設定ファイルに書き込みます。

```
#!/bin/bash
```

```
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_ENGINE_AUTH_TYPE=docker
ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":
{"username":"my_name","password":"my_password","email":"email@example.com"}}
ECS_LOGLEVEL=debug
ECS_WARM_POOLS_CHECK=true
EOF
```

カスタムインスタンス属性を設定するには、ECS_INSTANCE_ATTRIBUTES 環境変数を設定します。

```
#!/bin/bash
cat <<'EOF' >> ecs.config
ECS_INSTANCE_ATTRIBUTES={"envtype":"prod"}
EOF
```

Docker デーモン

Docker デーモンの構成情報は Amazon EC2 ユーザーデータで指定できます。設定オプションの詳細については、「Docker デーモンのドキュメント」を参照してください。

> **Note**
>
> AWSカスタム Docker 設定は、将来の Amazon ECS の変更や機能と警告なしに競合する可能性があるため、 はサポートしていません。

次の例では、カスタムオプションが Docker デーモン構成ファイル /etc/docker/daemon.json に追加され、インスタンスの起動時にユーザーデータで指定されます。

```
#!/bin/bash
cat <<EOF >/etc/docker/daemon.json
{"debug": true}
EOF
systemctl restart docker --no-block
```

次の例では、カスタムオプションが Docker デーモン構成ファイル /etc/docker/daemon.json に追加され、インスタンスの起動時にユーザーデータで指定されます。この例は、Docker デーモン設定ファイルの docker-proxy を無効にする方法を示します。

```
#!/bin/bash
cat <<EOF >/etc/docker/daemon.json
{"userland-proxy": false}
EOF
systemctl restart docker --no-block
```

スポットインスタンス通知を受信するように Amazon ECS Linux コンテナインスタンスを設定する

利用可能なキャパシティーがなくなった場合、または、スポット料金がお客様のリクエストの上限料金を超えた場合には、Amazon EC2 はスポットインスタンスを終了、停止、または休止状態にします。Amazon EC2 は、終了アクションおよび停止アクションに対して、スポットインスタンスに 2 分間の中断通知を提供します。休止状態のアクションについては、2 分間の通知は提供されません。インスタンスで Amazon ECS スポットインスタンスドレイニングが有効になっている場合、Amazon ECS はスポットインスタンスの中断通知を受け取り、インスタンスを DRAINING ステータスにします。

> **Important**
>
> 自動スケーリングキャパシティの再分散よってインスタンスが削除された場合、Amazon ECS は Amazon EC2 から通知を受信しません。詳細については、「Amazon EC2 Auto Scaling キャパシティの再分散」を参照してください。

コンテナインスタンスを DRAINING に設定すると、Amazon ECS によって新規タスクがそのコンテナインスタンスに配置されなくなります。ドレインしているコンテナインスタンス上にある PENDING 状態のサービスタスクは即時停止されます。クラスター内に使用可能なコンテナインスタンスがある場合、そのインスタンスで代わりのサービスタスクが開始されます。

デフォルトでは、スポットインスタンスドレイニングは無効になっています。

インスタンスの起動時にスポットインスタンスドレイニングを有効にできます。次のスクリプトを [ユーザーデータ] フィールドに追加します。*MyCluster* は、コンテナインスタンスを登録するクラスターの名前に置き換えてください。

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_ENABLE_SPOT_INSTANCE_DRAINING=true
EOF
```

詳細については、「Amazon ECS Linux コンテナインスタンスの起動」を参照してください。

既存のコンテナインスタンスに対してスポットインスタンスのドレインを有効にするには

1. SSH 経由でスポットインスタンスに接続します。
2. /etc/ecs/ecs.config ファイルを編集して、以下を追加します。

```
ECS_ENABLE_SPOT_INSTANCE_DRAINING=true
```

3. ecs サービスを再起動します。
   - Amazon ECS に最適化された Amazon Linux 2 AMI の場合:

```
sudo systemctl restart ecs
```

4. (オプション) エージェント詳細分析 API オペレーションをクエリして、エージェントが実行されていることを確認し、新しいコンテナインスタンスに関する情報の一部を表示できます。詳細については、「the section called “コンテナの詳細分析”」を参照してください。

```
curl http://localhost:51678/v1/metadata
```

Amazon ECS Linux コンテナインスタンスの起動時にスクリプトを実行する

すべてのコンテナインスタンスで特定のコンテナを実行して、モニタリング、セキュリティ、メトリクス、サービス検出、ログ記録などのオペレーションやセキュリティの問題に対処する必要がある場合があります。

これを行うには、起動時にユーザーデータスクリプトで、または Upstart や docker run などの一部の init システムで、systemd コマンドを呼び出すように、コンテナインスタンスを設定できます。これは機能しますが、いくつかの欠点があります。Amazon ECS は、コンテナに関する情報を渡されず、CPU、メモリ、ポートなどのリソースをモニタリングできないためです。Amazon ECS がすべてのタスクリソースについて適切に情報を得られるように、コンテナインスタンスで実行するコンテナのタスク定義を作成します。その後、Amazon ECS を使用して、起動時に Amazon EC2 ユーザーデータを渡してタスクを配置します。

以下の手順の Amazon EC2 ユーザーデータスクリプトは、Amazon ECS イントロスペクション API を使用してコンテナインスタンスを識別します。次に、AWS CLI と start-task コマンドを使用して、指定されたタスクをスタートアップ時に実行します。

コンテナインスタンスの起動時にタスクを開始するには

1. `ecsInstanceRole` IAM ロールを変更して、`StartTask` API オペレーションに対するアクセス権限を追加します。詳細については、「AWS Identity and Access Management ユーザーガイド」の「ロールに対するアクセス許可を更新する」を参照してください。
2. Amazon ECS に最適化された Amazon Linux 2 AMI を使用して、1 つ以上のコンテナインスタンスを起動します。新しいコンテナインスタンスを起動し、EC2 ユーザーデータで次のサンプルスクリプトを使用します。*`your_cluster_name`* は、コンテナインスタンスに登録するクラスターに置き換え、*`my_task_def`* は、起動時にインスタンスで実行するタスク定義に置き換えてください。

   詳細については、「Amazon ECS Linux コンテナインスタンスの起動」を参照してください。

   > **Note**
   >
   > 次の MIME マルチパートの内容では、シェルスクリプトを使用して値を設定し、パッケージをインストールしています。また、systemd ジョブを使用して、ecs サービスが実行されイントロスペクション API が使用可能になった後にタスクが開始されるようにしています。

```
Content-Type: multipart/mixed; boundary="==BOUNDARY=="
MIME-Version: 1.0

--==BOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"

#!/bin/bash
# Specify the cluster that the container instance should register into
cluster=your_cluster_name

# Write the cluster configuration variable to the ecs.config file
# (add any other configuration variables here also)
echo ECS_CLUSTER=$cluster >> /etc/ecs/ecs.config

START_TASK_SCRIPT_FILE="/etc/ecs/ecs-start-task.sh"
cat <<- 'EOF' > ${START_TASK_SCRIPT_FILE}
 exec 2>>/var/log/ecs/ecs-start-task.log
 set -x
```

```
 # Install prerequisite tools
 yum install -y jq aws-cli

 # Wait for the ECS service to be responsive
 until curl -s http://localhost:51678/v1/metadata
 do
  sleep 1
 done

 # Grab the container instance ARN and AWS Region from instance metadata
 instance_arn=$(curl -s http://localhost:51678/v1/metadata | jq -r '.
 | .ContainerInstanceArn' | awk -F/ '{print $NF}' )
 cluster=$(curl -s http://localhost:51678/v1/metadata | jq -r '. | .Cluster' | awk
 -F/ '{print $NF}' )
 region=$(curl -s http://localhost:51678/v1/metadata | jq -r '.
 | .ContainerInstanceArn' | awk -F: '{print $4}')

 # Specify the task definition to run at launch
 task_definition=my_task_def

 # Run the AWS CLI start-task command to start your task on this container instance
 aws ecs start-task --cluster $cluster --task-definition $task_definition --
container-instances $instance_arn --started-by $instance_arn --region $region
EOF

# Write systemd unit file
UNIT="ecs-start-task.service"
cat <<- EOF > /etc/systemd/system/${UNIT}
      [Unit]
      Description=ECS Start Task
      Requires=ecs.service
      After=ecs.service

      [Service]
      Restart=on-failure
      RestartSec=30
      ExecStart=/usr/bin/bash ${START_TASK_SCRIPT_FILE}

      [Install]
      WantedBy=default.target
EOF

# Enable our ecs.service dependent service with `--no-block` to prevent systemd
 deadlock
```

```
# See https://github.com/aws/amazon-ecs-agent/issues/1707
systemctl enable --now --no-block "${UNIT}"
--==BOUNDARY==--
```

3. コンテナインスタンスが正しいクラスターで起動し、タスクが開始されていることを確認します。

   a. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。

   b. ナビゲーションバーから、クラスターがあるリージョンを選択します。

   c. ナビゲーションペインで [Clusters] を選択し、コンテナインスタンスをホストするクラスターを選択します。

   d. [クラスター] ページで [タスク] を選択した後、使用するタスクを選択します。

      起動した各コンテナインスタンスでは、このタスクが実行状態になります。

      タスクが表示されない場合は、SSH を使用してコンテナインスタンスにログインし、`/var/log/ecs/ecs-start-task.log` ファイルでデバッグの情報を確認できます。

Amazon ECS Linux コンテナインスタンスのネットワークインターフェイスを増やす

> **Note**
>
> この機能は Fargate では使用できません。

`awsvpc` ネットワークモードを使用するタスクには、それぞれ独自の Elastic Network Interface (ENI) が提供されます。この ENI は、ENI をホストするコンテナインスタンスにアタッチされています。Amazon EC2 インスタンスにアタッチできるネットワークインターフェイスの数にはデフォルトの制限があり、プライマリネットワークインターフェイスも 1 つとしてカウントされます。例えば、デフォルトでは `c5.large` インスタンスには最大 3 つの ENI がアタッチされています。このインスタンスのプライマリネットワークインターフェイスも 1 つとしてカウントされるため、このインスタンスに追加でアタッチできる ENI は 2 つです。`awsvpc` ネットワークモードを使用する各タスクには ENI が必要なため、通常このインスタンスタイプでは、このようなタスクを 2 つだけ実行できます。

Amazon ECS は、サポートされている Amazon EC2 インスタンスタイプを使用して、ENI 密度が高いコンテナインスタンスの起動をサポートしています。これらのインスタンスタイプを使用し、`awsvpcTrunking` アカウント設定を有効にすると、新しく起動されたコンテナインスタンスで

追加の ENI を利用できます。この設定により、各コンテナインスタンスにより多くのタスクを配置できます。コンソールを使用して機能を有効にするには、「Amazon ECS アカウント設定の変更」を参照してください。AWS CLI を使用して機能を有効にするには、「AWS CLI を使用した Amazon ECS アカウント設定の管理」を参照してください。

例えば、`awsvpcTrunking` を持つ `c5.large` インスタンスでは、ENI の制限が 12 に引き上げられています。コンテナインスタンスはプライマリネットワークインターフェイスを持ち、Amazon ECS はコンテナインスタンスの「トランク」ネットワークインターフェイスを作成およびアタッチします。したがって、この設定では、現在の 2 個ではなく 10 個のタスクをコンテナインスタンスで起動できます。

トランクネットワークインターフェイスは Amazon ECS によって完全に管理され、コンテナインスタンスを削除またはクラスターから登録解除するときに削除されます。詳細については、「EC2 の Amazon ECS タスクネットワークオプション」を参照してください。

考慮事項

ENI トランキング機能を使用する場合は、以下の点を考慮してください。

- ENI の制限引き上げに対応しているのは、Amazon ECS に最適化された AMI の Linux バリアント、バージョン `1.28.1` 以降のコンテナエージェントを備えたその他の Amazon Linux バリアント、バージョン `1.28.1-2` 以降の ecs-init パッケージのみです。Amazon ECS に最適化された AMI の最新の Linux バリアントを使用する場合、これらの要件が満たされます。現時点では、Windows コンテナはサポートされていません。
- `awsvpcTrunking` を有効にした後に起動した新しい Amazon EC2 インスタンスのみに、引き上げられた ENI 制限とトランクネットワークインターフェイスが適用されます。以前に起動されたインスタンスは、実行されたアクションに関係なく、これらの機能を受け取りません。
- Amazon EC2 インスタンスでは、リソースベースの IPv4 DNS リクエストがオフになっている必要があります。このオプションを無効にする場合、Amazon EC2 コンソールをで新しいインスタンスを作成する際に、[Enable resource-based IPV4 (A record) DNS requests](リソースベースの IPV4 (A レコード) DNS リクエストを有効化) オプションの選択を外します。AWS CLI を使ってこのオプションを無効にするには、次のコマンドを使用します。

```
aws ec2 modify-private-dns-name-options --instance-id i-xxxxxxx --no-enable-resource-name-dns-a-record --no-dry-run
```

- 共有サブネットの Amazon EC2 インスタンスはサポートされません。これらを使用すると、クラスターへの登録に失敗します。

- タスクは、awsvpc ネットワークモードと EC2 を使用する必要があります。Fargate を使用するタスクは、起動数に関係なく、常に専用の ENI が割り当てられるため、この機能は必要ありません。
- タスクは、コンテナインスタンスと同じ Amazon VPC で起動する必要があります。タスクが同じ VPC 内にない場合は属性エラーが発生し、タスクを開始することができません。
- 新しいコンテナインスタンスを起動するときに、インスタンスは REGISTERING 状態に移行し、トランク Elastic Network Interface がインスタンスに対してプロビジョニングされます。登録に失敗した場合、インスタンスは REGISTRATION_FAILED 状態に移行します。失敗した登録のトラブルシューティングを行うには、コンテナインスタンスを記述して、失敗の原因を説明する statusReason フィールドを表示するようにします。その後、コンテナインスタンスは手動で登録解除または終了できます。コンテナインスタンスが正常に登録解除または終了すると、Amazon ECS はトランク ENI を削除します。

  > **Note**
  >
  > Amazon ECS は、REGISTRATION_FAILED 状態に移行するインスタンスを監視できるコンテナインスタンスの状態変更イベントを発行します。詳細については、「Amazon ECS コンテナインスタンス状態変更イベント」を参照してください。

- コンテナインスタンスが削除されると、インスタンスは DEREGISTERING 状態に移行し、トランク Elastic Network Interface が解放されます。次に、インスタンスは INACTIVE 状態に移行します。
- ENI 制限が引き上げられたパブリックサブネットのコンテナインスタンスが停止した後再起動されると、インスタンスはパブリック IP アドレスを失い、コンテナエージェントは接続を失います。
- awsvpcTrunking を有効にすると、コンテナインスタンスは VPC のデフォルトセキュリティグループを使用する追加の ENI が割り当てられ、Amazon ECS によって管理されます。

  デフォルト VPC には、各アベイラビリティーゾーンのパブリックサブネット、インターネットゲートウェイ、および DNS 解決を有効にするための設定があります。サブネットはパブリックサブネットです。メインルートテーブルがインターネット用のサブネットのトラフィックをインターネットゲートウェイに送信するためです。デフォルトサブネットをプライベートサブネットにするには、送信元 0.0.0.0/0 からインターネットゲートウェイへのルートを削除します。ただし、この操作を行った場合、そのサブネットで実行されているすべてのコンテナインスタンスがインターネットにアクセスできなくなります。セキュリティグループルールを追加または削除して、サブネットに出入りするトラフィックを制御できます。詳細については、「Amazon Virtual Private Cloud ユーザーガイド」の「セキュリティグループのルール」を参照してください。

前提条件

ENI 制限が引き上げられたコンテナインスタンスを起動する前に、次の前提条件を満たす必要があります。

- Amazon ECS のサービスにリンクされたロールを作成する必要があります。Amazon ECS のサービスにリンクされたロールは、Amazon ECS に、お客様に代わって他のAWSサービスを呼び出すアクセス権限を付与します。このロールは、クラスターを作成する際、または AWS マネジメントコンソール でサービスを作成または更新すると、自動的に作成されます。詳細については、「Amazon ECS のサービスリンクロールの使用」を参照してください。サービスにリンクされたロールは、次の AWS CLI コマンドを使用して作成することもできます。

```
aws iam create-service-linked-role --aws-service-name ecs.amazonaws.com
```

- お客様のアカウントまたはコンテナインスタンスの IAM ロールは、`awsvpcTrunking` アカウント設定に有効化する必要があります。2 つのコンテナインスタンスロール (`ecsInstanceRole`) を作成することをお勧めします。そして、1 つのロールの `awsvpcTrunking` アカウント設定を有効にして、そのロールを ENI トランキングを必要とするタスクに使用することができます。コンテナインスタンスロールについては、「Amazon ECS コンテナインスタンスの IAM ロール」を参照してください。

前提条件が満たされると、サポートされているいずれかの Amazon EC2 インスタンスタイプを使用して新しいコンテナインスタンスを起動でき、インスタンスの ENI 制限が引き上げられています。サポートされているインスタンスタイプについては、「Amazon ECS コンテナネットワークインターフェイスの増加でサポートされるインスタンス」を参照してください。コンテナインスタンスで、バージョン `1.28.1` 以降のコンテナエージェントと、バージョン `1.28.1-2` 以降の ecs-init パッケージが必要です。Amazon ECS に最適化された AMI の最新の Linux バリアントを使用する場合、これらの要件が満たされます。詳細については、「Amazon ECS Linux コンテナインスタンスの起動」を参照してください。

> **Important**
>
> Amazon EC2 インスタンスでは、リソースベースの IPv4 DNS リクエストがオフになっている必要があります。このオプションを無効にする場合は、Amazon EC2 コンソールを使用した新しいインスタンスの作成時に [Enable resource-based IPV4 (A record) DNS requests] (リソースベースの IPV4 (A レコード) DNS リクエストを有効化) オプションを選択していないことを確認してください。AWS CLI を使ってこのオプションを無効にするには、次のコマンドを使用します。

```
aws ec2 modify-private-dns-name-options --instance-id i-xxxxxxx --no-enable-
resource-name-dns-a-record --no-dry-run
```

AWS CLI を使用して、ENI 制限が引き上げられたコンテナインスタンスを表示するには

各コンテナインスタンスにはデフォルトのネットワークインターフェイスがあり、これはトランクネットワークインターフェイストランクと呼ばれます。トランクネットワークインターフェイスがあることを示す ecs.awsvpc-trunk-id 属性のクエリを実行して、ENI 制限が引き上げられたコンテナインスタンスを一覧表示するには、次のコマンドを使用します。

- list-attributes (AWS CLI)

```
aws ecs list-attributes \
      --target-type container-instance \
      --attribute-name ecs.awsvpc-trunk-id \
      --cluster cluster_name \
      --region us-east-1
```

- Get-ECSAttributeList (AWS Tools for Windows PowerShell)

```
Get-ECSAttributeList -TargetType container-instance -AttributeName ecs.awsvpc-trunk-
id -Region us-east-1
```

Amazon ECS コンテナネットワークインターフェイスの増加でサポートされるインスタンス

以下に示しているのは、サポートされる Amazon EC2 インスタンスタイプと、awsvpcTrunking アカウント設定を有効化する前後で各インスタンスタイプにおいて awsvpc ネットワークモードを使用して起動できるタスクの数です。

Important

同じインスタンスファミリーでは、他のインスタンスタイプがサポートされていますが、a1.metal、c5.metal、c5a.8xlarge、c5ad.8xlarge、c5d.metal、m5.metal、p3dn.24 の各インスタンスタイプはサポートされていません。
c5n、d3、d3en、g3、g3s、g4dn、i3、i3en、inf1、m5dn、m5n、m5zn、mac1、r5b、r5n、r5 および z1d インスタンスファミリーはサポートされていません。

トピック

- 汎用
- コンピューティングの最適化
- メモリ最適化
- ストレージの最適化
- 高速コンピューティング
- ハイパフォーマンスコンピューティング

汎用

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
a1.medium	1	10
a1.large	2	10
a1.xlarge	3	20
a1.2xlarge	3	40
a1.4xlarge	7	60
m5.large	2	10
m5.xlarge	3	20
m5.2xlarge	3	40
m5.4xlarge	7	60
m5.8xlarge	7	60
m5.12xlarge	7	60
m5.16xlarge	14	120
m5.24xlarge	14	120

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
m5a.large	2	10
m5a.xlarge	3	20
m5a.2xlarge	3	40
m5a.4xlarge	7	60
m5a.8xlarge	7	60
m5a.12xlarge	7	60
m5a.16xlarge	14	120
m5a.24xlarge	14	120
m5ad.large	2	10
m5ad.xlarge	3	20
m5ad.2xlarge	3	40
m5ad.4xlarge	7	60
m5ad.8xlarge	7	60
m5ad.12xlarge	7	60
m5ad.16xlarge	14	120
m5ad.24xlarge	14	120
m5d.large	2	10
m5d.xlarge	3	20
m5d.2xlarge	3	40
m5d.4xlarge	7	60

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
m5d.8xlarge	7	60
m5d.12xlarge	7	60
m5d.16xlarge	14	120
m5d.24xlarge	14	120
m5d.metal	14	120
m6a.large	2	10
m6a.xlarge	3	20
m6a.2xlarge	3	40
m6a.4xlarge	7	60
m6a.8xlarge	7	90
m6a.12xlarge	7	120
m6a.16xlarge	14	120
m6a.24xlarge	14	120
m6a.32xlarge	14	120
m6a.48xlarge	14	120
m6a.metal	14	120
m6g.medium	1	4
m6g.large	2	10
m6g.xlarge	3	20
m6g.2xlarge	3	40

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
m6g.4xlarge	7	60
m6g.8xlarge	7	60
m6g.12xlarge	7	60
m6g.16xlarge	14	120
m6g.metal	14	120
m6gd.medium	1	4
m6gd.large	2	10
m6gd.xlarge	3	20
m6gd.2xlarge	3	40
m6gd.4xlarge	7	60
m6gd.8xlarge	7	60
m6gd.12xlarge	7	60
m6gd.16xlarge	14	120
m6gd.metal	14	120
m6i.large	2	10
m6i.xlarge	3	20
m6i.2xlarge	3	40
m6i.4xlarge	7	60
m6i.8xlarge	7	90
m6i.12xlarge	7	120

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
m6i.16xlarge	14	120
m6i.24xlarge	14	120
m6i.32xlarge	14	120
m6i.metal	14	120
m6id.large	2	10
m6id.xlarge	3	20
m6id.2xlarge	3	40
m6id.4xlarge	7	60
m6id.8xlarge	7	90
m6id.12xlarge	7	120
m6id.16xlarge	14	120
m6id.24xlarge	14	120
m6id.32xlarge	14	120
m6id.metal	14	120
m6idn.large	2	10
m6idn.xlarge	3	20
m6idn.2xlarge	3	40
m6idn.4xlarge	7	60
m6idn.8xlarge	7	90
m6idn.12xlarge	7	120

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
m6idn.16xlarge	14	120
m6idn.24xlarge	14	120
m6idn.32xlarge	15	120
m6idn.metal	15	120
m6in.large	2	10
m6in.xlarge	3	20
m6in.2xlarge	3	40
m6in.4xlarge	7	60
m6in.8xlarge	7	90
m6in.12xlarge	7	120
m6in.16xlarge	14	120
m6in.24xlarge	14	120
m6in.32xlarge	15	120
m6in.metal	15	120
m7a.medium	1	4
m7a.large	2	10
m7a.xlarge	3	20
m7a.2xlarge	3	40
m7a.4xlarge	7	60
m7a.8xlarge	7	90

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
m7a.12xlarge	7	120
m7a.16xlarge	14	120
m7a.24xlarge	14	120
m7a.32xlarge	14	120
m7a.48xlarge	14	120
m7a.metal-48xl	14	120
m7g.medium	1	4
m7g.large	2	10
m7g.xlarge	3	20
m7g.2xlarge	3	40
m7g.4xlarge	7	60
m7g.8xlarge	7	60
m7g.12xlarge	7	60
m7g.16xlarge	14	120
m7g.metal	14	120
m7gd.medium	1	4
m7gd.large	2	10
m7gd.xlarge	3	20
m7gd.2xlarge	3	40
m7gd.4xlarge	7	60

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
m7gd.8xlarge	7	60
m7gd.12xlarge	7	60
m7gd.16xlarge	14	120
m7gd.metal	14	120
m7i.large	2	10
m7i.xlarge	3	20
m7i.2xlarge	3	40
m7i.4xlarge	7	60
m7i.8xlarge	7	90
m7i.12xlarge	7	120
m7i.16xlarge	14	120
m7i.24xlarge	14	120
m7i.48xlarge	14	120
m7i.metal-24xl	14	120
m7i.metal-48xl	14	120
m7i-flex.large	2	4
M7i-flex.xlarge	3	10
m7i-flex.2xlarge	3	20
m7i-flex.4xlarge	7	40
m7i-flex.8xlarge	7	60

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
m7i-flex.12xlarge	7	120
m7i-flex.16xlarge	14	120
m8a.medium	1	4
m8a.large	2	10
m8a.xlarge	3	20
m8a.2xlarge	3	40
m8a.4xlarge	7	60
m8a.8xlarge	9	90
m8a.12xlarge	11	120
m8a.16xlarge	15	120
m8a.24xlarge	15	120
m8a.48xlarge	23	120
m8a.metal-24xl	15	120
m8a.metal-48xl	23	120
m8azn.medium	2	4
m8azn.large	3	10
m8azn.xlarge	3	20
m8azn.3xlarge	7	40
m8azn.6xlarge	7	60
m8azn.12xlarge	15	120

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
m8azn.24xlarge	15	120
m8azn.metal-12xl	15	120
m8azn.metal-24xl	15	120
m8g.medium	1	4
m8g.large	2	10
m8g.xlarge	3	20
m8g.2xlarge	3	40
m8g.4xlarge	7	60
m8g.8xlarge	7	60
m8g.12xlarge	7	60
m8g.16xlarge	14	120
m8g.24xlarge	14	120
m8g.48xlarge	14	120
m8g.metal-24xl	14	120
m8g.metal-48xl	14	120
m8gb.medium	1	4
m8gb.large	2	10
m8gb.xlarge	3	20
m8gb.2xlarge	3	40
m8gb.4xlarge	7	60

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
m8gb.8xlarge	9	60
m8gb.12xlarge	11	60
m8gb.16xlarge	15	120
m8gb.24xlarge	23	120
m8gb.48xlarge	23	120
m8gb.metal-24xl	23	120
m8gb.metal-48xl	23	120
m8gd.medium	1	4
m8gd.large	2	10
m8gd.xlarge	3	20
m8gd.2xlarge	3	40
m8gd.4xlarge	7	60
m8gd.8xlarge	7	60
m8gd.12xlarge	7	60
m8gd.16xlarge	14	120
m8gd.24xlarge	14	120
m8gd.48xlarge	14	120
m8gd.metal-24xl	14	120
m8gd.metal-48xl	14	120
m8gn.medium	1	4

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
m8gn.large	2	10
m8gn.xlarge	3	20
m8gn.2xlarge	3	40
m8gn.4xlarge	7	60
m8gn.8xlarge	9	60
m8gn.12xlarge	11	60
m8gn.16xlarge	15	120
m8gn.24xlarge	23	120
m8gn.48xlarge	23	120
m8gn.metal-24xl	23	120
m8gn.metal-48xl	23	120
m8i.large	2	10
m8i.xlarge	3	20
m8i.2xlarge	3	40
m8i.4xlarge	7	60
m8i.8xlarge	9	90
m8i.12xlarge	11	120
m8i.16xlarge	15	120
m8i.24xlarge	15	120
m8i.32xlarge	23	120

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
m8i.48xlarge	23	120
m8i.96xlarge	23	120
m8i.metal-48xl	23	120
m8i.metal-96xl	23	120
m8id.large	2	10
m8id.xlarge	3	20
m8id.2xlarge	3	40
m8id.4xlarge	7	60
m8id.8xlarge	9	90
m8id.12xlarge	11	120
m8id.16xlarge	15	120
m8id.24xlarge	15	120
m8id.32xlarge	23	120
m8id.48xlarge	23	120
m8id.96xlarge	23	120
m8id.metal-48xl	23	120
m8id.metal-96xl	23	120
m8i-flex.large	2	4
m8i-flex.xlarge	3	10
m8i-flex.2xlarge	3	20

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
m8i-flex.4xlarge	7	40
m8i-flex.8xlarge	9	60
m8i-flex.12xlarge	11	120
m8i-flex.16xlarge	15	120
m8in.large	3	10
m8in.xlarge	3	20
m8in.2xlarge	3	40
m8in.4xlarge	7	60
m8in.8xlarge	7	90
m8in.12xlarge	11	120
m8in.16xlarge	15	120
m8in.24xlarge	15	120
m8in.32xlarge	15	120
m8in.48xlarge	23	120
m8in.96xlarge	23	120
m8in.metal-48xl	23	120
m8in.metal-96xl	23	120
m8idn.large	3	10
m8idn.xlarge	3	20
m8idn.2xlarge	3	40

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
m8idn.4xlarge	7	60
m8idn.8xlarge	7	90
m8idn.12xlarge	11	120
m8idn.16xlarge	15	120
m8idn.24xlarge	15	120
m8idn.32xlarge	15	120
m8idn.48xlarge	23	120
m8idn.96xlarge	23	120
m8idn.metal-48xl	23	120
m8idn.metal-96xl	23	120
m8ine.large	3	10
m8ine.xlarge	3	20
m8ine.2xlarge	3	40
m8ine.4xlarge	7	60
m8ine.8xlarge	7	90
m8ine.12xlarge	11	120
m8ib.large	3	10
m8ib.xlarge	3	20
m8ib.2xlarge	3	40
m8ib.4xlarge	7	60

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
m8ib.8xlarge	7	90
m8ib.12xlarge	11	120
m8ib.16xlarge	15	120
m8ib.24xlarge	15	120
m8ib.32xlarge	15	120
m8ib.48xlarge	23	120
m8ib.96xlarge	23	120
m8ib.metal-48xl	23	120
m8ib.metal-96xl	23	120
m8idb.large	3	10
m8idb.xlarge	3	20
m8idb.2xlarge	3	40
m8idb.4xlarge	7	60
m8idb.8xlarge	7	90
m8idb.12xlarge	11	120
m8idb.16xlarge	15	120
m8idb.24xlarge	15	120
m8idb.32xlarge	15	120
m8idb.48xlarge	23	120
m8idb.96xlarge	23	120

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
m8idb.metal-48xl	23	120
m8idb.metal-96xl	23	120
m9g.medium	1	4
m9g.large	2	10
m9g.xlarge	3	20
m9g.2xlarge	3	40
m9g.4xlarge	7	60
m9g.8xlarge	9	60
m9g.12xlarge	11	60
m9g.16xlarge	15	120
m9g.24xlarge	23	120
m9g.48xlarge	23	120
m9g.metal-48xl	23	120
m9gd.medium	1	4
m9gd.large	2	10
m9gd.xlarge	3	20
m9gd.2xlarge	3	40
m9gd.4xlarge	7	60
m9gd.8xlarge	9	60
m9gd.12xlarge	11	60

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
m9gd.16xlarge	15	120
m9gd.24xlarge	23	120
m9gd.48xlarge	23	120
m9gd.metal-48xl	23	120
mac2.metal	7	12
mac2-m1ultra.metal	7	12
mac2-m2.metal	7	12
mac2-m2pro.metal	7	12
mac-m3ultra.metal	7	12
mac-m4.metal	7	12
mac-m4pro.metal	7	12
mac-m4max.metal	7	12

コンピューティングの最適化

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
c5.large	2	10
c5.xlarge	3	20
c5.2xlarge	3	40
c5.4xlarge	7	60

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
c5.9xlarge	7	60
c5.12xlarge	7	60
c5.18xlarge	14	120
c5.24xlarge	14	120
c5a.large	2	10
c5a.xlarge	3	20
c5a.2xlarge	3	40
c5a.4xlarge	7	60
c5a.12xlarge	7	60
c5a.16xlarge	14	120
c5a.24xlarge	14	120
c5ad.large	2	10
c5ad.xlarge	3	20
c5ad.2xlarge	3	40
c5ad.4xlarge	7	60
c5ad.12xlarge	7	60
c5ad.16xlarge	14	120
c5ad.24xlarge	14	120
c5d.large	2	10
c5d.xlarge	3	20

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
c5d.2xlarge	3	40
c5d.4xlarge	7	60
c5d.9xlarge	7	60
c5d.12xlarge	7	60
c5d.18xlarge	14	120
c5d.24xlarge	14	120
c6a.large	2	10
c6a.xlarge	3	20
c6a.2xlarge	3	40
c6a.4xlarge	7	60
c6a.8xlarge	7	90
c6a.12xlarge	7	120
c6a.16xlarge	14	120
c6a.24xlarge	14	120
c6a.32xlarge	14	120
c6a.48xlarge	14	120
c6a.metal	14	120
c6g.medium	1	4
c6g.large	2	10
c6g.xlarge	3	20

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
c6g.2xlarge	3	40
c6g.4xlarge	7	60
c6g.8xlarge	7	60
c6g.12xlarge	7	60
c6g.16xlarge	14	120
c6g.metal	14	120
c6gd.medium	1	4
c6gd.large	2	10
c6gd.xlarge	3	20
c6gd.2xlarge	3	40
c6gd.4xlarge	7	60
c6gd.8xlarge	7	60
c6gd.12xlarge	7	60
c6gd.16xlarge	14	120
c6gd.metal	14	120
c6gn.medium	1	4
c6gn.large	2	10
c6gn.xlarge	3	20
c6gn.2xlarge	3	40
c6gn.4xlarge	7	60

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
c6gn.8xlarge	7	60
c6gn.12xlarge	7	60
c6gn.16xlarge	14	120
c6i.large	2	10
c6i.xlarge	3	20
c6i.2xlarge	3	40
c6i.4xlarge	7	60
c6i.8xlarge	7	90
c6i.12xlarge	7	120
c6i.16xlarge	14	120
c6i.24xlarge	14	120
c6i.32xlarge	14	120
c6i.metal	14	120
c6id.large	2	10
c6id.xlarge	3	20
c6id.2xlarge	3	40
c6id.4xlarge	7	60
c6id.8xlarge	7	90
c6id.12xlarge	7	120
c6id.16xlarge	14	120

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
c6id.24xlarge	14	120
c6id.32xlarge	14	120
c6id.metal	14	120
c6in.large	2	10
c6in.xlarge	3	20
c6in.2xlarge	3	40
c6in.4xlarge	7	60
c6in.8xlarge	7	90
c6in.12xlarge	7	120
c6in.16xlarge	14	120
c6in.24xlarge	14	120
c6in.32xlarge	15	120
c6in.metal	15	120
c7a.medium	1	4
c7a.large	2	10
c7a.xlarge	3	20
c7a.2xlarge	3	40
c7a.4xlarge	7	60
c7a.8xlarge	7	90
c7a.12xlarge	7	120

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
c7a.16xlarge	14	120
c7a.24xlarge	14	120
c7a.32xlarge	14	120
c7a.48xlarge	14	120
c7a.metal-48xl	14	120
c7g.medium	1	4
c7g.large	2	10
c7g.xlarge	3	20
c7g.2xlarge	3	40
c7g.4xlarge	7	60
c7g.8xlarge	7	60
c7g.12xlarge	7	60
c7g.16xlarge	14	120
c7g.metal	14	120
c7gd.medium	1	4
c7gd.large	2	10
c7gd.xlarge	3	20
c7gd.2xlarge	3	40
c7gd.4xlarge	7	60
c7gd.8xlarge	7	60

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
c7gd.12xlarge	7	60
c7gd.16xlarge	14	120
c7gd.metal	14	120
c7gn.medium	1	4
c7gn.large	2	10
c7gn.xlarge	3	20
c7gn.2xlarge	3	40
c7gn.4xlarge	7	60
c7gn.8xlarge	7	60
c7gn.12xlarge	7	60
c7gn.16xlarge	14	120
c7gn.metal	14	120
c7i.large	2	10
c7i.xlarge	3	20
c7i.2xlarge	3	40
c7i.4xlarge	7	60
c7i.8xlarge	7	90
c7i.12xlarge	7	120
c7i.16xlarge	14	120
c7i.24xlarge	14	120

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
c7i.48xlarge	14	120
c7i.metal-24xl	14	120
c7i.metal-48xl	14	120
c7i-flex.large	2	4
c7i-flex.xlarge	3	10
c7i-flex.2xlarge	3	20
c7i-flex.4xlarge	7	40
c7i-flex.8xlarge	7	60
c7i-flex.12xlarge	7	120
c7i-flex.16xlarge	14	120
c8a.medium	1	4
c8a.large	2	10
c8a.xlarge	3	20
c8a.2xlarge	3	40
c8a.4xlarge	7	60
c8a.8xlarge	9	90
c8a.12xlarge	11	120
c8a.16xlarge	15	120
c8a.24xlarge	15	120
c8a.48xlarge	23	120

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
c8a.metal-24xl	15	120
c8a.metal-48xl	23	120
c8g.medium	1	4
c8g.large	2	10
c8g.xlarge	3	20
c8g.2xlarge	3	40
c8g.4xlarge	7	60
c8g.8xlarge	7	60
c8g.12xlarge	7	60
c8g.16xlarge	14	120
c8g.24xlarge	14	120
c8g.48xlarge	14	120
c8g.metal-24xl	14	120
c8g.metal-48xl	14	120
c8gb.medium	1	4
c8gb.large	2	10
c8gb.xlarge	3	20
c8gb.2xlarge	3	40
c8gb.4xlarge	7	60
c8gb.8xlarge	9	60

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
c8gb.12xlarge	11	60
c8gb.16xlarge	15	120
c8gb.24xlarge	23	120
c8gb.48xlarge	23	120
c8gb.metal-24xl	23	120
c8gb.metal-48xl	23	120
c8gd.medium	1	4
c8gd.large	2	10
c8gd.xlarge	3	20
c8gd.2xlarge	3	40
c8gd.4xlarge	7	60
c8gd.8xlarge	7	60
c8gd.12xlarge	7	60
c8gd.16xlarge	14	120
c8gd.24xlarge	14	120
c8gd.48xlarge	14	120
c8gd.metal-24xl	14	120
c8gd.metal-48xl	14	120
c8gn.medium	1	4
c8gn.large	2	10

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
c8gn.xlarge	3	20
c8gn.2xlarge	3	40
c8gn.4xlarge	7	60
c8gn.8xlarge	9	60
c8gn.12xlarge	11	60
c8gn.16xlarge	15	120
c8gn.24xlarge	23	120
c8gn.48xlarge	23	120
c8gn.metal-24xl	23	120
c8gn.metal-48xl	23	120
c8i.large	2	10
c8i.xlarge	3	20
c8i.2xlarge	3	40
c8i.4xlarge	7	60
c8i.8xlarge	9	90
c8i.12xlarge	11	120
c8i.16xlarge	15	120
c8i.24xlarge	15	120
c8i.32xlarge	23	120
c8i.48xlarge	23	120

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
c8i.96xlarge	23	120
c8i.metal-48xl	23	120
c8i.metal-96xl	23	120
c8id.large	2	10
c8id.xlarge	3	20
c8id.2xlarge	3	40
c8id.4xlarge	7	60
c8id.8xlarge	9	90
c8id.12xlarge	11	120
c8id.16xlarge	15	120
c8id.24xlarge	15	120
c8id.32xlarge	23	120
c8id.48xlarge	23	120
c8id.96xlarge	23	120
c8id.metal-48xl	23	120
c8id.metal-96xl	23	120
c8i-flex.large	2	4
c8i-flex.xlarge	3	10
c8i-flex.2xlarge	3	20
c8i-flex.4xlarge	7	40

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
c8i-flex.8xlarge	9	60
c8i-flex.12xlarge	11	120
c8i-flex.16xlarge	15	120
c8in.large	3	10
c8in.xlarge	3	20
c8in.2xlarge	3	40
c8in.4xlarge	7	60
c8in.8xlarge	7	90
c8in.12xlarge	11	120
c8in.16xlarge	15	120
c8in.24xlarge	15	120
c8in.32xlarge	15	120
c8in.48xlarge	23	120
c8in.96xlarge	23	120
c8in.metal-48xl	23	120
c8in.metal-96xl	23	120
c8ine.large	3	10
c8ine.xlarge	3	20
c8ine.2xlarge	3	40
c8ine.4xlarge	7	60

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
c8ine.8xlarge	7	90
c8ine.12xlarge	11	120
c8ib.large	3	10
c8ib.xlarge	3	20
c8ib.2xlarge	3	40
c8ib.4xlarge	7	60
c8ib.8xlarge	7	90
c8ib.12xlarge	11	120
c8ib.16xlarge	15	120
c8ib.24xlarge	15	120
c8ib.32xlarge	15	120
c8ib.48xlarge	23	120
c8ib.96xlarge	23	120
c8ib.metal-48xl	23	120
c8ib.metal-96xl	23	120
c9g.medium	1	4
c9g.large	2	10
c9g.xlarge	3	20
c9g.2xlarge	3	40
c9g.4xlarge	7	60

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
c9g.8xlarge	9	60
c9g.12xlarge	11	60
c9g.16xlarge	15	120
c9g.24xlarge	23	120
c9g.48xlarge	23	120
c9g.metal-48xl	23	120
c9gd.medium	1	4
c9gd.large	2	10
c9gd.xlarge	3	20
c9gd.2xlarge	3	40
c9gd.4xlarge	7	60
c9gd.8xlarge	9	60
c9gd.12xlarge	11	60
c9gd.16xlarge	15	120
c9gd.24xlarge	23	120
c9gd.48xlarge	23	120
c9gd.metal-48xl	23	120

メモリ最適化

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
r5.large	2	10
r5.xlarge	3	20
r5.2xlarge	3	40
r5.4xlarge	7	60
r5.12xlarge	7	60
r5.16xlarge	14	120
r5.24xlarge	14	120
r5a.large	2	10
r5a.xlarge	3	20
r5a.2xlarge	3	40
r5a.4xlarge	7	60
r5a.8xlarge	7	60
r5a.12xlarge	7	60
r5a.16xlarge	14	120
r5a.24xlarge	14	120
r5ad.large	2	10
r5ad.xlarge	3	20
r5ad.2xlarge	3	40
r5ad.4xlarge	7	60

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
r5ad.8xlarge	7	60
r5ad.12xlarge	7	60
r5ad.16xlarge	14	120
r5ad.24xlarge	14	120
r5b.16xlarge	14	120
r5d.large	2	10
r5d.xlarge	3	20
r5d.2xlarge	3	40
r5d.4xlarge	7	60
r5d.8xlarge	7	60
r5d.12xlarge	7	60
r5d.16xlarge	14	120
r5d.24xlarge	14	120
r5dn.16xlarge	14	120
r6a.large	2	10
r6a.xlarge	3	20
r6a.2xlarge	3	40
r6a.4xlarge	7	60
r6a.8xlarge	7	90
r6a.12xlarge	7	120

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
r6a.16xlarge	14	120
r6a.24xlarge	14	120
r6a.32xlarge	14	120
r6a.48xlarge	14	120
r6a.metal	14	120
r6g.medium	1	4
r6g.large	2	10
r6g.xlarge	3	20
r6g.2xlarge	3	40
r6g.4xlarge	7	60
r6g.8xlarge	7	60
r6g.12xlarge	7	60
r6g.16xlarge	14	120
r6g.metal	14	120
r6gd.medium	1	4
r6gd.large	2	10
r6gd.xlarge	3	20
r6gd.2xlarge	3	40
r6gd.4xlarge	7	60
r6gd.8xlarge	7	60

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
r6gd.12xlarge	7	60
r6gd.16xlarge	14	120
r6gd.metal	14	120
r6i.large	2	10
r6i.xlarge	3	20
r6i.2xlarge	3	40
r6i.4xlarge	7	60
r6i.8xlarge	7	90
r6i.12xlarge	7	120
r6i.16xlarge	14	120
r6i.24xlarge	14	120
r6i.32xlarge	14	120
r6i.metal	14	120
r6id.large	2	10
r6id.xlarge	3	20
r6id.2xlarge	3	40
r6id.4xlarge	7	60
r6id.8xlarge	7	90
r6id.12xlarge	7	120
r6id.16xlarge	14	120

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
r6id.24xlarge	14	120
r6id.32xlarge	14	120
r6id.metal	14	120
r6idn.large	2	10
r6idn.xlarge	3	20
r6idn.2xlarge	3	40
r6idn.4xlarge	7	60
r6idn.8xlarge	7	90
r6idn.12xlarge	7	120
r6idn.16xlarge	14	120
r6idn.24xlarge	14	120
r6idn.32xlarge	15	120
r6idn.metal	15	120
r6in.large	2	10
r6in.xlarge	3	20
r6in.2xlarge	3	40
r6in.4xlarge	7	60
r6in.8xlarge	7	90
r6in.12xlarge	7	120
r6in.16xlarge	14	120

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
r6in.24xlarge	14	120
r6in.32xlarge	15	120
r6in.metal	15	120
r7a.medium	1	4
r7a.large	2	10
r7a.xlarge	3	20
r7a.2xlarge	3	40
r7a.4xlarge	7	60
r7a.8xlarge	7	90
r7a.12xlarge	7	120
r7a.16xlarge	14	120
r7a.24xlarge	14	120
r7a.32xlarge	14	120
r7a.48xlarge	14	120
r7a.metal-48xl	14	120
r7g.medium	1	4
r7g.large	2	10
r7g.xlarge	3	20
r7g.2xlarge	3	40
r7g.4xlarge	7	60

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
r7g.8xlarge	7	60
r7g.12xlarge	7	60
r7g.16xlarge	14	120
r7g.metal	14	120
r7gd.medium	1	4
r7gd.large	2	10
r7gd.xlarge *	3	20
r7gd.2xlarge	3	40
r7gd.4xlarge	7	60
r7gd.8xlarge	7	60
r7gd.12xlarge	7	60
r7gd.16xlarge	14	120
r7gd.metal	14	120
r7i.large	2	10
r7i.xlarge	3	20
r7i.2xlarge	3	40
r7i.4xlarge	7	60
r7i.8xlarge	7	90
r7i.12xlarge	7	120
r7i.16xlarge	14	120

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
r7i.24xlarge	14	120
r7i.48xlarge	14	120
r7i.metal-24xl	14	120
r7i.metal-48xl	14	120
r7iz.large	2	10
r7iz.xlarge	3	20
r7iz.2xlarge	3	40
r7iz.4xlarge	7	60
r7iz.8xlarge	7	90
r7iz.12xlarge	7	120
r7iz.16xlarge	14	120
r7iz.32xlarge	14	120
r7iz.metal-16xl	14	120
r7iz.metal-32xl	14	120
r8a.medium	1	4
r8a.large	2	10
r8a.xlarge	3	20
r8a.2xlarge	3	40
r8a.4xlarge	7	60
r8a.8xlarge	9	90

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
r8a.12xlarge	11	120
r8a.16xlarge	15	120
r8a.24xlarge	15	120
r8a.48xlarge	23	120
r8a.metal-24xl	15	120
r8a.metal-48xl	23	120
r8g.medium	1	4
r8g.large	2	10
r8g.xlarge	3	20
r8g.2xlarge	3	40
r8g.4xlarge	7	60
r8g.8xlarge	7	60
r8g.12xlarge	7	60
r8g.16xlarge	14	120
r8g.24xlarge	14	120
r8g.48xlarge	14	120
r8g.metal-24xl	14	120
r8g.metal-48xl	14	120
r8gb.medium	1	4
r8gb.large	2	10

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
r8gb.xlarge	3	20
r8gb.2xlarge	3	40
r8gb.4xlarge	7	60
r8gb.8xlarge	9	60
r8gb.12xlarge	11	60
r8gb.16xlarge	15	120
r8gb.24xlarge	23	120
r8gb.48xlarge	23	120
r8gb.metal-24xl	23	120
r8gb.metal-48xl	23	120
r8gd.medium	1	4
r8gd.large	2	10
r8gd.xlarge	3	20
r8gd.2xlarge	3	40
r8gd.4xlarge	7	60
r8gd.8xlarge	7	60
r8gd.12xlarge	7	60
r8gd.16xlarge	14	120
r8gd.24xlarge	14	120
r8gd.48xlarge	14	120

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
r8gd.metal-24xl	14	120
r8gd.metal-48xl	14	120
r8gn.medium	1	4
r8gn.large	2	10
r8gn.xlarge	3	20
r8gn.2xlarge	3	40
r8gn.4xlarge	7	60
r8gn.8xlarge	9	60
r8gn.12xlarge	11	60
r8gn.16xlarge	15	120
r8gn.24xlarge	23	120
r8gn.48xlarge	23	120
r8gn.metal-24xl	23	120
r8gn.metal-48xl	23	120
r8i.large	2	10
r8i.xlarge	3	20
r8i.2xlarge	3	40
r8i.4xlarge	7	60
r8i.8xlarge	9	90
r8i.12xlarge	11	120

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
r8i.16xlarge	15	120
r8i.24xlarge	15	120
r8i.32xlarge	23	120
r8i.48xlarge	23	120
r8i.96xlarge	23	120
r8i.metal-48xl	23	120
r8i.metal-96xl	23	120
r8id.large	2	10
r8id.xlarge	3	20
r8id.2xlarge	3	40
r8id.4xlarge	7	60
r8id.8xlarge	9	90
r8id.12xlarge	11	120
r8id.16xlarge	15	120
r8id.24xlarge	15	120
r8id.32xlarge	23	120
r8id.48xlarge	23	120
r8id.96xlarge	23	120
r8id.metal-48xl	23	120
r8id.metal-96xl	23	120

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
r8i-flex.large	2	4
r8i-flex.xlarge	3	10
r8i-flex.2xlarge	3	20
r8i-flex.4xlarge	7	40
r8i-flex.8xlarge	9	60
r8i-flex.12xlarge	11	120
r8i-flex.16xlarge	15	120
r8in.large	3	10
r8in.xlarge	3	20
r8in.2xlarge	3	40
r8in.4xlarge	7	60
r8in.8xlarge	7	90
r8in.12xlarge	11	120
r8in.16xlarge	15	120
r8in.24xlarge	15	120
r8in.32xlarge	15	120
r8in.48xlarge	23	120
r8in.96xlarge	23	120
r8in.metal-48xl	23	120
r8in.metal-96xl	23	120

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
r8idn.large	3	10
r8idn.xlarge	3	20
r8idn.2xlarge	3	40
r8idn.4xlarge	7	60
r8idn.8xlarge	7	90
r8idn.12xlarge	11	120
r8idn.16xlarge	15	120
r8idn.24xlarge	15	120
r8idn.32xlarge	15	120
r8idn.48xlarge	23	120
r8idn.96xlarge	23	120
r8idn.metal-48xl	23	120
r8idn.metal-96xl	23	120
r8ib.large	3	10
r8ib.xlarge	3	20
r8ib.2xlarge	3	40
r8ib.4xlarge	7	60
r8ib.8xlarge	7	90
r8ib.12xlarge	11	120
r8ib.16xlarge	15	120

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
r8ib.24xlarge	15	120
r8ib.32xlarge	15	120
r8ib.48xlarge	23	120
r8ib.96xlarge	23	120
r8ib.metal-48xl	23	120
r8ib.metal-96xl	23	120
r8idb.large	3	10
r8idb.xlarge	3	20
r8idb.2xlarge	3	40
r8idb.4xlarge	7	60
r8idb.8xlarge	7	90
r8idb.12xlarge	11	120
r8idb.16xlarge	15	120
r8idb.24xlarge	15	120
r8idb.32xlarge	15	120
r8idb.48xlarge	23	120
r8idb.96xlarge	23	120
r8idb.metal-48xl	23	120
r8idb.metal-96xl	23	120
r9g.medium	1	4

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
r9g.large	2	10
r9g.xlarge	3	20
r9g.2xlarge	3	40
r9g.4xlarge	7	60
r9g.8xlarge	9	60
r9g.12xlarge	11	60
r9g.16xlarge	15	120
r9g.24xlarge	23	120
r9g.48xlarge	23	120
r9g.metal-48xl	23	120
r9gd.medium	1	4
r9gd.large	2	10
r9gd.xlarge	3	20
r9gd.2xlarge	3	40
r9gd.4xlarge	7	60
r9gd.8xlarge	9	60
r9gd.12xlarge	11	60
r9gd.16xlarge	15	120
r9gd.24xlarge	23	120
r9gd.48xlarge	23	120

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
r9gd.metal-48xl	23	120
u-3tb1.56xlarge	7	12
u-6tb1.56xlarge	14	12
u-18tb1.112xlarge	14	12
u-18tb1.metal	14	12
u-24tb1.112xlarge	14	12
u-24tb1.metal	14	12
u7i-6tb.112xlarge	14	120
u7i-8tb.112xlarge	14	120
u7i-12tb.224xlarge	14	120
u7in-16tb.224xlarge	15	120
u7in-24tb.224xlarge	15	120
u7in-32tb.224xlarge	15	120
u7inh-32tb.480xlarge	15	120
x2gd.medium	1	10
x2gd.large	2	10
x2gd.xlarge	3	20
x2gd.2xlarge	3	40
x2gd.4xlarge	7	60
x2gd.8xlarge	7	60

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
x2gd.12xlarge	7	60
x2gd.16xlarge	14	120
x2gd.metal	14	120
x2idn.16xlarge	14	120
x2idn.24xlarge	14	120
x2idn.32xlarge	14	120
x2idn.metal	14	120
x2iedn.xlarge	3	13
x2iedn.2xlarge	3	29
x2iedn.4xlarge	7	60
x2iedn.8xlarge	7	120
x2iedn.16xlarge	14	120
x2iedn.24xlarge	14	120
x2iedn.32xlarge	14	120
x2iedn.metal	14	120
x2iezn.2xlarge	3	64
x2iezn.4xlarge	7	120
x2iezn.6xlarge	7	120
x2iezn.8xlarge	7	120
x2iezn.12xlarge	14	120

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
x2iezn.metal	14	120
x8g.medium	1	4
x8g.large	2	10
x8g.xlarge	3	20
x8g.2xlarge	3	40
x8g.4xlarge	7	60
x8g.8xlarge	7	60
x8g.12xlarge	7	60
x8g.16xlarge	14	120
x8g.24xlarge	14	120
x8g.48xlarge	14	120
x8g.metal-24xl	14	120
x8g.metal-48xl	14	120
x8aedz.large	3	10
x8aedz.xlarge	3	20
x8aedz.3xlarge	7	40
x8aedz.6xlarge	7	60
x8aedz.12xlarge	15	120
x8aedz.24xlarge	15	120
x8aedz.metal-12xl	15	120

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
x8aedz.metal-24xl	15	120
x8i.large	2	10
x8i.xlarge	3	20
x8i.2xlarge	3	40
x8i.4xlarge	7	60
x8i.8xlarge	9	90
x8i.12xlarge	11	120
x8i.16xlarge	15	120
x8i.24xlarge	15	120
x8i.32xlarge	23	120
x8i.48xlarge	23	120
x8i.64xlarge	23	120
x8i.96xlarge	23	120
x8i.metal-48xl	23	120
x8i.metal-96xl	23	120

ストレージの最適化

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
i4g.large	2	10

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
i4g.xlarge	3	20
i4g.2xlarge	3	40
i4g.4xlarge	7	60
i4g.8xlarge	7	60
i4g.16xlarge	14	120
i4i.xlarge	3	8
i4i.2xlarge	3	28
i4i.4xlarge	7	58
i4i.8xlarge	7	118
i4i.12xlarge	7	118
i4i.16xlarge	14	248
i4i.24xlarge	14	118
i4i.32xlarge	14	498
i4i.metal	14	498
i7i.large	2	10
i7i.xlarge	3	20
i7i.2xlarge	3	40
i7i.4xlarge	7	60
i7i.8xlarge	7	90
i7i.12xlarge	7	90

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
i7i.16xlarge	14	120
i7i.24xlarge	14	120
i7i.48xlarge	14	120
i7i.metal-24xl	14	120
i7i.metal-48xl	14	120
i7ie.large	2	20
i7ie.xlarge	3	29
i7ie.2xlarge	3	29
i7ie.3xlarge	3	29
i7ie.6xlarge	7	60
i7ie.12xlarge	7	60
i7ie.18xlarge	14	120
i7ie.24xlarge	14	120
i7ie.48xlarge	14	120
i7ie.metal-24xl	14	120
i7ie.metal-48xl	14	120
i8g.large	2	10
i8g.xlarge	3	20
i8g.2xlarge	3	40
i8g.4xlarge	7	60

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
i8g.8xlarge	7	60
i8g.12xlarge	7	60
i8g.16xlarge	14	120
i8g.24xlarge	14	120
i8g.48xlarge	14	120
i8g.metal-24xl	14	120
i8g.metal-48xl	14	120
i8ge.large	2	20
i8ge.xlarge	3	29
i8ge.2xlarge	3	29
i8ge.3xlarge	5	29
i8ge.6xlarge	9	60
i8ge.12xlarge	11	60
i8ge.18xlarge	15	120
i8ge.24xlarge	15	120
i8ge.48xlarge	23	120
i8ge.metal-24xl	15	120
i8ge.metal-48xl	23	120
im4gn.large	2	10
im4gn.xlarge	3	20

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
im4gn.2xlarge	3	40
im4gn.4xlarge	7	60
im4gn.8xlarge	7	60
im4gn.16xlarge	14	120
is4gen.medium	1	4
is4gen.large	2	10
is4gen.xlarge	3	20
is4gen.2xlarge	3	40
is4gen.4xlarge	7	60
is4gen.8xlarge	7	60

高速コンピューティング

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
dl1.24xlarge	59	120
dl2q.24xlarge	14	120
f2.6xlarge	7	90
f2.12xlarge	7	120
f2.48xlarge	14	120
g4ad.xlarge	1	12

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
g4ad.2xlarge	1	12
g4ad.4xlarge	2	12
g4ad.8xlarge	3	12
g4ad.16xlarge	7	12
g5.xlarge	3	6
g5.2xlarge	3	19
g5.4xlarge	7	40
g5.8xlarge	7	90
g5.12xlarge	14	120
g5.16xlarge	7	120
g5.24xlarge	14	120
g5.48xlarge	6	120
g5g.xlarge	3	20
g5g.2xlarge	3	40
g5g.4xlarge	7	60
g5g.8xlarge	7	60
g5g.16xlarge	14	120
g5g.metal	14	120
g6.xlarge	3	20
g6.2xlarge	3	40

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
g6.4xlarge	7	60
g6.8xlarge	7	90
g6.12xlarge	7	120
g6.16xlarge	14	120
g6.24xlarge	14	120
g6.48xlarge	14	120
g6e.xlarge	3	20
g6e.2xlarge	3	40
g6e.4xlarge	7	60
g6e.8xlarge	7	90
g6e.12xlarge	9	120
g6e.16xlarge	14	120
g6e.24xlarge	19	120
g6e.48xlarge	39	120
g6f.large	1	10
g6f.xlarge	3	20
g6f.2xlarge	3	40
g6f.4xlarge	7	60
gr6.4xlarge	7	60
gr6.8xlarge	7	90

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
gr6f.4xlarge	7	60
g7.2xlarge	3	40
g7.4xlarge	7	60
g7.8xlarge	9	90
g7.12xlarge	11	120
g7.24xlarge	23	120
g7.48xlarge	23	120
g7e.2xlarge	3	242
g7e.4xlarge	7	242
g7e.8xlarge	7	242
g7e.12xlarge	9	242
g7e.24xlarge	19	242
g7e.48xlarge	39	242
inf2.xlarge	3	20
inf2.8xlarge	7	90
inf2.24xlarge	14	120
inf2.48xlarge	14	120
p4d.24xlarge	59	120
p4de.24xlarge	59	120
p5.4xlarge	3	60

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
p5.48xlarge	63	242
p5e.48xlarge	63	242
p5en.48xlarge	63	242
p6-b200.48xlarge	31	242
p6-b300.48xlarge	67	242
p6e-gb200.36xlarge	38	120
trn1.2xlarge	3	19
trn1.32xlarge	39	120
trn1n.32xlarge	79	242
trn2.3xlarge	1	14
trn2.48xlarge	31	242
trn2u.48xlarge	31	242
vt1.3xlarge	3	40
vt1.6xlarge	7	60
vt1.24xlarge	14	120

ハイパフォーマンスコンピューティング

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
hpc6a.48xlarge	1	120

インスタンスタイプ	ENI トランキングなしのタスク制限	ENI トランキングありのタスク制限
hpc6id.32xlarge	1	120
hpc7g.4xlarge	3	120
hpc7g.8xlarge	3	120
hpc7g.16xlarge	3	120
hpc8a.96xlarge	3	-2

Amazon ECS Linux コンテナインスタンスのメモリを予約する

Amazon ECS コンテナエージェントがクラスターにコンテナインスタンスを登録する場合、エージェントは、コンテナインスタンスがタスク用に予約できるメモリ容量を決定する必要があります。プラットフォームのメモリオーバーヘッドとシステムカーネルが占めるメモリのため、この数値は、Amazon EC2 インスタンスとして公開されているインストール済みメモリ量とは異なります。例えば、`m4.large` インスタンスには 8 GiB のメモリがインストールされています。しかし、これはコンテナインスタンスが登録されたときに、タスクに使用できるメモリが正確に 8192 MiB に変換されるとは限りません。

ECS マネージドインスタンスのメモリリソースの決定

Amazon ECS マネージドインスタンスは、階層的なアプローチを使用してタスクのメモリリソース要件を決定します。Docker のメモリイントロスペクションに依存する EC2 上の ECS とは異なり、ECS マネージドインスタンスは、スケジューリングの決定時にタスクペイロードから直接メモリ要件を計算します。

ECS マネージドインスタンスのエージェントはタスクを受信すると、次の優先順位を使用してメモリ要件を計算します。

1. タスクレベルのメモリ (最高優先度) – タスクレベルのメモリがタスク定義で指定されている場合、エージェントはこの値を直接使用します。この場合、すべてのコンテナレベルのメモリ設定より優先されます。
2. コンテナレベルのメモリ合計 (フォールバック) – タスクレベルのメモリが指定されていない場合 (または 0 の場合)、エージェントはタスク内のすべてのコンテナのメモリ要件を合計します。コンテナごとに以下を使用します。

a. メモリ予約 (ソフト制限) – コンテナが設定で memoryReservation を指定した場合、エージェントはこの値を使用します。
b. コンテナメモリ (ハード制限) – memoryReservation が指定されていない場合、エージェントはコンテナの memoryフィールドを使用します。

Example指定されたタスクレベルのメモリ

タスクレベルのメモリを指定すると、コンテナレベルの設定より優先されます。

```
{
  "family": "my-task",
  "memory": "2048",
  "containerDefinitions": [
    {
      "name": "container1",
      "memory": 1024,
      "memoryReservation": 512
    }
  ]
}
```

エージェントは 2048 MiB を予約します (タスクレベルのメモリが優先)。

Example予約を含むコンテナレベルのメモリ

タスクレベルのメモリが指定されていない場合、エージェントはコンテナのメモリ要件を合計します。

```
{
  "family": "my-task",
  "containerDefinitions": [
    {
      "name": "container1",
      "memory": 1024,
      "memoryReservation": 512
    },
    {
      "name": "container2",
      "memory": 512
    }
  ]
```

```
}
```

エージェントは 512 MiB (container1 予約) + 512 MiB (container2 メモリ) = 合計 1024 MiB を予約します。

ECS マネージドインスタンスのエージェントは、次の 3 つのフェーズでメモリ計算を実行します。

1. タスク受信 – ECS コントロールプレーンからタスクペイロードが到着すると、エージェントは直ちに必要なメモリを計算します。
2. リソースストレージ – 計算されたメモリ要件は、後でリソースの課金オペレーションで使用するため、タスクモデルに保存されます。
3. スケジュールの決定 – タスクを受け入れる前に、エージェントは十分なメモリが使用可能かどうかを確認します。メモリが不足している場合タスクは拒否され、リソースが利用可能になるまで ECS サービスキューに残留します。

> **Note**
>
> EC2 の ECS とは異なり、ECS マネージドインスタンスは ECS_RESERVED_MEMORY の設定変数を使用しません。システムプロセスのメモリ予約は、基盤となるプラットフォームのリソース管理を通じて処理され、エージェントはタスク定義に基づいて正確なリソースの課金を実行します。

EC2 上の ECS の場合、Amazon ECS コンテナエージェントには、タスクに割り当てられたプールから指定したメモリ容量 (MiB) を削除するのに使用できる、ECS_RESERVED_MEMORY という設定変数があります。これにより、重要なシステムプロセスのメモリを効果的に確保することができます。

タスクでコンテナインスタンスのすべてのメモリを占有している場合、メモリが不可欠なシステムプロセスとタスクが競合し、システム障害が発生する可能性があります。

例えば、コンテナエージェント設定ファイルで ECS_RESERVED_MEMORY=256 を指定すると、そのインスタンスの総メモリマイナス 256 MiB が登録され、256 MiBのメモリは ECS タスクに割り当てされなくなります。エージェント構成変数とその設定方法の詳細については、Amazon ECS コンテナエージェントの設定 および Amazon ECS Linux コンテナインスタンスをブートストラップしてデータを渡す を参照してください 。

タスクに 8192 MiB を指定して、使用可能なメモリが 8192 MiB 以上のコンテナインスタンスがなくこの要件を満たせない場合、そのタスクはクラスターに配置できません。マネージド型コンピュー

ティング環境を使用している場合、リクエストに対応するために AWS Batch はより大きなインスタンスタイプを起動する必要があります。

Amazon ECS コンテナエージェントは、Docker `ReadMemInfo()`関数を使用してオペレーティングシステムで使用可能な合計メモリのクエリを実行します。Linux と Windows の両方に、合計メモリを判断できるコマンドラインユーティリティが備わっています。

Example- Linux 合計メモリを決定

free コマンドは、オペレーティングシステムによって認識される合計メモリを返します。

```
$ free -b
```

Amazon ECS に最適化された Amazon Linux AMI を実行する `m4.large` インスタンスの出力例。

```
             total       used       free     shared    buffers     cached
Mem:    8373026816  348180480 8024846336      90112   25534464  205418496
-/+ buffers/cache:  117227520 8255799296
```

このインスタンスには 8373026816 バイトの合計メモリーがあり、タスクに使用できる 7985 MiB に変換されます。

Example-Windows 合計メモリを決定

wmic コマンドは、オペレーティングシステムによって認識される合計メモリを返します。

```
C:\> wmic ComputerSystem get TotalPhysicalMemory
```

Amazon ECS に最適化された Windows Server AMI を実行する `m4.large` インスタンスの出力例。

```
TotalPhysicalMemory
8589524992
```

このインスタンスには合計メモリーが 8589524992 バイトあり、タスクに使用可能な 8191 MiB に変換されます。

コンテナインスタンスのメモリを表示する

Amazon ECS コンソール (または DescribeContainerInstances API 操作) で、コンテナインスタンスに登録されているメモリ容量を表示できます。特定のインスタンスタイプに対して、可能な限り多く

のメモリをタスクに割り当て、リソース使用率を最大化しようとしている場合は、そのコンテナインスタンスに使用可能なメモリを確認してから、そのタスクに十分なメモリを割り当てることができます。

コンテナインスタンスメモリを表示するには

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
2. ナビゲーションペインで [クラスター] を選択し、コンテナインスタンスをホストするクラスターを選択します。
3. [インフラストラクチャ] を選択し、[コンテナインスタンス] でコンテナインスタンスを選択します。
4. [リソース] セクションに、コンテナインスタンス用に登録された使用可能なメモリが表示されます。

   [登録済み] メモリの値は、Amazon ECS の初回起動時に登録されたときのコンテナインスタンスのメモリの値で、[利用可能] メモリの値は、まだ タスクに割り当てられていないメモリの値です。

AWS Systems Manager を使用して Amazon ECS コンテナインスタンスをリモートで管理する

AWS Systems Manager (Systems Manager) で Run Command 機能を使用すると、Amazon ECS コンテナインスタンスの設定を安全にリモートで管理できます。Run Command を使用すると、インスタンスにローカルにログオンしなくても一般的な管理タスクを簡単に実行することができます。複数のコンテナインスタンスでコマンドを同時に実行することで、クラスター全体の設定の変更を管理できます。Run Command は各コマンドのステータスと結果をレポートします。

ここでは、Run Command を使用して実行できるタスクのタイプについていくつか例を示します。

- パッケージをインストールまたはアンインストールする。
- セキュリティ更新プログラムを実行する。
- Docker イメージをクリーンアップする。
- サービスを停止または起動する。
- システムリソースを表示する。
- ログファイルを表示する。
- ファイルオペレーションを実行する。

Run Command の詳細については、の「AWS Systems Managerユーザーガイド」の「AWS Systems ManagerRun Command」を参照してください。

Amazon ECS で Systems Manager を使用するには、以下の前提条件があります。

1. コンテナインスタンスロール (ecsInstanceRole) に、Systems Manager API にアクセスするためのアクセス許可を付与する必要があります。これを行うには、AmazonSSMManagedInstanceCore を `ecsInstanceRole` ロールに割り当てます。ポリシーをロールにアタッチする方法については、「AWS Identity and Access Management ユーザーガイド」の「ロールに対するアクセス許可を更新する」を参照してください。
2. SSM Agent がコンテナインスタンスにインストールされていることを確認します。詳細については、「Linux 用 EC2 インスタンスに SSM Agent を手動でインストールおよびアンインストールする」を参照してください。

Systems Manager 管理ポリシーを `ecsInstanceRole` にアタッチして、AWS Systems Manager エージェント (SSM Agent) がコンテナインスタンスにインストールされていることを確認したら、Run Command を使用してコマンドをコンテナインスタンスに送信することができます。インスタンスでのコマンドおよびシェルスクリプトの実行と、その結果の出力の表示については、AWS Systems Managerユーザーガイド の「Systems Manager Run Command を使用したコマンドの実行」および「Run Command のチュートリアル」を参照してください。

一般的なユースケースは、Run Command でコンテナインスタンスソフトウェアを更新することです。「AWS Systems Manager ユーザーガイド」の手順を実行し、以下のパラメータを指定します。

パラメータ	値
コマンドのドキュメント	`AWS-RunShellScript`
--コマンド:	`$ yum update -y`
ターゲットインスタンス	コンテナインスタンス

Amazon ECS Linux コンテナインスタンスに HTTP プロキシを使用する

Amazon ECS コンテナエージェントと Docker デーモンの両方に HTTP プロキシを使用するように Amazon ECS コンテナインスタンスを設定できます。これは、コンテナインスタンスに、Amazon

VPC インターネットゲートウェイ、NAT ゲートウェイ、またはインスタンスを介した外部ネットワークアクセスがない場合に便利です。

HTTP プロキシを使用するように Amazon ECS Linux コンテナインスタンスを設定するには、起動時に該当するファイルで以下の変数に Amazon EC2 ユーザーデータを設定します。手動で設定ファイルを編集してから、エージェントを再起動することもできます。

`/etc/ecs/ecs.config` (Amazon Linux 2 および Amazon Linux AMI)

`HTTP_PROXY=10.0.0.131:3128`

この値を、Amazon ECS エージェントがインターネットへの接続に使用する HTTP プロキシのホスト名 (または IP アドレス) とポート番号に設定します。例えば、コンテナインスタンスに、Amazon VPC インターネットゲートウェイ、NAT ゲートウェイ、またはインスタンスを介した外部ネットワークアクセスがない場合です。

`NO_PROXY=169.254.169.254,169.254.170.2,/var/run/docker.sock`

この値を `169.254.169.254,169.254.170.2,/var/run/docker.sock` に設定して、EC2 インスタンスのメタデータ、タスク用の IAM; ロール、および Docker デーモンのトラフィックをプロキシからフィルタリングします。

`/etc/systemd/system/ecs.service.d/http-proxy.conf` (Amazon Linux 2 のみ)

`Environment="HTTP_PROXY=10.0.0.131:3128/"`

この値を、`ecs-init` がインターネットへの接続に使用する HTTP プロキシのホスト名 (または IP アドレス) とポート番号に設定します。例えば、コンテナインスタンスに、Amazon VPC インターネットゲートウェイ、NAT ゲートウェイ、またはインスタンスを介した外部ネットワークアクセスがない場合です。

`Environment="NO_PROXY=169.254.169.254,169.254.170.2,/var/run/docker.sock"`

この値を `169.254.169.254,169.254.170.2,/var/run/docker.sock` に設定して、EC2 インスタンスのメタデータ、タスク用の IAM; ロール、および Docker デーモンのトラフィックをプロキシからフィルタリングします。

`/etc/init/ecs.override` (Amazon Linux AMI のみ)

`env HTTP_PROXY=10.0.0.131:3128`

この値を、`ecs-init` がインターネットへの接続に使用する HTTP プロキシのホスト名 (または IP アドレス) とポート番号に設定します。例えば、コンテナインスタンスに、Amazon

VPC インターネットゲートウェイ、NAT ゲートウェイ、またはインスタンスを介した外部ネットワークアクセスがない場合です。

```
env NO_PROXY=169.254.169.254,169.254.170.2,/var/run/docker.sock
```

この値を 169.254.169.254,169.254.170.2,/var/run/docker.sock に設定して、EC2 インスタンスのメタデータ、タスク用の IAM; ロール、および Docker デーモンのトラフィックをプロキシからフィルタリングします。

/etc/systemd/system/docker.service.d/http-proxy.conf (Amazon Linux 2 のみ)

```
Environment="HTTP_PROXY=http://10.0.0.131:3128"
```

この値を、Docker デーモンがインターネットへの接続に使用する HTTP プロキシのホスト名 (または IP アドレス) とポート番号に設定します。例えば、コンテナインスタンスに、Amazon VPC インターネットゲートウェイ、NAT ゲートウェイ、またはインスタンスを介した外部ネットワークアクセスがない場合です。

```
Environment="NO_PROXY=169.254.169.254,169.254.170.2"
```

この値を 169.254.169.254,169.254.170.2 に設定して、EC2 インスタンスのメタデータをプロキシからフィルタリングします。

/etc/sysconfig/docker (Amazon Linux AMI および Amazon Linux 2 のみ)

```
export HTTP_PROXY=http://10.0.0.131:3128
```

この値を、Docker デーモンがインターネットへの接続に使用する HTTP プロキシのホスト名 (または IP アドレス) とポート番号に設定します。例えば、コンテナインスタンスに、Amazon VPC インターネットゲートウェイ、NAT ゲートウェイ、またはインスタンスを介した外部ネットワークアクセスがない場合です。

```
export NO_PROXY=169.254.169.254,169.254.170.2
```

この値を 169.254.169.254,169.254.170.2 に設定して、EC2 インスタンスのメタデータをプロキシからフィルタリングします。

これらの環境変数を前述のファイルで設定すると、Amazon ECS コンテナエージェント、ecs-init、および Docker デーモンのみに影響があります。プロキシを使用する他のサービス (yum など) を設定することはありません。

プロキシの設定方法については、「Amazon Linux 2 または AL2023 で Docker および Amazon ECS コンテナエージェントに HTTP プロキシを設定する方法」を参照してください。

Amazon ECS Auto Scaling グループ用に事前初期化されたインスタンスを設定する

Amazon ECS では、Amazon EC2 Auto Scaling ウォームプールをサポートします。ウォームプールは、事前に初期化済みの Amazon EC2 インスタンスグループでサービス開始が準備されています。アプリケーションがスケールアウトする必要がある場合は、常に Amazon EC2 Auto Scaling はコールドインスタンスを起動するのではなく、ウォームプールから事前に初期化されたインスタンスを使用し、最終的な初期化プロセスの実行を許可し、インスタンスを使用開始します。

ウォームプールの詳細、および Auto Scaling グループにウォームプールを追加する方法については、「Amazon EC2 Auto Scaling ユーザーガイド」の「Amazon EC2 Auto Scaling のウォームプール」を参照してください。

Amazon ECS の Auto Scaling グループのウォームプールを作成または更新する場合、スケールイン (`ReuseOnScaleIn`) 時にインスタンスをウォームプールに戻すオプションを設定することはできません。詳細については、「AWS Command Line Interface リファレンス」の「put-warm-pool」を参照してください。

Amazon ECS クラスターでウォームプールを使用するには、Amazon EC2 Auto Scaling グループ起動テンプレートの [User data] (ユーザーデータ) フィールドで `ECS_WARM_POOLS_CHECK` エージェント設定変数を `true` に設定します。

以下は、Linux インスタンス用の Amazon EC2 起動テンプレートの [User data] (ユーザーデータ) フィールドでエージェント設定変数の指定方法の例を示しています。*`MyCluster`* をクラスターの名前に置き換えます。Windows インスタンスの場合は、「Amazon ECS Windows インスタンスでのウォームプールの使用」を参照してください。

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_WARM_POOLS_CHECK=true
EOF
```

この `ECS_WARM_POOLS_CHECK` 変数は、エージェントバージョン `1.59.0` 以降でのみサポートされています。変数の詳細については、「Amazon ECS コンテナエージェントの設定」を参照してください。

Amazon ECS Windows インスタンスでのウォームプールの使用

Windows インスタンスで 停止 プール状態でウォームプールを使用する場合は、Amazon EC2 Auto Scaling ライフサイクルフックを使用して、インスタンスが停止する前にユーザーデータの実行が完

了していることを確認します。ライフサイクルフックがないと、Amazon ECS の初期化が完了する前にインスタンスが停止することがあります。

Windows ウォームプールインスタンスのライフサイクルフックを設定するには:

1. 次の IAM ポリシーをコンテナインスタンスロールに追加して、インスタンスがライフサイクルアクションを完了できるようにします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "autoscaling:CompleteLifecycleAction"
            ],
            "Resource": "*"
        }
    ]
}
```

2. デフォルトの結果として ABANDON で 起動するインスタンス の Auto Scaling グループにライフサイクルフックを追加し、Amazon ECS の初期化に十分な時間 (10 分など) を許容するハートビートタイムアウトを追加します。詳細については、「Amazon EC2 Auto Scaling ユーザーガイド」の「ライフサイクルフックの追加」を参照してください。
3. 以下のユーザーデータを使用して、スケジュールされたタスクを作成し、一定時間後にライフサイクルアクションを完了させます。この遅延により、EC2Launch はインスタンスが停止される前に状態の保存を完了することが保証されます。*MyCluster*、*my-lifecycle-hook*、*my-asg-name* を独自の値に置き換えます。

```
<powershell>
$lifecycleHookName = "my-lifecycle-hook"
$autoScalingGroupName = "my-asg-name"

$scriptContent = @'
$token = Invoke-RestMethod -Method Put -Uri http://169.254.169.254/latest/api/token -
Headers @{"X-aws-ec2-metadata-token-ttl-seconds" = "21600"}
$instanceId = Invoke-RestMethod -Method Get -Uri http://169.254.169.254/latest/meta-
data/instance-id -Headers @{"X-aws-ec2-metadata-token" = $token}
$region = Invoke-RestMethod -Method Get -Uri http://169.254.169.254/latest/meta-data/
placement/region -Headers @{"X-aws-ec2-metadata-token" = $token}
```

```
$lifecycleHookName = "my-lifecycle-hook"
$autoScalingGroupName = "my-asg-name"
Complete-ASLifecycleAction -LifecycleHookName $lifecycleHookName -
AutoScalingGroupName $autoScalingGroupName -InstanceId $instanceId -
LifecycleActionResult 'CONTINUE' -Region $region
'@
Set-Content -Path "C:\ProgramData\Amazon\ECS\complete-lifecycle.ps1" -Value
 $scriptContent

try
{
    [Environment]::SetEnvironmentVariable("ECS_WARM_POOLS_CHECK", "true", "Machine")
    Import-Module ECSTools
    Initialize-ECSAgent -Cluster MyCluster -EnableTaskIAMRole

    $taskAction = New-ScheduledTaskAction -Execute "powershell.exe" -Argument
 "-NoProfile -ExecutionPolicy Bypass -File C:\ProgramData\Amazon\ECS\complete-
lifecycle.ps1"
    $taskTriggerOnce = New-ScheduledTaskTrigger -Once -At ((Get-Date).AddSeconds(60))
    $taskTriggerStartup = New-ScheduledTaskTrigger -AtStartup
    $taskTriggerStartup.Delay = "PT60S"
    $settings = New-ScheduledTaskSettingsSet -StartWhenAvailable
    $taskPrincipal = New-ScheduledTaskPrincipal -UserId "SYSTEM" -LogonType
 ServiceAccount -RunLevel Highest
    Register-ScheduledTask -TaskName "CompleteLifecycleHook" -Action $taskAction -
Trigger $taskTriggerOnce, $taskTriggerStartup -Principal $taskPrincipal -Settings
 $settings -Force
}
catch
{
    $token = Invoke-RestMethod -Method Put -Uri http://169.254.169.254/latest/api/
token -Headers @{"X-aws-ec2-metadata-token-ttl-seconds" = "21600"}
    $INSTANCE = Invoke-RestMethod -Method Get -Uri http://169.254.169.254/latest/
meta-data/instance-id -Headers @{"X-aws-ec2-metadata-token" = $token}
    $region = Invoke-RestMethod -Method Get -Uri http://169.254.169.254/latest/meta-
data/placement/region -Headers @{"X-aws-ec2-metadata-token" = $token}
    Complete-ASLifecycleAction -InstanceId $INSTANCE -LifecycleHookName
 $lifecycleHookName -AutoScalingGroupName $autoScalingGroupName -
LifecycleActionResult ABANDON -Region $region
}
</powershell>
```

このユーザーデータスクリプトは、以下を実行します。

- ウォームプールチェックを有効にして Amazon ECS エージェントを初期化します。
- ユーザーデータの完了から 60 秒待機するスケジュールされたタスクを作成し、ライフサイクルフックにシグナルを送信して続行します。
- その後の再起動ごとに実行されるスケジュールされたタスクを作成し、60 秒後に続行するようにライフサイクルフックに通知します。
- 初期化が失敗すると、ライフサイクルアクションは ABANDON で完了するため、インスタンスは終了して置き換えられます。

Amazon ECS コンテナエージェントをアップデートする

場合によっては、バグの修正や新機能を取得するために Amazon ECS コンテナエージェントを更新する必要があります。Amazon ECS コンテナエージェントの更新によって、コンテナインスタンスで実行中のタスクやサービスが中断されることはありません。エージェントを更新するプロセスは、コンテナインスタンスが Amazon ECS 対応 AMI で起動されたか、別のオペレーティングシステムで起動されたかによって異なります。

> **Note**
>
> エージェント更新は Windows コンテナインスタンスに適用されません。Windows クラスター内のエージェントバージョンを更新するには、新しいコンテナインスタンスを起動することをお勧めします。

Amazon ECSコンテナエージェントバージョンの確認

コンテナインスタンスで実行中のコンテナエージェントのバージョンをチェックして、更新が必要かどうかを確認できます。Amazon ECSコンソールのコンテナインスタンスビューにエージェントバージョンが表示されます。以下の手順を使用してエージェントバージョンを確認します。

Amazon ECS console

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションバーから、外部インスタンスが存在するリージョンを選択します。
3. ナビゲーションペインで [Clusters] (クラスター) を選択し、外部インスタンスをホストするクラスターを選択します。
4. [Cluster : ***name***] (クラスター: 名前) のページで、[Infrastructure] (インフラストラクチャ) タブを選択します。

5. [Container instances] (コンテナインスタンス) で、コンテナインスタンスの [Agent version] (エージェントバージョン) 列に注意してください。コンテナインスタンスに最新バージョンのコンテナエージェントが含まれていない場合、コンソールにアラートメッセージが表示され、古いエージェントバージョンにフラグが設定されます。

   エージェントバージョンが古い場合、次の手順でコンテナエージェントを更新できます。

   - コンテナインスタンスで Amazon ECS 対応AMIを実行している場合は、「Amazon ECS 対応 AMI での Amazon ECS コンテナエージェントのアップデート」を参照してください。
   - コンテナインスタンスで Amazon ECS 対応AMI を実行してない場合は、「Amazon ECS コンテナエージェントの手動更新 (Amazon ECS 最適化以外の AMI の場合)」を参照してください。

   > **Important**
   >
   > Amazon ECS 対応AMIで Amazon ECSエージェントバージョンを v1.0.0 より古いバージョンから更新するには、現行のコンテナインスタンスを終了し、最新バージョンの AMI で新しいインスタンスを起動することをお勧めします。プレビューバージョンを使用しているコンテナインスタンスは削除し、最新バージョンの AMI に置き換える必要があります。詳細については、「Amazon ECS Linux コンテナインスタンスの起動」を参照してください。

Amazon ECS container agent introspection API

また、コンテナインスタンス自体からAmazon ECS コンテナエージェントの詳細分析 API のバージョンを確認するために使用できます。詳細については、「Amazon ECS コンテナの詳細分析」を参照してください。

詳細分析 API を使用して、Amazon ECS コンテナエージェントで最新バージョンが実行されているかどうかを確認するには

1. SSH 経由でコンテナインスタンスにログインします。
2. 詳細分析 API をクエリします。

```
[ec2-user ~]$ curl -s 127.0.0.1:51678/v1/metadata | python3 -mjson.tool
```

> **Note**
>
> 詳細分析 API は`Version` 情報を Amazon ECS コンテナエージェントのバージョン v1.0.0 に追加しました。詳細分析 API をクエリして `Version` が存在しない場合、または詳細分析 API 自体がエージェントに存在しない場合、実行しているバージョンが v0.0.3 以前であり、更新する必要があります。

Amazon ECS 対応 AMI での Amazon ECS コンテナエージェントのアップデート

Amazon ECS対応AMI を使用している場合は、いくつかの方法で最新バージョンの Amazon ECS コンテナエージェントを取得できます (推奨される順に示します)。

- 現在のコンテナインスタンスを終了して Amazon ECS 対応 Amazon Linux 2 AMI最新バージョンを起動します (手動で起動するか、最新の AMI で自動スケーリング起動設定を更新して起動します)。これにより、最新のテスト済みおよび検証済みバージョンの Amazon Linux、Docker、`ecs-init`、および Amazon ECS コンテナエージェントを備えたの新しいコンテナインスタンスが提供されます。詳細については、「Amazon ECS に最適化された Linux AMI」を参照してください。
- インスタンスに SSH で接続し、`ecs-init` パッケージ (および依存関係) を最新バージョンに更新します。このオペレーションにより、Amazon Linux リポジトリで最新のテスト済みおよび検証済みバージョンの Docker と `ecs-init`、およびAmazon ECS コンテナエージェントの最新バージョンが提供されます。詳細については、「Amazon ECS対応 &AMI; の `ecs-init` パッケージを更新するには」を参照してください。
- コンソールまたは `UpdateContainerAgent` や AWS CLI SDK 経由で AWS API オペレーションを使用し、コンテナエージェントを更新します。詳細については、「`UpdateContainerAgent` API オペレーションで Amazon ECS コンテナエージェントを更新する」を参照してください。

> **Note**
>
> エージェント更新は Windows コンテナインスタンスに適用されません。Windows クラスター内のエージェントバージョンを更新するには、新しいコンテナインスタンスを起動することをお勧めします。

Amazon ECS対応 &AMI; の **ecs-init** パッケージを更新するには

1. SSH 経由でコンテナインスタンスにログインします。
2. 次のコマンドを使用して、ecs-init パッケージを更新します。

```
sudo yum update -y ecs-init
```

> **Note**
>
> ecs-init パッケージと Amazon ECS コンテナエージェントが即座に更新されます。ただし、新しいバージョンの Docker は、Docker デーモンを再起動するまでロードされません。インスタンスを再起動するか、インスタンスで次のコマンドを実行して、再起動します。
>
> - Amazon ECS に最適化された Amazon Linux 2 AMI
>
>   ```
>   sudo systemctl restart docker
>   ```
>
> - Amazon ECS に最適化された Amazon Linux AMI
>
>   ```
>   sudo service docker restart && sudo start ecs
>   ```

UpdateContainerAgent API オペレーションで Amazon ECS コンテナエージェントを更新する

> **Important**
>
> -UpdateContainerAgentAPI は、Amazon ECS に最適化された AMI の Linux バリアントでのみサポートされます。ただし、Amazon ECS に最適化された Amazon Linux 2 (arm64) AMI は例外です。Amazon ECS に最適化された Amazon Linux 2 (arm64) AMI を使用するコンテナインスタンスの場合、ecs-initパッケージを使用してエージェントを更新します。他のオペレーティングシステムを実行しているコンテナインスタンスについては、「Amazon ECS コンテナエージェントの手動更新 (Amazon ECS 最適化以外の AMI の場合)」を参照してください。Windows コンテナインスタンスを使用している場合は、新しいコンテナインスタンスを起動して、Windows クラスター内のエージェントバージョンを更新することをお勧めします。

UpdateContainerAgentAPI 処理は、コンソールまたは AWS CLI または AWS SDKを使ってエージェントのアップデートを要求したときに始まります。Amazon ECS は、現在のエージェントバージョンと使用可能な最新バージョンを比較し、更新が可能かどうかを確認します。更新が利用できない場合 (例えばすでに最新バージョンがエージェントで実行されている場合) は、NoUpdateAvailableException が返されます。

更新プロセスのステージは、次のとおりです。

PENDING

エージェントを更新できます。更新プロセスが開始されました。

STAGING

エージェントで、エージェントの更新のダウンロードが開始されています。エージェントで更新をダウンロードできない場合や、更新の内容が正しくないか破損している場合、エージェントは失敗通知を送信し、更新は FAILED 状態に遷移します。

STAGED

エージェントのダウンロードが完了し、エージェントの内容が確認されました。

UPDATING

ecs-init サービスが再起動され、新しいエージェントバージョンが取得されます。エージェントが何らかの理由で再起動できない場合、更新は FAILED 状態に遷移します。それ以外の場合は、エージェントから Amazon ECS に更新完了のシグナルが送信されます。

> **Note**
>
> エージェント更新は Windows コンテナインスタンスに適用されません。Windows クラスター内のエージェントバージョンを更新するには、新しいコンテナインスタンスを起動することをお勧めします。

Amazon ECSに最適化されたAMIのAmazon ECS コンテナエージェントをコンソールでアップデートするには

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションバーから、外部インスタンスが存在するリージョンを選択します。
3. ナビゲーションペインで、[Clusters] を選択し、クラスターを選択します。

4. [Cluster : ***name***] (クラスター: 名前) のページで、[Infrastructure] (インフラストラクチャ) タブを選択します。
5. [コンテナインスタンス] で、更新するインスタンスを選択し、[アクション]、[エージェントの更新] を選択します。

Amazon ECS コンテナエージェントの手動更新 (Amazon ECS 最適化以外の AMI の場合)

場合によっては、バグの修正や新機能を取得するために Amazon ECS コンテナエージェントを更新する必要があります。Amazon ECS コンテナエージェントの更新によって、コンテナインスタンスで実行中のタスクやサービスが中断されることはありません。

> **Note**
>
> エージェント更新は Windows コンテナインスタンスに適用されません。Windows クラスター内のエージェントバージョンを更新するには、新しいコンテナインスタンスを起動することをお勧めします。

1. SSH 経由でコンテナインスタンスにログインします。
2. エージェントが ECS_DATADIR 環境変数を使用して状態を保存しているかどうかを確認します。

   ```
   ubuntu:~$ docker inspect ecs-agent | grep ECS_DATADIR
   ```

   出力:

   ```
   "ECS_DATADIR=/data",
   ```

   > **Important**
   >
   > 前のコマンドで ECS_DATADIR 環境変数が返されない場合は、エージェントを更新する前に、このコンテナインスタンスで実行されているタスクをすべて停止する必要があります。より新しいエージェントは ECS_DATADIR 環境変数を使用して状態を保存するため、タスクが実行中でも問題なく更新できます。

3. Amazon ECS コンテナエージェントを停止します。

```
ubuntu:~$ docker stop ecs-agent
```

4. エージェントコンテナを削除します。

```
ubuntu:~$ docker rm ecs-agent
```

5. /etc/ecs ディレクトリと Amazon ECS コンテナエージェント設定ファイルが /etc/ecs/ecs.config に存在することを確認します。

```
ubuntu:~$ sudo mkdir -p /etc/ecs && sudo touch /etc/ecs/ecs.config
```

6. /etc/ecs/ecs.config ファイルを編集して、少なくとも以下の変数宣言が必ず含まれるようにします。コンテナインスタンスをデフォルトのクラスターに登録しない場合は、クラスター名を ECS_CLUSTER の値として指定します。

```
ECS_DATADIR=/data
ECS_ENABLE_TASK_IAM_ROLE=true
ECS_ENABLE_TASK_IAM_ROLE_NETWORK_HOST=true
ECS_LOGFILE=/log/ecs-agent.log
ECS_AVAILABLE_LOGGING_DRIVERS=["json-file","awslogs"]
ECS_LOGLEVEL=info
ECS_CLUSTER=default
```

これらや他のエージェントランタイムオプションの詳細については、「Amazon ECS コンテナエージェントの設定」を参照してください。

> **Note**
>
> オプションで、エージェント環境変数を Amazon S3 に保存できます (これらの環境変数は、Amazon EC2 ユーザーデータを使用して、起動時にコンテナインスタンスにダウンロードできます)。プライベートリポジトリの認証情報などの機密情報については、この方法を推奨します。詳細については、「Amazon S3 に Amazon ECS コンテナインスタンスの設定を保存する」および「Amazon ECS での AWS 以外のコンテナイメージの使用」を参照してください。

7. Amazon Elastic Container Registry Public から最新の Amazon ECS コンテナエージェントイメージを取得します。

```
ubuntu:~$ docker pull public.ecr.aws/ecs/amazon-ecs-agent:latest
```

出力:

```
Pulling repository amazon/amazon-ecs-agent
a5a56a5e13dc: Download complete
511136ea3c5a: Download complete
9950b5d678a1: Download complete
c48ddcf21b63: Download complete
Status: Image is up to date for amazon/amazon-ecs-agent:latest
```

8. コンテナインスタンスで最新の Amazon ECS コンテナエージェントを実行します。

> **Note**
>
> Docker 再起動ポリシーまたはプロセスマネージャー (upstart または systemd など) を使用してコンテナエージェントをサービスまたはデーモンとして扱い、終了後に確実に再起動されるようにします。Amazon ECS に最適化された AMI はこのために `ecs-init` RPM を使用します。この[RPM のソースコード]は、GitHub で参照できます。

次のエージェント実行コマンドの例は、各オプションを示すために複数の行に分けられています。これらや他のエージェントランタイムオプションの詳細については、「Amazon ECS コンテナエージェントの設定」を参照してください。

> **Important**
>
> SELinux 対応オペレーティングシステムでは、`--privileged` コマンドに docker run オプションが必要です。さらに、SELinux 対応コンテナインスタンスの場合は、`:Z` および `/log` ボリュームマウントに `/data` オプションを追加することをお勧めします。ただし、コマンドを実行する前に、これらのボリュームのホストマウントが存在する必要があります。存在しないと、`no such file or directory` エラーが発生します。SELinux 対応コンテナインスタンスで Amazon ECS エージェントの実行に問題が発生する場合は、次のアクションを実行します。
>
> - コンテナインスタンスにホストボリュームマウントポイントを作成します。
>
>   ```
>   ubuntu:~$ sudo mkdir -p /var/log/ecs /var/lib/ecs/data
>   ```

- 次の docker run コマンドに `--privileged` オプションを追加します。
- 次の docker run コマンドで、`/log` および `/data` コンテナボリュームマウントに `:Z` オプションを追加します (`--volume=/var/log/ecs/:/log:Z` など)。

```
ubuntu:~$ sudo docker run --name ecs-agent \
--detach=true \
--restart=on-failure:10 \
--volume=/var/run:/var/run \
--volume=/var/log/ecs/:/log \
--volume=/var/lib/ecs/data:/data \
--volume=/etc/ecs:/etc/ecs \
--volume=/etc/ecs:/etc/ecs/pki \
--net=host \
--env-file=/etc/ecs/ecs.config \
amazon/amazon-ecs-agent:latest
```

> **Note**
>
> `Error response from daemon: Cannot start container` メッセージが表示された場合は、sudo docker rm ecs-agent コマンドを使用して障害のあるコンテナを削除し、再度エージェントの実行を試みることができます。

Amazon ECS に最適化された Windows AMI

Amazon ECS に最適化された AMI には、Amazon ECS ワークロードの実行に必要なコンポーネントがあらかじめ設定されています。Amazon ECSで一元化されたワークロードを実行するために必要な基本的な仕様を満たす独自のコンテナインスタンス AMI を作成することはできますが、Amazon ECS に最適化された AMI は事前設定され、AWS エンジニアにより Amazon ECS でテストされています。これは最も簡単に開始できる方法であり、AWS でコンピューティングリソースをすばやく実行できます。

Amazon ECS に最適化された AMI メタデータ (各バリアントの AMI 名、Amazon ECS コンテナエージェントバージョン、Docker バージョンを含む Amazon ECS ランタイムバージョンなど) は、プログラムで取得できます。詳細については、「the section called “Amazon ECS に最適化された Windows AMI メタデータを取得する”」を参照してください。

> **Important**
>
> 2022 年 8 月以降に作成された ECS に最適化された AMI バリアントはすべて Docker EE (Mirantis) から Docker CE (Moby プロジェクト) に移行されます。
> お客様がデフォルトで最新のセキュリティ更新を受けられるように、Amazon ECS は Windows ECS に最適化された AMI を 3 つ以上維持しています。Amazon ECS は、新しい Windows Amazon ECS に最適化された AMI をリリースした後、古いプライベートである Windows Amazon ECS に最適化された AMI を作成します。アクセスが必要なプライベート AMI がある場合は、クラウドSupport のチケットを提出してください。

Amazon ECS に最適化された AMI バリアント

Amazon EC2 インスタンスでは、Amazon ECS に最適化された AMI の次の Windows Server バリアントを使用できます。

> **Important**
>
> 8 月以降に製造された ECS に最適化された AMI バリアントはすべて Docker EE (Mirantis) から Docker CE (Moby プロジェクト) に移行される予定です。

- Amazon ECS に最適化された Windows Server 2025 Full AMI
- Amazon ECS に最適化された Windows Server 2025 Core AMI
- Amazon ECS に最適化された Windows Server 2022 Full AMI
- Amazon ECS に最適化された Windows Server 2022 Core AMI
- Amazon ECS に最適化された Windows Server 2019 Full AMI
- Amazon ECS に最適化された Windows Server 2019 Core AMI
- Amazon ECS に最適化された Windows Server 2016 Full AMI

> **Important**
>
> Windows Server 2016 は、25.x.x などの最新の Docker バージョンをサポートしていません。そのため、Windows Server 2016 Full AMI には Docker ランタイムに対するセキュリティパッチやバグパッチは適用されません。次の Windows プラットフォームのいずれかに移行することをお勧めします。

- Windows Server 2022 Full
- Windows Server 2022 Core
- Windows Server 2019 Full
- Windows Server 2019 Core

2022 年 8 月 9 日、Amazon ECS に最適化された Windows Server 20H2 Core AMI はサポートを終了しました。この AMI の新しいバージョンはリリースされません。詳細については、「Windows Server のリリース情報」を参照してください。

Windows Server 2025、Windows Server 2022、Windows Server 2019 および Windows Server 2016 は、長期サービスチャネル (LTSC) リリースです。Windows Server 20H2 は、半期チャネル (SAC) リリースです。詳細については、「Windows Server のリリース情報」を参照してください。

考慮事項

以下は、Amazon EC2 Windows コンテナと Amazon ECS についての留意点です。

- Windows コンテナは Linux コンテナインスタンスでは実行できません。逆の場合も同様です。Windows タスクと Linux タスクをより適切に配置するには、Windows コンテナインスタンスと Linux コンテナインスタンスを別々のクラスターに保持し、Windows タスクは Windows クラスターにのみ配置します。次の配置制約 `memberOf(ecs.os-type=='windows')` を設定して、Windows のタスク定義が Windows インスタンスのみに配置されるようにする必要があります。
- Windows コンテナは、EC2 および Fargate を使用するタスクでサポートされています。
- Windows コンテナとコンテナインスタンスでは、Linux コンテナとコンテナインスタンス用のタスク定義パラメータは全面的にはサポートされていません。まったくサポートされないパラメータもあり、Windows での動作と Linux での動作が異なるパラメータもあります。詳細については、「Windows を実行している EC2 インスタンスでの Amazon ECS タスク定義の違い」を参照してください。
- タスクの IAM ロール 機能については、起動時に機能を許可するように Windows コンテナインスタンスを設定する必要があります。コンテナは、この機能を使用するときに、指定された PowerShell コードの一部を実行する必要があります。詳細については、「Amazon EC2 Windows インスタンスの追加設定」を参照してください。
- タスク用の IAM ロール の機能では認証情報プロキシを使用してコンテナに認証情報を提供します。この認証情報プロキシは、コンテナインスタンスのポート 80 を占有するため、タスク用の

IAM ロール を使用する場合、タスクにポート 80 を使用することができません。ウェブサービスコンテナの場合は、 Application Load Balancer と動的なポートマッピングを使用して標準の HTTP ポート 80 接続をコンテナに提供できます。詳細については、「ロードバランサーを使用して Amazon ECS サービストラフィックを分散する」を参照してください。

- Windows サーバーのドッカーイメージは大きめです (9 GiB)。そのため、Windows コンテナインスタンスには Linux コンテナインスタンスよりも多くのストレージスペースが必要です。
- Windows Server で Windows コンテナを実行するには、コンテナのベースイメージの OS バージョンがホストのバージョンと一致する必要があります。詳細については、マイクロソフトのドキュメント Web サイトの「Windows コンテナバージョンの互換性」を参照してください。クラスターが複数の Windows バージョンを実行している場合、次の配置制約を使用して、同じバージョンで実行されている EC2 インスタンスにタスクが配置されるようにすることができます: `memberOf(attribute:ecs.os-family == WINDOWS_SERVER_<OS_Release>_<FULL or CORE>)`。詳細については、「the section called “Amazon ECS に最適化された Windows AMI メタデータを取得する”」を参照してください。

Amazon ECS に最適化された Windows AMI メタデータを取得する

Amazon ECS に最適化された AMI の各バリアントの AMI ID、イメージ名、オペレーティングシステム、コンテナエージェントバージョン、ランタイムバージョンは、Systems Manager パラメータストア API のクエリを実行してプログラムで取得できます。Systems Manager パラメータストア API の詳細については、「GetParameters」および「GetParametersByPath」を参照してください。

> **Note**
>
> Amazon ECS に最適化された AMI メタデータを取得するには、管理ユーザーに次の IAM アクセス権限が必要です。`AmazonECS_FullAccess` IAM ポリシーには、次の許可が追加されています。
>
> - ssm:GetParameters
> - ssm:GetParameter
> - ssm:GetParametersByPath

Systems Manager パラメータストアのパラメータフォーマット

> **Note**
>
> 以下の Systems Manager Parameter Store API パラメータは廃止されているため、最新の Windows AMI の取得には使用しないでください。
>
> - `/aws/service/ecs/optimized-ami/windows_server/2016/english/full/recommended/image_id`
> - `/aws/service/ecs/optimized-ami/windows_server/2019/english/full/recommended/image_id`

以下は、各 Amazon ECS に最適化された AMI バリアントのパラメータ名のフォーマットです。

- Windows Server 2025 Full AMI メタデータ:

```
/aws/service/ami-windows-latest/Windows_Server-2025-English-Full-ECS_Optimized
```

- Windows Server 2025 Core AMI メタデータ:

```
/aws/service/ami-windows-latest/Windows_Server-2025-English-Core-ECS_Optimized
```

- Windows Server 2022 Full AMI メタデータ:

```
/aws/service/ami-windows-latest/Windows_Server-2022-English-Full-ECS_Optimized
```

- Windows Server 2022 Core AMI メタデータ:

```
/aws/service/ami-windows-latest/Windows_Server-2022-English-Core-ECS_Optimized
```

- Windows Server 2019 フル AMI メタデータ:

```
/aws/service/ami-windows-latest/Windows_Server-2019-English-Full-ECS_Optimized
```

- Windows Server 2019 コア AMI メタデータ:

```
/aws/service/ami-windows-latest/Windows_Server-2019-English-Core-ECS_Optimized
```

- Windows Server 2016 フル AMI メタデータ:

```
/aws/service/ami-windows-latest/Windows_Server-2016-English-Full-ECS_Optimized
```

次のパラメーター名の形式は、最新の安定した Windows Server 2019 Full AMI のメタデータを取得します

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2019-
English-Full-ECS_Optimized
```

以下は、パラメータ値で返る JSON オブジェクトの例です。

```
{
    "Parameters": [
        {
            "Name": "/aws/service/ami-windows-latest/Windows_Server-2019-English-Full-
ECS_Optimized",
            "Type": "String",
            "Value": "{\"image_name\":\"Windows_Server-2019-English-Full-
ECS_Optimized-2023.06.13\",\"image_id\":\"ami-0debc1fb48e4aee16\",\"ecs_runtime_version
\":\"Docker (CE) version 20.10.21\",\"ecs_agent_version\":\"1.72.0\"}",
            "Version": 58,
            "LastModifiedDate": "2023-06-22T19:37:37.841000-04:00",
            "ARN": "arn:aws:ssm:us-east-1::parameter/aws/service/ami-windows-latest/
Windows_Server-2019-English-Full-ECS_Optimized",
            "DataType": "text"
        }
    ],
    "InvalidParameters": []
}
```

前述の出力の各フィールドは、サブパラメータとしてクエリに利用できます。サブパラメータのパラメータパスを構築するには、選択した AMI のパスにサブパラメータ名を追加します。以下のサブパラメータが利用可能です。

- schema_version
- image_id
- image_name
- os
- ecs_agent_version

- `ecs_runtime_version`

例

以下の例は、それぞれの Amazon ECS に最適化された AMI バリアントのメタデータを取得する方法を示しています。

安定している最新の Amazon ECS に最適化された AMI メタデータを取得する

安定している最新の Amazon ECS に最適化された AMI を取得するには、AWS CLI で次の AWS CLI コマンドを使用します。

- Amazon ECS に最適化された Windows Server 2025 Full AMI

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2025-English-Full-ECS_Optimized --region us-east-1
```

- Amazon ECS に最適化された Windows Server 2025 Core AMI

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2025-English-Core-ECS_Optimized --region us-east-1
```

- Amazon ECS に最適化された Windows Server 2022 Full AMI

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2022-English-Full-ECS_Optimized --region us-east-1
```

- Amazon ECS に最適化された Windows Server 2022 Core AMI

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2022-English-Core-ECS_Optimized --region us-east-1
```

- Amazon ECS に最適化された Windows Server 2019 Full AMI

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2019-English-Full-ECS_Optimized --region us-east-1
```

- Amazon ECS に最適化された Windows Server 2019 Core AMI

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2019-English-Core-ECS_Optimized --region us-east-1
```

- Amazon ECS に最適化された Windows Server 2016 Full AMI

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2016-English-Full-ECS_Optimized --region us-east-1
```

CloudFormation テンプレートの推奨される最新の Amazon ECS に最適化された AMI を使用する

Systems Manager パラメータストア名を参照することにより、CloudFormation テンプレートで推奨される最新の Amazon ECS に最適化された AMI を参照できます。

```
Parameters:
  LatestECSOptimizedAMI:
    Description: AMI ID
    Type: AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>
    Default: /aws/service/ami-windows-latest/Windows_Server-2019-English-Full-ECS_Optimized/image_id
```

Amazon ECS に最適化された Windows AMI バージョン

Amazon ECS に最適化された AMI の現在のバージョンと以前のバージョン、それらに対応する Amazon ECS コンテナエージェント、Docker、ecs-init パッケージのバージョンを表示します。

AMI ID を含む、Amazon ECS に最適化された AMI のメタデータでは、各バリアントをプログラム的に取得することができます。詳細については、「the section called “Amazon ECS に最適化された Windows AMI メタデータを取得する”」を参照してください。

次のタブには、Windows Amazon ECS に最適化された AMI のバージョンの一覧が表示されます。CloudFormation テンプレートで Systems Manager パラメータストアのパラメータを参照する方法については、CloudFormation テンプレートの推奨される最新の Amazon ECS に最適化された AMI を使用する を参照してください。

> **Important**
>
> お客様がデフォルトで最新のセキュリティ更新を受けられるように、Amazon ECS は Windows ECS に最適化された AMI を 3 つ以上維持しています。Amazon ECS は、新しい Windows Amazon ECS に最適化された AMI をリリースした後、古いプライベートである Windows Amazon ECS に最適化された AMI を作成します。アクセスが必要なプライベート AMI がある場合は、クラウドSupport のチケットを提出してください。

Windows Server 2016 は、25.x.x などの最新の Docker バージョンをサポートしていません。そのため、Windows Server 2016 Full AMI には Docker ランタイムに対するセキュリティパッチやバグパッチは適用されません。次の Windows プラットフォームのいずれかに移行することをお勧めします。

- Windows Server 2022 Full
- Windows Server 2022 Core
- Windows Server 2019 Full
- Windows Server 2019 Core

Note

gMSA プラグインのログ記録は、2025 年 8 月の AMI リリースで、ファイルベースのログ記録 (`C:\ProgramData\Amazon\gmsa`) から Windows Event logging に移行されました。パブリックログコレクタースクリプトは、すべての gMSA ログを収集します。詳細については、「Amazon ECS ログコレクターを使用したコンテナログの収集」を参照してください。

Windows Server 2025 Full AMI versions

次の表は、Amazon ECS に最適化された Windows Server 2025 Full AMI の現在のバージョンと以前のバージョン、およびそれらに対応するバージョンの Amazon ECS コンテナエージェントと Docker のリスト表示です。

Amazon ECS に最適化された Windows Server 2025 Full AMI	Amazon ECS コンテナエージェントバージョン	Docker バージョン	可視性
Windows_Server-2025-English-Full-ECS_Optimized-2026.08.15	`1.106.1`	`25.0.16 (Docker CE)`	Public
Windows_Server-2025-English-Full-ECS	`1.105.1`	`25.0.16 (Docker CE)`	Public

Amazon ECS に最適化された Windows Server 2025 Full AMI	Amazon ECS コンテナエージェントバージョン	Docker バージョン	可視性
_Optimized-2026.07.18			
Windows_Server-2025-English-Full-ECS_Optimized-2026.06.23	1.105.0	25.0.16 (Docker CE)	Public
Windows_Server-2025-English-Full-ECS_Optimized-2026.06.15	1.104.0	25.0.16 (Docker CE)	Public
Windows_Server-2025-English-Full-ECS_Optimized-2026.05.15	1.103.1	25.0.6 (Docker CE)	Public

現在の Amazon ECS に最適化された Windows Server 2025 Full AMI は、AWS CLI の次のコマンドで取得できます。

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2025-English-Full-ECS_Optimized
```

Windows Server 2025 Core AMI versions

次の表は、Amazon ECS に最適化された Windows Server 2025 Core AMI の現在のバージョンと以前のバージョン、およびそれらに対応するバージョンの Amazon ECS コンテナエージェントと Docker のリスト表示です。

Amazon ECS に最適化された Windows Server 2025 Core AMI	Amazon ECS コンテナエージェントバージョン	Docker バージョン	可視性
Windows_Server-2025-English-Core-ECS_Optimized-2026.08.15	1.106.1	25.0.16 (Docker CE)	Public
Windows_Server-2025-English-Core-ECS_Optimized-2026.07.18	1.105.1	25.0.16 (Docker CE)	Public
Windows_Server-2025-English-Core-ECS_Optimized-2026.06.23	1.105.0	25.0.16 (Docker CE)	Public
Windows_Server-2025-English-Core-ECS_Optimized-2026.06.15	1.104.0	25.0.16 (Docker CE)	Public
Windows_Server-2025-English-Core-ECS_Optimized-2026.05.15	1.103.1	25.0.6 (Docker CE)	Public

現在の Amazon ECS に最適化された Windows Server 2025 Core AMI は、AWS CLI の次のコマンドで取得できます。

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2025-English-Core-ECS_Optimized
```

Windows Server 2022 Full AMI versions

次の表は、Amazon ECS に最適化された Windows Server 2022 Full AMI の現在のバージョンと以前のバージョン、およびそれらに対応するバージョンの Amazon ECS コンテナエージェントと Docker のリスト表示です。

Amazon ECS に最適化された Windows Server 2022 Full AMI	Amazon ECS コンテナエージェントバージョン	Docker バージョン	可視性
Windows_Server-2022-English-Full-ECS_Optimized-2026.08.15	1.106.1	25.0.16 (Docker CE)	Public
Windows_Server-2022-English-Full-ECS_Optimized-2026.07.18	1.105.1	25.0.16 (Docker CE)	Public
Windows_Server-2022-English-Full-ECS_Optimized-2026.06.23	1.105.0	25.0.16 (Docker CE)	Public
Windows_Server-2022-English-Full-ECS_Optimized-2026.06.15	1.104.0	25.0.16 (Docker CE)	Public
Windows_Server-2022-English-Full-ECS_Optimized-2026.05.15	1.103.1	25.0.6 (Docker CE)	Public

現在の Amazon ECS に最適化された Windows Server 2022 Full AMI は、AWS CLI の次のコマンドで取得できます。

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2022-English-Full-ECS_Optimized
```

Windows Server 2022 Core AMI versions

次の表は、Amazon ECS に最適化された Windows Server 2022 Core AMI の現在のバージョンと以前のバージョン、およびそれらに対応するバージョンの Amazon ECS コンテナエージェントと Docker のリスト表示です。

Amazon ECS に最適化された Windows Server 2022 Core AMI	Amazon ECS コンテナエージェントバージョン	Docker バージョン	可視性
Windows_Server-2022-English-Core-ECS_Optimized-2026.08.15	1.106.1	25.0.16 (Docker CE)	Public
Windows_Server-2022-English-Core-ECS_Optimized-2026.07.18	1.105.1	25.0.16 (Docker CE)	Public
Windows_Server-2022-English-Core-ECS_Optimized-2026.06.23	1.105.0	25.0.16 (Docker CE)	Public
Windows_Server-2022-English-Core-ECS_Optimized-2026.06.16	1.104.0	25.0.16 (Docker CE)	Public
Windows_Server-2022-English-Core-ECS_Optimized-2026.05.15	1.103.1	25.0.6 (Docker CE)	Public

現在の Amazon ECS に最適化された Windows Server 2022 Full AMI は、AWS CLI の次のコマンドで取得できます。

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2022-English-Core-ECS_Optimized
```

Windows Server 2019 Full AMI versions

次の表は、Amazon ECS に最適化された Windows Server 2019 Full AMI の現在のバージョンと以前のバージョン、およびそれらに対応するバージョンの Amazon ECS コンテナエージェントと Docker のリスト表示です。

Amazon ECS に最適化された Windows Server 2019 Full AMI	Amazon ECS コンテナエージェントバージョン	Docker バージョン	可視性
Windows_Server-2019-English-Full-ECS_Optimized-2026.08.15	1.106.1	25.0.16 (Docker CE)	Public
Windows_Server-2019-English-Full-ECS_Optimized-2026.07.18	1.105.1	25.0.16 (Docker CE)	Public
Windows_Server-2019-English-Full-ECS_Optimized-2026.06.23	1.105.0	25.0.16 (Docker CE)	Public
Windows_Server-2019-English-Full-ECS_Optimized-2026.06.15	1.104.0	25.0.16 (Docker CE)	Public
Windows_Server-2019-English-Full-ECS	1.103.1	25.0.6 (Docker CE)	Public

Amazon ECS に最適化された Windows Server 2019 Full AMI	Amazon ECS コンテナエージェントバージョン	Docker バージョン	可視性
_Optimized-2026.05.15			

現在の Amazon ECS に最適化された Windows Server 2019 Full AMI は、AWS CLI の次のコマンドで取得できます。

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2019-English-Full-ECS_Optimized
```

Windows Server 2019 Core AMI versions

次の表は、Amazon ECS に最適化された Windows Server 2019 Core AMI の現在のバージョンと以前のバージョン、およびそれらに対応するバージョンの Amazon ECS コンテナエージェントと Docker のリスト表示です。

Amazon ECS に最適化された Windows Server 2019 Core AMI	Amazon ECS コンテナエージェントバージョン	Docker バージョン	可視性
Windows_Server-2019-English-Core-ECS_Optimized-2026.08.15	1.106.1	25.0.16 (Docker CE)	Public
Windows_Server-2019-English-Core-ECS_Optimized-2026.07.18	1.105.1	25.0.16 (Docker CE)	Public
Windows_Server-2019-English-Core-ECS	1.105.0	25.0.16 (Docker CE)	Public

Amazon ECS に最適化された Windows Server 2019 Core AMI	Amazon ECS コンテナエージェントバージョン	Docker バージョン	可視性
_Optimized-2026.06.23			
Windows_Server-2019-English-Core-ECS_Optimized-2026.06.15	1.104.0	25.0.16 (Docker CE)	Public
Windows_Server-2019-English-Core-ECS_Optimized-2026.05.15	1.103.1	25.0.6 (Docker CE)	Public

現在の Amazon ECS に最適化された Windows Server 2019 Full AMI は、AWS CLI の次のコマンドで取得できます。

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2019-English-Core-ECS_Optimized
```

Windows Server 2016 Full AMI versions

> **Important**
>
> Windows Server 2016 は、25.x.x などの最新の Docker バージョンをサポートしていません。そのため、Windows Server 2016 Full AMI には Docker ランタイムに対するセキュリティパッチやバグパッチは適用されません。次の Windows プラットフォームのいずれかに移行することをお勧めします。
>
> - Windows Server 2022 Full
> - Windows Server 2022 Core
> - Windows Server 2019 Full
> - Windows Server 2019 Core

次の表は、Amazon ECS に最適化された Windows Server 2016 Full AMI の現在のバージョンと以前のバージョン、およびそれらに対応するバージョンの Amazon ECS コンテナエージェントと Docker のリスト表示です。

Amazon ECS に最適化された Windows Server 2016 Full AMI	Amazon ECS コンテナエージェントバージョン	Docker バージョン	可視性
Windows_Server-2016-English-Full-ECS_Optimized-2026.08.15	1.106.1	20.10.23 (Docker CE)	Public
Windows_Server-2016-English-Full-ECS_Optimized-2026.07.18	1.105.1	20.10.23 (Docker CE)	Public
Windows_Server-2016-English-Full-ECS_Optimized-2026.06.23	1.105.0	20.10.23 (Docker CE)	Public
Windows_Server-2016-English-Full-ECS_Optimized-2026.06.12	1.104.0	20.10.23 (Docker CE)	Public
Windows_Server-2016-English-Full-ECS_Optimized-2026.05.15	1.103.1	20.10.23 (Docker CE)	Public

次の AWS CLI Amazon ECS に最適化された Windows Server 2016 Full AMI を使用します。

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2016-English-Full-ECS_Optimized
```

Amazon ECS に最適化された独自の Windows AMI の作成

EC2 Image Builder を使用して、Amazon ECS に最適化された独自のカスタム Windows AMI を構築します。これにより、Amazon ECS の独自のライセンスを持つ Windows AMI を簡単に使用できます。Amazon ECS は、コンテナをホストするために Windows インスタンスを実行するために必要なシステム設定を行うマネージド Image Builder コンポーネントを提供します。各 Amazon ECS マネージドコンポーネントには、特定のコンテナエージェントと Docker バージョンが含まれます。最新の Amazon ECS マネージドコンポーネントを使用するようにイメージをカスタマイズできます。また、古いコンテナエージェントまたは Docker バージョンが必要な場合は、別のコンポーネントを指定できます。

EC2 Image Builder 全体の使用に関するチュートリアルについては、「EC2 Image Builder ユーザーガイド」の「EC2 Image Builder入門」を参照してください。

EC2 Image Builder を使ってAmazon ECS に最適化された Windows AMI を構築する場合は、イメージ recipe を作成します。イメージ recipe は、次の要件を満たしている必要があります。

- [ソースイメージ] は、Windows Server 2019 Core、Windows Server 2019 Full、Windows Server 2022 Core、または Windows Server 2022 Full に基づいている必要があります。その他の Windows オペレーティングシステムはサポートされておらず、コンポーネントと互換性がない可能性があります。
- ビルドコンポーネントを指定する場合、`ecs-optimized-ami-windows` コンポーネントは必須です。`update-windows` コンポーネントをお勧めします。これにより、イメージに最新のセキュリティ更新プログラムが含まれていることを保証します。

  別のコンポーネントバージョンを指定するには、[Versioning options] メニューを展開し、使用するコンポーネントバージョンを指定します。詳細については、「`ecs-optimized-ami-windows` コンポーネントバージョンの一覧表示」を参照してください。

`ecs-optimized-ami-windows` コンポーネントバージョンの一覧表示

EC2 Image Builder recipe を作成して `ecs-optimized-ami-windows` コンポーネントを指定するときは、デフォルトのオプションを使用するか、特定のコンポーネントバージョンを指定できます。コンポーネント内に含まれる Amazon ECS コンテナエージェントおよび Docker バージョンとともに、使用可能なコンポーネントバージョンを確認するには、AWS マネジメントコンソール を使用します。

使用可能な **ecs-optimized-ami-windows** コンポーネントバージョンを一覧表示するには

1. で EC2 Image Builder コンソールを開きます。https://console.aws.amazon.com/imagebuilder/。
2. ナビゲーションバーで、イメージを構築しているリージョンを選択します。
3. ナビゲーションペインの [Saved configurations] メニューで、[Components] を選択します。
4. [Components] ページの検索バーに `ecs-optimized-ami-windows` を入力して認証情報メニューをプルダウンし、[Quick start (Amazon-managed)] を選択します。
5. [説明] 列を用いて、Amazon ECS コンテナエージェントを使用したコンポーネントのバージョンと、イメージに必要なバージョンの Docker バージョンを特定します。

Amazon ECS Windows コンテナインスタンスの管理

Amazon ECS ワークロードに EC2 インスタンスを使用する場合、インスタンスの維持はユーザーの責任となります。

エージェント更新は Windows コンテナインスタンスに適用されません。Windows クラスター内のエージェントバージョンを更新するには、新しいコンテナインスタンスを起動することをお勧めします。

管理手順

- Amazon ECS Windows コンテナインスタンスの起動
- Amazon ECS Windows コンテナインスタンスをブートストラップしてデータを渡す
- Amazon ECS Windows コンテナインスタンスに HTTP プロキシを使用する
- スポットインスタンス通知を受信するように Amazon ECS Windows コンテナインスタンスを設定する

Amazon ECS Windows コンテナインスタンスの起動

Amazon ECS コンテナインスタンスは、Amazon EC2 コンソールを使用して作成されます。開始する前に、Amazon ECS を使用するようにセットアップする のステップを完了するようにしてください。

起動ウィザードの詳細については、「Amazon EC2 ユーザーガイド」の「新しいインスタンス起動ウィザードを使用してインスタンスを起動する」を参照してください。

新しい Amazon EC2 ウィザードを使用してインスタンスを起動できます。パラメータには次のリストを使用できます。リストされていないパラメータは、デフォルトのままにしてください。次の手順では、各パラメータグループについて説明します。

手順

開始する前に、「Amazon ECS を使用するようにセットアップする」のステップを完了します。

1. Amazon EC2 コンソールの https://console.aws.amazon.com/ec2/ を開いてください。
2. 画面の上のナビゲーションバーで、現在の AWS リージョンが表示されます (例: 米国東部 (オハイオ))。インスタンスを起動するリージョンを選択します。一部の Amazon EC2 リソースはリージョン間で共有できるため、この選択は重要です。
3. Amazon EC2 コンソールダッシュボードで、[インスタンスを起動] を選択してください。

名前とタグ

インスタンス名はタグで、キーは [Name] (名前)、値は指定した名前です。インスタンス、ボリューム、および伸縮自在なグラフィックスにタグ付けできます。スポットインスタンスの場合、スポットインスタンスリクエストにのみタグを付けることができます。

インスタンス名と追加のタグを指定することはオプションです。

- [Name] (名前) に、インスタンスのわかりやすい名前を入力します。名前を指定しない場合は、インスタンスをその ID で識別できます。ID は、インスタンスの起動時に自動的に生成されます。
- タグを追加するには、[Add additional tag] (追加のタグを追加) を選択します。[Add tag] (タグを追加) を選択し、キーと値を入力し、タグ付けするリソースタイプを選択します。追加するタグごとに [Add tag] (タグの追加) を選択します。

アプリケーションと OS イメージ (Amazon マシンイメージ)

Amazon マシンイメージ (AMI) には、インスタンスの作成に必要な情報が含まれています。例えば、ある AMI には、ウェブサーバーとして動作するために必要なソフトウェア (Apache やウェブサイトなど) が含まれています。

最新の Amazon ECS 最適化 AMI とその値については、「Windows Amazon ECS-optimized AMI」(Windows Amazon ECS に最適化された AMI) を参照してください。

[Search] (検索) バーを使用して、AWS によって発行された適切な Amazon ECS 最適化 AMI を検索します。

1. 要件に基づいて、[Search] (検索) バーに次の AMI のいずれかを入力し、[Enter] を押します。
   - Windows_Server-2022-English-Full-ECS_Optimized
   - Windows_Server-2022-English-Core-ECS_Optimized
   - Windows_Server-2019-English-Full-ECS_Optimized
   - Windows_Server-2019-English-Core-ECS_Optimized
   - Windows_Server-2016-English-Full-ECS_Optimized
2. [Choose an Amazon Machine Image (AMI]) (Amazon マシンイメージ (AMI) を選択) ページで、[Community AMIs] (コミュニティ AMI) タブを選択します。
3. 表示されるリストから、発行日が最新の Microsoft 検証済み AMI を選択し、[Select] (選択) をクリックします。

インスタンスタイプ

インスタンスタイプは、インスタンスのハードウェア設定とサイズを定義します。インスタンスタイプが大きくなると、CPU およびメモリも増えます。詳細については、インスタンスタイプを参照してください。

- [Instance type] (インスタンスタイプ) で、インスタンスのインスタンスタイプを選択します。

  選択したインスタンスタイプによって、タスクの実行に使用できるリソースが決まります。

キーペア (ログイン)

[Key pair name] (キーペア名) には、既存のキーペアを選択するか、[Create new key pair] (新しいキーペアを作成) を選択して新しいキーペアを作成します。

> **Important**
>
> [Proceed without key pair] (キーペアなしで進む) オプションを選択した場合 (非推奨)、ユーザーが別の方法でログインすることを許可するように設定された AMI を選択した場合でなければ、インスタンスに接続できなくなります。

ネットワーク設定

必要に応じて、ネットワーク設定を設定します。

- [Networking platform] (ネットワーキングプラットフォーム): [Virtual Private Cloud (VPC)] を選択して、[Network interfaces] (ネットワークインターフェイス) セクションでサブネットを指定します。
- [VPC]: セキュリティグループを作成する既存の VPC を選択します。
- [サブネット]: インスタンスは、アベイラビリティーゾーン、ローカルゾーン、Wavelength Zone、Outpost のいずれかに関連付けられたサブネットで起動できます。

  アベイラビリティーゾーンでインスタンスを起動するには、インスタンスを起動するサブネットを選択します。新しいサブネットを作成するには、[Create new subnet] を選択して Amazon VPC コンソールに移動します。終了したらインスタンス起動ウィザードに戻り、[Refresh] (更新) アイコンを選択して一覧にサブネットを読み込みます。

  ローカルゾーンでインスタンスを起動するには、ローカルゾーン内に作成したサブネットを選択します。

  アウトポストでインスタンスを起動するには、アウトポストに関連付けられた VPC 内のサブネットを選択します。
- [Auto-assign Public IP] (パブリック IP の自動割り当て): インスタンスをインターネットからアクセス可能にする場合は、[Auto-assign Public IP] (パブリック IP の自動割り当て) フィールドが [Enable] (有効) に設定されていることを確認します。可能にしない場合には、このフィールドを [無効] に設定します。

  > **Note**
  >
  > コンテナインスタンスには、Amazon ECS サービスエンドポイントと通信するためのアクセスが必要です。これは、インターフェイス VPC エンドポイントを介して、またはパブリック IP アドレスを持つコンテナインスタンスを通じて可能になります。
  > インターフェイス VPC エンドポイントについての詳細は、「Amazon ECS とインターフェイス VPC エンドポイント (AWS PrivateLink)」を参照してください。
  > インターフェイス VPC エンドポイントが設定されておらず、コンテナインスタンスがパブリック IP アドレスを持たない場合、ネットワークアドレス変換 (NAT) を使用してこのアクセスを提供する必要があります。詳細については、「Amazon VPC ユーザーガイド」の「NAT ゲートウェイ」、およびこのガイドの「Amazon ECS Linux コンテナインスタンスに HTTP プロキシを使用する」を参照してください。
- [Firewall (security groups)] (ファイアウォール (セキュリティグループ)): セキュリティグループを使用して、コンテナインスタンスのファイアウォールルールを定義します。このルールでは、ど

の着信ネットワークトラフィックをコンテナインスタンスに配信するかを指定します。他のトラフィックはすべて無視されます。

- 既存のセキュリティグループを選択するには、[Select existing security group] (既存のセキュリティグループを選択) を選択し、Amazon ECS を使用するようにセットアップする で作成したセキュリティグループを選択します。

ストレージの設定

選択した AMI には、ルートボリュームを含む、1 つまたは複数のストレージボリュームが含まれます。インスタンスにアタッチする追加のボリュームを指定できます。

[Simple] (シンプル) ビューを使用できます。

- [Storage type] (ストレージタイプ): コンテナインスタンスのストレージを設定します。

  Amazon ECS に最適化された Amazon Linux AMI を使用している場合、インスタンスには 2 つのボリュームが設定されます。[Root] ボリュームはオペレーティングシステム用で、2 番目の Amazon EBS ボリューム (`/dev/xvdcz` にアタッチ) は Docker 用です。

  オプションで、アプリケーションのニーズに合わせてインスタンスのボリュームサイズを増減できます。

高度な詳細

[Advanced details] で、セクションを開いてフィールドを表示し、インスタンスの追加パラメータを指定します。

- [Purchasing option] (購入のオプション): [Request Spot Instances] (スポットインスタンスのリクエスト) を選択して、スポットインスタンスをリクエストします。また、スポットインスタンスに関連する他のフィールドも設定する必要があります。詳細については、「スポットインスタンスのリクエスト」を参照してください。

  > **Note**
  >
  > スポットインスタンスを使用していて、"`Not available`" メッセージが表示される場合は、別のインスタンスタイプを選択する必要があります。

.

- [IAM instance profile] (IAM インスタンスプロフィール): コンテナインスタンス IAM ロールを選択します。通常、これは `ecsInstanceRole` という名前です。

  > **Important**
  >
  > 適切な IAM アクセス許可を使用してコンテナインスタンスを起動しないと、Amazon ECS エージェントはクラスターに接続できません。詳細については、「Amazon ECS コンテナインスタンスの IAM ロール」を参照してください。

- (オプション) [User data] (ユーザーデータ): Amazon ECS コンテナエージェントの設定 からのエージェント環境変数のようなユーザーデータを使用して、Amazon ECS コンテナインスタンスを設定します。Amazon EC2 ユーザーデータスクリプトはインスタンスの初回起動時に 1 回のみ実行されます。以下に、ユーザーデータを使用する目的の一般的な例を紹介します。

  - デフォルトでは、コンテナインスタンスはデフォルトのクラスターで起動されます。デフォルト以外のクラスターで起動するには、[Advanced Details] (高度な詳細) リストを選択します。次に、[User data] フィールドに以下のスクリプトを貼り付け、*`your_cluster_name`* を使用するクラスターの名前に置き換えます。

    `EnableTaskIAMRole`は、タスクの Task IAM ロール機能をオンにします。

    さらに、以下のオプションは、`awsvpc` ネットワークモードを使用する場合に使用できます。

    - `EnableTaskENI`: このフラグは、`awsvpc`ネットワークモードを使用する場合に、タスクネットワークをオンにします。
    - `AwsvpcBlockIMDS`: `awsvpc`ネットワークモードで実行中のタスクコンテナが IMDS にアクセスするのをブロックします。
    - `AwsvpcAdditionalLocalRoutes`: このオプションフラグを使用すると、タスクの名前空間に追加のルートを持つことができます。

      置換 `ip-address`を追加ルートの IP アドレス (例えば 172.31.42.23/32) に置き換えます。

    ```
    <powershell>
    Import-Module ECSTools
    Initialize-ECSAgent -Cluster your_cluster_name -EnableTaskIAMRole -EnableTaskENI -
    AwsvpcBlockIMDS -AwsvpcAdditionalLocalRoutes
    '["ip-address"]'
    </powershell>
    ```

Amazon ECS Windows コンテナインスタンスをブートストラップしてデータを渡す

Amazon EC2 インスタンスを起動するときに、ユーザーデータを EC2 インスタンスに渡すことができます。インスタンスの起動時に、データを使って、一般的な自動設定タスクを実行したり、スクリプトを実行したりできます。Amazon ECS では、ユーザーデータの最も一般的なユースケースは、設定情報を Docker デーモンと Amazon ECS コンテナエージェントに渡すことです。

クラウドブートフック、シェルスクリプト、`cloud-init` ディレクティブなど、複数タイプのユーザーデータを Amazon EC2 に渡すことができます。これらおよびその他の形式の種類の詳細については、「Cloud-Init のドキュメント」を参照してください。

Amazon EC2 起動ウィザードを使用するときに、このユーザーデータを渡すことができます。詳細については、「Amazon ECS Linux コンテナインスタンスの起動」を参照してください。

デフォルト Windows ユーザーデータ

このユーザーデータスクリプト例で、コンソールを使用する場合、Windows コンテナインスタンスが受け取るデフォルトのユーザーデータが確認できます。次のスクリプトは、次を実行します。

- クラスター名に入力した名前を設定します。
- タスクの IAM ロールを設定します。
- `json-file` および `awslogs` を使用可能なロギングドライバーとして設定します。

さらに、以下のオプションは、`awsvpc` ネットワークモードを使用する場合に使用できます。

- `EnableTaskENI`: このフラグは、`awsvpc`ネットワークモードを使用する場合に、タスクネットワークをオンにします。
- `AwsvpcBlockIMDS`: このオプションのフラグは、`awsvpc`ネットワークモードで実行されているタスクコンテナに対する IMDS アクセスをブロックします。
- `AwsvpcAdditionalLocalRoutes`: このオプションフラグを使用すると、追加のルートを持つことができます。

  置換 `ip-address`を追加ルートの IP アドレス (たとえば 172.31.42.23/32) に置き換えます。

このスクリプトは、独自のコンテナインスタンスに使用できます (Amazon ECS 最適化　Windows Server AMI から起動される場合)。

`-Cluster` *`cluster-name`* 行を置き換えて、独自のクラスター名を指定します。

```
<powershell>
Initialize-ECSAgent -Cluster cluster-name -EnableTaskIAMRole -LoggingDrivers '["json-
file","awslogs"]' -EnableTaskENI -AwsvpcBlockIMDS -AwsvpcAdditionalLocalRoutes
'["ip-address"]'
</powershell>
```

awslogs ログドライバーを使用するように設定された Windows タスクの場合は、コンテナインスタンスで ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE 環境変数も設定する必要があります。以下の構文を使用します。

-Cluster *cluster-name* 行を置き換えて、独自のクラスター名を指定します。

```
<powershell>
[Environment]::SetEnvironmentVariable("ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE",
 $TRUE, "Machine")
Initialize-ECSAgent -Cluster cluster-name -EnableTaskIAMRole -LoggingDrivers '["json-
file","awslogs"]'
</powershell>
```

Windows エージェントのインストールユーザーデータ

この例のユーザーデータスクリプトは、Windows_Server-2016-English-Full-Containers AMI で起動されたインスタンスにAmazon ECS コンテナエージェントをインストールします。これは、Amazon ECS コンテナエージェントの GitHub リポジトリ README ページのエージェントのインストール手順から変更されています。

> **Note**
>
> このスクリプトは、サンプル目的で共有されます。Amazon ECS に最適化された Windows AMI を使用すると、Windows コンテナの使用を開始するほうが、はるかに簡単です。詳細については、「Fargate ワークロード用の Amazon ECS クラスターを作成する」を参照してください。

Windows Server 2022 Full に Amazon ECS エージェントをインストールする方法については、GitHub の「Issue 3753」を参照してください。

このスクリプトは、独自のコンテナインスタンスに使用できます (それらのインスタンスが Windows_Server-2016-English-Full-Containers AMI のバージョンで起動される場合)。*windows* 行を置き換えて、独自のクラスターの名前を指定します (windows クラスターを使用しない場合)。

```
<powershell>
# Set up directories the agent uses
New-Item -Type directory -Path ${env:ProgramFiles}\Amazon\ECS -Force
New-Item -Type directory -Path ${env:ProgramData}\Amazon\ECS -Force
New-Item -Type directory -Path ${env:ProgramData}\Amazon\ECS\data -Force
# Set up configuration
$ecsExeDir = "${env:ProgramFiles}\Amazon\ECS"
[Environment]::SetEnvironmentVariable("ECS_CLUSTER", "windows", "Machine")
[Environment]::SetEnvironmentVariable("ECS_LOGFILE", "${env:ProgramData}\Amazon\ECS\log
\ecs-agent.log", "Machine")
[Environment]::SetEnvironmentVariable("ECS_DATADIR", "${env:ProgramData}\Amazon\ECS
\data", "Machine")
# Download the agent
$agentVersion = "latest"
$agentZipUri = "https://s3.amazonaws.com/amazon-ecs-agent/ecs-agent-windows-
$agentVersion.zip"
$zipFile = "${env:TEMP}\ecs-agent.zip"
Invoke-RestMethod -OutFile $zipFile -Uri $agentZipUri
# Put the executables in the executable directory.
Expand-Archive -Path $zipFile -DestinationPath $ecsExeDir -Force
Set-Location ${ecsExeDir}
# Set $EnableTaskIAMRoles to $true to enable task IAM roles
# Note that enabling IAM roles will make port 80 unavailable for tasks.
[bool]$EnableTaskIAMRoles = $false
if (${EnableTaskIAMRoles}) {
  $HostSetupScript = Invoke-WebRequest https://raw.githubusercontent.com/aws/amazon-
ecs-agent/master/misc/windows-deploy/hostsetup.ps1
  Invoke-Expression $($HostSetupScript.Content)
}
# Install the agent service
New-Service -Name "AmazonECS" `
        -BinaryPathName "$ecsExeDir\amazon-ecs-agent.exe -windows-service" `
        -DisplayName "Amazon ECS" `
        -Description "Amazon ECS service runs the Amazon ECS agent" `
        -DependsOn Docker `
        -StartupType Manual
sc.exe failure AmazonECS reset=300 actions=restart/5000/restart/30000/restart/60000
sc.exe failureflag AmazonECS 1
Start-Service AmazonECS
</powershell>
```

Amazon ECS Windows コンテナインスタンスに HTTP プロキシを使用する

Amazon ECS コンテナエージェントと Docker デーモンの両方に HTTP プロキシを使用するように Amazon ECS コンテナインスタンスを設定できます。これは、コンテナインスタンスに、Amazon VPC インターネットゲートウェイ、NAT ゲートウェイ、またはインスタンスを介した外部ネットワークアクセスがない場合に便利です。

Amazon ECS Windows コンテナインスタンスが HTTP プロキシを使用するように設定するには、起動時に (Amazon EC2ユーザーデータを使用して) 以下の変数を設定します。

```
[Environment]::SetEnvironmentVariable("HTTP_PROXY",
"http://proxy.mydomain:port", "Machine")
```

HTTP_PROXY を、Amazon ECS エージェントがインターネットへの接続に使用する HTTP プロキシのホスト名 (または IP アドレス) とポート番号に設定します。例えば、コンテナインスタンスに、Amazon VPC インターネットゲートウェイ、NAT ゲートウェイ、またはインスタンスを介した外部ネットワークアクセスがない場合です。

```
[Environment]::SetEnvironmentVariable("NO_PROXY",
"169.254.169.254,169.254.170.2,\\.\pipe\docker_engine", "Machine")
```

NO_PROXY を 169.254.169.254,169.254.170.2,\\.\pipe\docker_engine に設定して、EC2 インスタンスのメタデータ、タスク用の IAM ロール、および Docker デーモンのトラフィックをプロキシからフィルタリングします。

Example Windows HTTP プロキシのユーザーデータスクリプト

次のユーザーデータ PowerShell スクリプトの例では、 Amazon ECS コンテナエージェント、および Docker デーモンが指定された HTTP プロキシを使用するように設定します。コンテナインスタンス自体が登録されているクラスターを指定することもできます。

コンテナインスタンスの起動時にこのスクリプトを使用するには、「the section called “コンテナインスタンスの起動”」のステップに従います。次の PowerShell スクリプトをコピーして [ユーザーデータ] フィールドに貼り付けます (必ず、赤いサンプル値を実際のプロキシおよびクラスターの情報に置き換えてください)。

> **Note**
>
> この -EnableTaskIAMRole オプションは、タスクの IAM ロールを有効にするために必要です。詳細については、「Amazon EC2 Windows インスタンスの追加設定」を参照してください。

```
<powershell>
Import-Module ECSTools

$proxy = "http://proxy.mydomain:port"
[Environment]::SetEnvironmentVariable("HTTP_PROXY", $proxy, "Machine")
[Environment]::SetEnvironmentVariable("NO_PROXY", "169.254.169.254,169.254.170.2,\\.
\pipe\docker_engine", "Machine")

Restart-Service Docker
Initialize-ECSAgent -Cluster MyCluster -EnableTaskIAMRole
</powershell>
```

スポットインスタンス通知を受信するように Amazon ECS Windows コンテナインスタンスを設定する

利用可能なキャパシティーがなくなった場合、または、スポット料金がお客様のリクエストの上限料金を超えた場合には、Amazon EC2 はスポットインスタンスを終了、停止、または休止状態にします。Amazon EC2 はスポットインスタンスが中断される 2 分前に、そのインスタンスに対し中断を警告するための通知を送信します。インスタンスで Amazon ECS スポットインスタンスのドレインが有効になっている場合、ECS はスポットインスタンスの中断通知を受け取り、インスタンスを DRAINING ステータスにします。

> **Important**
>
> Amazon ECS は、terminate および stop インスタンスアクションがあるスポットインスタンスの中断通知をモニタリングします。スポットインスタンスまたはスポットフリートのリクエスト時に hibernate インスタンスの中断動作を指定した場合、Amazon ECS スポットインスタンスのドレインはこれらのインスタンスではサポートされません。

コンテナインスタンスを DRAINING に設定すると、Amazon ECS によって新規タスクがそのコンテナインスタンスに配置されなくなります。ドレインしているコンテナインスタンス上にある

PENDING 状態のサービスタスクは即時停止されます。クラスター内に使用可能なコンテナインスタンスがある場合、そのインスタンスで代わりのサービスタスクが開始されます。

インスタンスの起動時にスポットインスタンスドレイニングを有効にできます。コンテナエージェントを開始する前に ECS_ENABLE_SPOT_INSTANCE_DRAINING パラメータを設定する必要があります。*my-cluster* の部分は自分のクラスター名に置き換えます。

```
[Environment]::SetEnvironmentVariable("ECS_ENABLE_SPOT_INSTANCE_DRAINING", "true",
 "Machine")

# Initialize the agent
Initialize-ECSAgent -Cluster my-cluster
```

詳細については、「the section called “コンテナインスタンスの起動”」を参照してください。

外部インスタンスの Amazon ECS クラスター

Amazon ECS Anywhere は、オンプレミスサーバーや仮想マシン (VM) などの外部インスタンスを Amazon ECS クラスターに登録するためのサポートを提供します。外部インスタンスは、アウトバウンドトラフィックを生成したり、データを処理したりするアプリケーションを実行するために最適化されています。アプリケーションがインバウンドトラフィックを必要とする場合、Elastic Load Balancing のサポートがないため、これらのワークロードの実行効率が低下します。Amazon ECS は、新しいEXTERNAL起動タイプで、サービスを作成したり、外部インスタンスでタスクを実行したりできます。

サポートされるオペレーティングシステムとシステムアーキテクチャ

以下は、サポートされているオペレーティングシステムのリストです。x86_64およびARM64CPU アーキテクチャがサポートされています。

- Amazon Linux 2023
- Ubuntu 20、Ubuntu 22、Ubuntu 24
- RHEL 9 – ECS Anywhere インストールスクリプトを実行する前に、Docker がインストールされていることを確認する必要があります。詳細については、Docker ドキュメントの「Install Docker Engine on RHEL」を参照してください。

以下のオペレーティングシステムは、2026 年 8 月 7 日付けで Amazon ECS Anywhere のサポート対象外になります。

- Amazon Linux 2
- CentOS Stream 9
- RHEL 7、RHEL 8
- Fedora 32、Fedora 33、Fedora 40
- openSUSE タンブルウィード
- Ubuntu 18
- Debian 9、Debian 10、Debian 11、Debian 12
- SUSE Enterprise Server 15
- Windows Server 2022、Windows Server 2019、Windows Server 2016、Windows Server 20H2

考慮事項

外部インスタンスの使用を開始する前に、以下の考慮事項に注意してください。

- 外部インスタンスは、一度に 1 つずつクラスターに登録できます。外部インスタンスを別のクラスターに登録する方法については、「Amazon ECS 外部インスタンスの登録を解除する」を参照してください。
- 外部インスタンスには、AWS API との通信を許可する IAM ロールが必要です。詳細については、「Amazon ECS Anywhere IAM ロール」を参照してください。
- 外部インスタンスには、事前設定されたインスタンス認証情報チェーンをローカルに定義しないでください。これは、登録スクリプトに干渉するためです。
- コンテナログを CloudWatch Logs に送信するには、タスク定義でタスク実行 IAM ロールを作成し、指定してください。
- 外部インスタンスがクラスターに登録されると、`ecs.capability.external`属性がインスタンスに関連付けられています。この属性は、インスタンスを外部インスタンスとして識別します。カスタム属性を外部インスタンスに追加して、タスクの配置制約として使用できます。詳細については、「カスタム属性」を参照してください。
- 外部インスタンスにリソースタグを追加できます。詳細については、「Amazon ECS の外部コンテナインスタンスにタグを追加する」を参照してください。
- ECS Exec は、外部インスタンスでサポートされています。詳細については、「ECS Exec を使用して Amazon ECS コンテナをモニタリングする」を参照してください。
- 外部インスタンスとのネットワーキングに固有の追加の考慮事項を次に示します。詳細については、「ネットワーク」を参照してください。

- サービスの負荷分散はサポートされていません。
- サービス検出はサポートされていません。
- 外部インスタンスで実行されるタスクは、`bridge`,`host`, または`none` ネットワークモードを使用する必要があります。`awsvpc` ネットワークモードはサポートされていません。
- 各 AWS リージョンに Amazon ECS サービスドメインがあります。これらのサービスドメインは、外部インスタンスへのトラフィックの送信を許可する必要があります。
- 外部インスタンスにインストールされた SSM Agent は、ハードウェアフィンガープリントを使用して 30 分ごとにローテーションされる IAM 認証情報を保持します。外部インスタンスがAWSに設定されている場合、SSM Agent は接続の再確立後にクレデンシャルを自動的に更新します。詳細については、AWS Systems Managerユーザーガイドの「ハードウェアフィンガープリントを使用したオンプレミスサーバーと仮想マシンの検証」を参照してください。
- インスタンスが IPv6 のみのサブネットにある限り、IPv6 のみの設定の外部インスタンスで Linux タスクを実行できます。詳細については、「IPv6-only モードで VPC を使用する」を参照してください。

- `UpdateContainerAgent` API はサポートされません。外部インスタンスで SSM Agent または Amazon ECS エージェントを更新する方法については、「外部インスタンス上の AWS Systems Manager エージェントと Amazon ECS コンテナエージェントを更新する」を参照してください。
- Amazon ECS キャパシティープロバイダーはサポートされていません。外部インスタンスでサービスを作成したり、スタンドアロンタスクを実行するには、`EXTERNAL`起動タイプを使用するタスクにのみ使用されます。
- SELinux はサポートされません。
- Amazon EFS ボリュームの使用、または `EFSVolumeConfiguration` はサポートされていません。
- App Mesh との統合はサポートされていません。
- コンソールを使用して外部インスタンスタスク定義を作成する場合は、コンソール JSON エディタでタスク定義を作成する必要があります。
- Amazon ECS に最適化されていない AMI を使用する場合は、外部コンテナインスタンスで次のコマンドを実行して、タスクに IAM ロールを使用するルールを設定します。詳細については、「外部インスタンスの追加設定」を参照してください。

```
$ sysctl -w net.ipv4.conf.all.route_localnet=1
$ iptables -t nat -A PREROUTING -p tcp -d 169.254.170.2 --dport 80 -j DNAT --to-
destination 127.0.0.1:51679
```

```
$ iptables -t nat -A OUTPUT -d 169.254.170.2 -p tcp -m tcp --dport 80 -j REDIRECT --
to-ports 51679
```

ネットワーク

Amazon ECS 外部インスタンスは、アウトバウンドトラフィックを生成したり、データを処理したりするアプリケーションを実行するために最適化されています。アプリケーションがウェブサービスなどのインバウンドトラフィックを必要とする場合、Elastic Load Balancing のサポートがないため、これらのワークロードをロードバランサーの背後に配置するためのサポートがないため、これらのワークロードの実行効率が低下します。

外部インスタンスとのネットワークに固有の追加の考慮事項を次に示します。

- サービスの負荷分散はサポートされていません。
- サービス検出はサポートされていません。
- 外部インスタンスで実行される Linux タスクは、`bridge`、`host`、またはnone ネットワークモードを使用する必要があります。`awsvpc` ネットワークモードはサポートされていません。

  各ネットワークモードの詳細については、「EC2 の Amazon ECS タスクネットワークオプション」を参照してください。
- インスタンスが IPv6 のみのサブネットにある限り、IPv6 のみの設定の外部インスタンスで Linux タスクを実行できます。詳細については、「IPv6-only モードで VPC を使用する」を参照してください。
- 各リージョンには Amazon ECS サービスドメインがあり、外部インスタンスへのトラフィックの送信を許可する必要があります。
- 外部インスタンスにインストールされた SSM Agent は、ハードウェアフィンガープリントを使用して 30 分ごとにローテーションされる IAM 認証情報を保持します。外部インスタンスがAWSに設定されている場合、SSM Agent は接続の再確立後にクレデンシャルを自動的に更新します。詳細については、AWS Systems Managerユーザーガイドの「ハードウェアフィンガープリントを使用したオンプレミスサーバーと仮想マシンの検証」を参照してください。

次のドメインは、Amazon ECS サービスと外部インスタンスにインストールされている Amazon ECS エージェント間の通信に使用されます。トラフィックが許可されていることと、DNS 解決が機能していることを確認します。各エンドポイントでは、*#####*は、米国東部 (オハイオ) リージョンの `us-east-2` のように、Amazon ECS でサポートされている AWS リージョンのリージョン識別子を表します。使用するすべてのリージョンのエンドポイントを許可する必要があります。`ecs-a`お

よびecs-tエンドポイントを使用する場合は、アスタリスク (例えば、ecs-a-*) を含める必要があります。

- ecs-a-*.*region*.amazonaws.com— このエンドポイントは、タスクを管理するときに使用されます。
- ecs-t-*.*region*.amazonaws.com— このエンドポイントは、タスクとコンテナのメトリクスを管理するために使用されます。
- ecs.*region*.amazonaws.com — これは、Amazon ECS のサービスエンドポイントです。
- ssm.*region*.amazonaws.com — これは、 のサービスエンドポイントです。AWS Systems Manager
- ec2messages.*region*.amazonaws.com — これは、AWS Systems Manager がクラウド内の Systems Manager エージェントと Systems Manager サービスの間の通信に使用するサービスエンドポイントです。
- ssmmessages.*region*.amazonaws.com — これは、クラウド内の Session Manager サービスでセッションチャネルを作成および削除するために必要なサービスエンドポイントです。
- タスクが他のタスクとの通信を必要とする場合AWSサービスを使用する場合は、これらのサービスエンドポイントが許可されていることを確認します。アプリケーション例としては、Amazon ECR を使用してコンテナイメージを取得したり、CloudWatch Logs に CloudWatch を使用したりすることが挙げられます。詳細については、AWS 全般のリファレンスガイドの「サービスエンドポイント」を参照してください。

Amazon FSx for Windows File ServerECS Anywhere を使用した

> **Important**
>
> Amazon ECS Anywhere の Windows サポートは廃止されました。このセクションは適用されなくなりました。

Amazon ECS 外部インスタンスで Amazon FSx for Windows File Server を使用するには、オンプレミスのデータセンターと AWS クラウド の間に接続を確立する必要があります。ネットワークを VPC に接続するオプションについては、「Amazon Virtual Private Cloud 接続オプション」を参照してください。

ECS Anywhere を使用した gMSA

> **Important**
>
> Amazon ECS Anywhere の Windows サポートは廃止されました。このセクションは適用されなくなりました。

以下のユースケースは、Windows がサポート対象オペレーティングシステムであったときに ECS Anywhere でサポートされていたものです。

- アクティブディレクトリは、AWS クラウド にあります。この構成では、AWS クラウド 接続を使用してオンプレミスネットワークと AWS Direct Connect の間の接続を確立します。接続を作成する方法については、「Amazon 仮想プライベートクラウド接続オプション」を参照してください。AWS クラウド にアクティブディレクトリを作成します。AWS Directory Service の使用開始方法の詳細については、「AWS Directory Service 管理ガイド」の「AWS Directory Service の設定」を参照してください。その後、AWS Direct Connect 接続を使用して、外部インスタンスをドメインに参加させることができます。Amazon ECS での gMSA の操作方法については、「the section called “EC2 Windows コンテナで gMSAs を使用する方法について説明します。”」を参照してください。
- アクティブディレクトリは、オンプレミスデータセンターで管理されています。-この設定では、外部インスタンスをオンプレミスのアクティブディレクトリに参加させます。その後、Amazon ECS タスクを実行する際に、ローカルで使用可能な認証情報を使用します。

外部インスタンスワークロード用の Amazon ECS クラスターを作成する

クラスターを作成して、タスクとサービスを実行するインフラストラクチャを定義します。

これを開始する前に、「Amazon ECS を使用するようにセットアップする」 の手順を完了し、適切な IAM 許可を割り当てる必要があります。詳細については、「the section called “Amazon ECS クラスターの例”」を参照してください。Amazon ECS コンソールでは、CloudFormation スタックを作成することで、Amazon ECS クラスターに必要なリソースを簡単に作成できます。

クラスターの作成プロセスをできるだけ簡単にするために、コンソールには、以下で説明する多くの選択肢に対するデフォルトの選択肢があります。コンソール内のほとんどのセクションには、詳細なコンテキストを提供するヘルプパネルもあります。

以下のオプションを変更できます。

- クラスターに名前空間を追加します。

  名前空間を使用すると、クラスターで作成したサービスを、追加の設定なしで名前空間内の他のサービスに接続できます。詳細については、「Amazon ECS サービスを相互接続する」を参照してください。
- クラスターを外部インスタンス用に設定します。
- マネージドストレージに AWS KMS キーを割り当てます。キーの作成方法の詳細については、AWS Key Management Service ユーザーガイドの「KMS キーを作成する」を参照してください。
- クラスターを識別しやすいようにタグを追加します。

新しいクラスターを作成するには (Amazon ECS コンソール)

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションバーから、使用するリージョンを選択します。
3. ナビゲーションペインで [Clusters] (クラスター) を選択します。
4. [Clusters] (クラスター) ページで、[Create Cluster] (クラスターの作成) を選択します。
5. [クラスター設定] で以下を設定します。
   - [クラスター名] に一意の名前を入力します。

     名前には、最大 255 文字 (大文字と小文字)、数字、およびハイフンを含めることができます。
   - (オプション) Service Connect に使用する名前空間をクラスター名と別のものにするには、[名前空間] に一意の名前を入力します。
6. (オプション) Container Insights を使用して モニタリング を展開し、次のいずれかのオプションを選択します。
   - オブザーバビリティが強化された推奨の Container Insights を使用するには、[オブザーバビリティが強化された Container Insights] を選択します。
   - Container Insights を使用するには、[Container Insights] を選択します。
7. (オプション) ECS Exec を使用してクラスター内のタスクをデバッグするには、[トラブルシューティング設定] を展開し、以下を設定します。
   - [ECS Exec をオンにする] を選択します。

- (オプション) [ECS Exec の AWS KMS キー] には、ECS Exec セッションデータの暗号化に使用する AWS KMS キーの ARN を入力します。
- (オプション) [ECS Exec ログ記録] で、ログの送信先を選択します。
  - CloudWatch Logs にログを送信するには、[Amazon CloudWatch] を選択します。
  - Amazon S3 にログを送信するには、[Amazon S3] を選択します。
  - ログ記録を無効にするには、[なし] を選択します。

8. (オプション) マネージドストレージのデータを暗号化します。[暗号化] の [マネージドストレージ] に、マネージドストレージデータの暗号化に使用する AWS KMS キーの ARN を入力します。
9. (オプション) クラスターを識別しやすくするには、[Tags] (タグ) を展開し、タグを設定します。

   [タグの追加] [タグの追加] を選択して、以下を実行します。

   - [キー] にはキー名を入力します。
   - [値] にキー値を入力します。
10. [作成] を選択します。

次のステップ

インスタンスをクラスターに登録する必要があります。詳細については、「Amazon ECS クラスターに外部インスタンスを登録する」を参照してください。

外部起動タイプのタスク定義を作成します。詳細については、コンソールを使用した Amazon ECS タスク定義の作成を参照してください。

スタンドアロンタスクとして、またはサービスの一部としてアプリケーションを実行します。詳細については次を参照してください:

- Amazon ECS タスクとしてのアプリケーションの実行
- Amazon ECS のローリング更新デプロイの作成

Amazon ECS クラスターに外部インスタンスを登録する

Amazon ECS クラスターに登録する外部インスタンスごとに、SSM Agent、Amazon ECS コンテナエージェント、および Docker がインストールされている必要があります。外部インスタンスを Amazon ECS クラスターに登録するには、最初に AWS Systems Manager マネージドインスタンス

を登録する必要があります。インストールスクリプトは、Amazon ECS コンソールで数回クリックするだけで作成できます。インストールスクリプトには、Systems Manager のアクティベーションキーと、必要なエージェントと Docker をインストールするためのコマンドが含まれています。インストールと登録の手順を完了するには、オンプレミスのサーバーまたは VM でインストールスクリプトを実行する必要があります。

> **Note**
>
> Linux の外部インスタンスをクラスターに登録する前に、`/etc/ecs/ecs.config` ファイルを作成し、必要なコンテナエージェント設定パラメータを追加します。外部インスタンスをクラスターに登録した後は、これを行うことはできません。詳細については、「Amazon ECS コンテナエージェントの設定」を参照してください。

AWS マネジメントコンソール

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションバーから、使用するリージョンを選択します。
3. ナビゲーションペインで [Clusters] (クラスター) を選択します。
4. リポジトリの [クラスター] ページで、外部インスタンスを登録するクラスターを選択します。
5. [Cluster : ***name***] (クラスター: 名前) のページで、[Infrastructure] (インフラストラクチャ) タブを選択します。
6. [Register external instances] (外部インスタンスの登録) ページで、次のステップを完了します。

    a. [Activation key duration (in days)] (アクティベーションキーの期間 (日数)) を使用する場合、アクティベーションキーがアクティブなままになる日数を入力します。入力した日数が経過すると、外部インスタンスの登録時にキーが機能しなくなります。

    b. インスタンス数を使用する場合　アクティベーションキーを使用してクラスターに登録する外部インスタンスの数を入力します。

    c. インスタンスロールを使用する場合、外部インスタンスに関連付ける IAM ロールを選択します。ロールがまだ作成されていない場合は、新規ロールの作成を選択すると、Amazon ECS がユーザーに代わってロールを作成します。外部インスタンスに必要な IAM 許可の詳細については、「Amazon ECS Anywhere IAM ロール」を参照してください。

d. 登録コマンドをコピーします。このコマンドは、クラスターに登録する各外部インスタンスで実行する必要があります。

> **Important**
>
> スクリプトの bash 部分は root として実行する必要があります。コマンドが root として実行されない場合、エラーが返されます。

e. [閉じる] を選択してください。

AWS CLI for Linux operating systems

1. Systems Manager のアクティベーションペアを作成します。これは、Systems Manager が管理するインスタンスのアクティベーションに使用されます。出力には、`ActivationId`および`ActivationCode`が含まれます。これらは、後のステップで使用します。作成した ECS Anywhere IAM ロールを指定していることを確認します。詳細については、「Amazon ECS Anywhere IAM ロール」を参照してください。

```
aws ssm create-activation --iam-role ecsAnywhereRole | tee ssm-activation.json
```

2. オンプレミスのサーバーまたは仮想マシン (VM) で、インストールスクリプトをダウンロードします。

```
curl --proto "https" -o "/tmp/ecs-anywhere-install.sh" "https://amazon-ecs-agent.s3.amazonaws.com/ecs-anywhere-install-latest.sh"
```

3. (オプション) オンプレミスサーバーまたは仮想マシン (VM) で、次の手順を使用して、スクリプト署名ファイルを使用してインストールスクリプトを確認します。

   a. GnuPG をダウンロードし、インストールします。GNUpg の詳細については、「GnuPG ウェブサイト」を参照してください。Linux システムの場合は、お使いの Linux ディストリビューションでパッケージマネージャーを使用して gpg をインストールします。

   b. Amazon ECS PGP パブリックキーを取得します。

```
gpg --keyserver hkp://keys.gnupg.net:80 --recv BCE9D9A42D51784F
```

c. インストールスクリプトの署名をダウンロードします。署名は、ASCII でデタッチ済みの PGP 署名で、拡張子が .asc のファイルに保存されています。

```
curl --proto "https" -o "/tmp/ecs-anywhere-install.sh.asc" "https://amazon-ecs-agent.s3.amazonaws.com/ecs-anywhere-install-latest.sh.asc"
```

d. キーを使用してインストールスクリプトファイルを確認します。

```
gpg --verify /tmp/ecs-anywhere-install.sh.asc /tmp/ecs-anywhere-install.sh
```

予想される出力は次のようになります。

```
gpg: Signature made Tue 25 May 2021 07:16:29 PM UTC
gpg:                using RSA key 50DECCC4710E61AF
gpg: Good signature from "Amazon ECS <ecs-security@amazon.com>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the
 owner.
Primary key fingerprint: F34C 3DDA E729 26B0 79BE  AEC6 BCE9 D9A4 2D51 784F
     Subkey fingerprint: D64B B6F9 0CF3 77E9 B5FB  346F 50DE CCC4 710E 61AF
```

4. オンプレミスのサーバーまたは仮想マシン (VM) で、インストールスクリプトを実行します。クラスター名、リージョン、Systems Manager のアクティベーション ID とアクティベーションコードを、最初のステップで指定します。

```
sudo bash /tmp/ecs-anywhere-install.sh \
    --region $REGION \
    --cluster $CLUSTER_NAME \
    --activation-id $ACTIVATION_ID \
    --activation-code $ACTIVATION_CODE
```

オンプレミスサーバーまたは仮想マシン (VM) で、GPU ワークロード用の NVIDIA ドライバーがインストールされている場合は、インストールスクリプトに --enable-gpu フラグを設定する必要があります。このフラグを指定すると、インストールスクリプトは NVIDIA ドライバが実行中であることを確認してから、Amazon ECS タスクを実行するために必要な構成変数を追加します。GPU ワークロードの実行と、タスク定義での GPU 要件の指定の詳細については、「Amazon ECS タスク定義での GPU の指定」を参照してください。

```
sudo bash /tmp/ecs-anywhere-install.sh \
    --region $REGION \
```

```
    --cluster $CLUSTER_NAME \
    --activation-id $ACTIVATION_ID \
    --activation-code $ACTIVATION_CODE \
    --enable-gpu
```

既存の外部インスタンスを別のクラスターに登録するには、次のステップを実行します。

既存の外部インスタンスを別のクラスターに登録するには

1. Amazon ECS コンテナエージェントを停止します。

```
sudo systemctl stop ecs.service
```

2. /etc/ecs/ecs.configファイルとECS_CLUSTER行を編集して、クラスター名が外部インスタンスを登録するクラスターの名前と一致していることを確認します。
3. 既存の Amazon ECS エージェントデータを削除します。

```
sudo rm /var/lib/ecs/data/agent.db
```

4. Amazon ECS コンテナエージェントを開始する

```
sudo systemctl start ecs.service
```

AWS CLI for Windows operating systems

1. Systems Manager のアクティベーションペアを作成します。これは、Systems Manager が管理するインスタンスのアクティベーションに使用されます。出力には、ActivationIdおよびActivationCodeが含まれます。これらは、後のステップで使用します。作成した ECS Anywhere IAM ロールを指定していることを確認します。詳細については、「Amazon ECS Anywhere IAM ロール」を参照してください。

```
aws ssm create-activation --iam-role ecsAnywhereRole | tee ssm-activation.json
```

2. オンプレミスのサーバーまたは仮想マシン (VM) で、インストールスクリプトをダウンロードします。

```
Invoke-RestMethod -URI "https://amazon-ecs-agent.s3.amazonaws.com/ecs-anywhere-
install.ps1" -OutFile “ecs-anywhere-install.ps1”
```

3. (オプション) Powershell スクリプトは Amazon によって署名されているため、Windows は同じ場所で証明書の検証を自動的に実行します。手動で検証を実行する必要はありません。

   証明書を手動で検証するには、ファイルを右クリックしてプロパティに移動し、[Digital Signatures] (デジタル署名) タブを使用して詳細を取得します。

   このオプションは、ホストが証明書ストアに証明書がある場合にのみ使用できます。

   検証の結果、次のような情報が返されます。

   ```
   # Verification (PowerShell)
   Get-AuthenticodeSignature -FilePath .\ecs-anywhere-install.ps1

   SignerCertificate                         Status      Path
   -----------------                         ------      ----
   EXAMPLECERTIFICATE                        Valid       ecs-anywhere-install.ps1

   ...

   Subject              : CN="Amazon Web Services, Inc.",...

   ----
   ```

4. オンプレミスのサーバーまたは仮想マシン (VM) で、インストールスクリプトを実行します。クラスター名、リージョン、Systems Manager のアクティベーション ID とアクティベーションコードを、最初のステップで指定します。

   ```
   .\ecs-anywhere-install.ps1 -Region $Region -Cluster $Cluster -
   ActivationID $ActivationID -ActivationCode $ActivationCode
   ```

5. Amazon ECS コンテナエージェントが実行されていることを確認します。

   ```
   Get-Service AmazonECS

   Status   Name               DisplayName
   ------   ----               -----------
   Running  AmazonECS          Amazon ECS
   ```

既存の外部インスタンスを別のクラスターに登録するには、次のステップを実行します。

既存の外部インスタンスを別のクラスターに登録するには

1. Amazon ECS コンテナエージェントを停止します。

```
Stop-Service AmazonECS
```

2. クラスター名が外部インスタンスを登録するクラスターの名前と一致するように ECS_CLUSTER パラメータを変更します。

```
[Environment]::SetEnvironmentVariable("ECS_CLUSTER", $ECSCluster,
 [System.EnvironmentVariableTarget]::Machine)
```

3. 既存の Amazon ECS エージェントデータを削除します。

```
Remove-Item -Recurse -Force $env:ProgramData\Amazon\ECS\data\*
```

4. Amazon ECS コンテナエージェントを開始する

```
Start-Service AmazonECS
```

AWS CLIを使用して、外部インスタンスの登録プロセスを完了するためにインストールスクリプトを実行する前に、Systems Manager のアクティベーションを作成できます

Amazon ECS 外部インスタンスの登録を解除する

インスタンスの使用が終了したら、Amazon ECS とAWS Systems Manager をインスタンスから登録解除することをお勧めします。登録解除後、コンテナインスタンスは新しいタスクを受けることができなくなります。

登録解除するときにコンテナインスタンスでタスクが実行されている場合、インスタンスを削除するかタスクが他の手段で停止するまで、これらのタスクは実行されたままになります。ただし、これらのタスクは Amazon ECS によるモニタリングや情報収集の対象外になります。外部インスタンス上のこれらのタスクが Amazon ECS サービスに含まれる場合、サービススケジューラは、可能であれば、別のコンテナインスタンスでそのタスクの別のコピーを開始します。

インスタンスの登録を解除した後、インスタンス上の残りの AWS リソースをクリーンアップします。その後、新しいクラスターに登録できます。

手順

AWS マネジメントコンソール

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションバーから、外部インスタンスが存在するリージョンを選択します。
3. ナビゲーションペインで [Clusters] (クラスター) を選択し、外部インスタンスをホストするクラスターを選択します。
4. [Cluster : ***name***] (クラスター: 名前) のページで、[Infrastructure] (インフラストラクチャ) タブを選択します。
5. [Container instances] (コンテナインスタンス)で、登録解除する外部のインスタンス ID を選択します。コンテナインスタンスの詳細ページにリダイレクトされます。
6. [Container Instance : ***id***] (コンテナインスタンス: id) ページで、[Deregister] (登録解除) を選択します。
7. 登録解除メッセージを確認します。[Deregistered fromAWS Systems Manager]を選択して、外部インスタンスを Systems Manager 管理対象インスタンスとして登録解除することもできます。[Deregister] (登録解除) を選択します。

   > **Note**
   >
   > Systems Manager コンソールで、外部インスタンスを Systems Manager 管理対象インスタンスとして登録解除できます。手順については、「AWS Systems Manager ユーザーガイド」の「ハイブリッドおよびマルチクラウド環境でのマネージドノードの登録解除」を参照してください。

8. インスタンスの登録を解除した後は、オンプレミスサーバーまたは VM にある AWS リソースをクリーンアップします。

オペレーティングシステム	Steps	
Linux	a. インスタンスの Amazon ECS コンテナエージェントと SSM Agent サービスを停止します。	

オペレーティングシステム	Steps

	`sudo systemctl stop ecs amazon-ssm-agent`
	b. Amazon ECS パッケージおよびSystems Manager パッケージを削除します。
	CentOS 7、CentOS 8、および RHEL 7 の場合
	`sudo yum remove -y amazon-ecs-init amazon-ssm-agent`
	SUSE Linux Enterprise Server 15 の場合
	`sudo zypper remove -y amazon-ecs-init amazon-ssm-agent`
	Ubuntu と Debian の場合
	`sudo apt remove -y amazon-ecs-init amazon-ssm-agent`
	c. 残ったディレクトリを削除します。
	`sudo rm -rf /var/lib/ecs /etc/ecs /var/lib/amazon/ss`

オペレーティングシステム	Steps
	`m /var/log/ecs / var/log/amazon/ssm`

Windows	a. インスタンスの Amazon ECS コンテナエージェントと SSM Agent サービスを停止します。
	`Stop-Service AmazonECS`
	`Stop-Service AmazonSSMAgent`
	b. Amazon ECS パッケージを削除します。
	`.\ecs-anywhere-install.ps1 -Uninstall`

AWS CLI

1. コンテナインスタンスを登録解除するには、インスタンス ID とコンテナインスタンス ARN が必要です。これらの値がない場合は、次のコマンドを実行してください

   次のコマンドを実行して、インスタンス ID を取得します。

   インスタンス ID (`instanceID`) を使用し、コンテナインスタンス ARN (`containerInstanceARN`) を取得します。

   ```
   instanceId=$(aws ssm describe-instance-information --region "{{ region }}" |
    jq ".InstanceInformationList[] |select(.IPAddress==\"{{ IPv4 Address }}\")
    | .InstanceId" | tr -d'"'
   ```

   以下のコマンドを実行します。

インスタンス (deregister-container-instance) を登録解除するには、コマンドのパラメータとして containerInstanceArn を使用します。

```
instances=$(aws ecs list-container-instances --cluster "{{ cluster }}" --region
 "{{ region }}" | jq -c '.containerInstanceArns')
containerInstanceArn=$(aws ecs describe-container-instances --cluster
 "{{ cluster }}" --region "{{ region }}" --container-instances $instances
 | jq ".containerInstances[] | select(.ec2InstanceId==\"{{ instanceId }}\")
 | .containerInstanceArn" | tr -d '"')
```

2. 次のコマンドを実行して、インスタンスをドレインします。

```
aws ecs update-container-instances-state --cluster "{{ cluster }}" --region
 "{{ region }}" --container-instances "{{ containerInstanceArn }}" --status
 DRAINING
```

3. コンテナインスタンスのドレインが終了したら、次のコマンドを実行してインスタンスを登録解除します。

```
aws ecs deregister-container-instance --cluster "{{ cluster }}" --region
 "{{ region }}" --container-instance "{{ containerInstanceArn }}"
```

4. 次のコマンドを実行し、SSM からコンテナインスタンスを削除します。

```
aws ssm deregister-managed-instance --region "{{ region }}" --instance-id
 "{{ instanceId }}"
```

5. インスタンスの登録を解除した後は、オンプレミスサーバーまたは VM にある AWS リソースをクリーンアップします。

オペレーティングシステム	Steps	
Linux	a. インスタンスの Amazon ECS コンテナエージェントと SSM Agent サービスを停止します。	

オペレーティングシステム	Steps	

	`sudo systemctl stop ecs amazon-ssm-agent`
	b. Amazon ECS パッケージおよびSystems Manager パッケージを削除します。
	`sudo (yum/apt/zypper) remove amazon-ecs-init amazon-ssm-agent`
	c. 残ったディレクトリを削除します。
	`sudo rm -rf /var/lib/ecs /etc/ecs /var/lib/amazon/ssm /var/log/ecs /var/log/amazon/ssm`	

オペレーティングシステム	Steps	

Windows	a. インスタンスの Amazon ECS コンテナエージェントと SSM Agent サービスを停止します。
	`Stop-Service AmazonECS`
	`Stop-Service AmazonSSMAgent`
	b. Amazon ECS パッケージを削除します。
	`.\ecs-anywhere-install.ps1 -Uninstall`	

外部インスタンス上の AWS Systems Manager エージェントと Amazon ECS コンテナエージェントを更新する

オンプレミスのサーバーまたは VM で、AWS Systems Managerエージェント（SSM Agent）と Amazon ECS コンテナエージェント（Amazon ECS ワークロードの実行時）AWSは、機能が追加または更新されたときに、これらのエージェントの新しいバージョンをリリースします。外部インスタンスがいずれかのエージェントの以前のバージョンを使用している場合は、次の手順を使用して更新できます。

外部インスタンスでの SSM Agent の更新

AWS Systems Managerインスタンスで SSM Agent を更新するプロセスを自動化することをお勧めします。これらは、更新を自動化するためのいくつかの方法を提供します。詳細については、AWS Systems Managerユーザーガイドの「SSM Agent の更新を自動化する」を参照してください。

外部インスタンス上の Amazon ECS エージェントを更新しています

外部インスタンスでは、Amazon ECS コンテナエージェントはecs-initパッケージをアップグレードすることで更新されます。Amazon ECS エージェントを更新しても、実行中のタスクやサービスが中断されることはありません。Amazon ECS がecs-initパッケージと署名ファイルを、各リージョンの Amazon S3 バケットに作成します。ecs-initから始まるバージョン1.52.1-1では、Amazon ECS は、外部インスタンスが使用するオペレーティングシステムとシステムアーキテクチャに応じて個別の ecs-init パッケージを提供します。

以下の表を使用して、ecs-initパッケージは、外部インスタンスが使用するオペレーティングシステムとシステムアーキテクチャに基づいてダウンロードする必要があります。

> **Note**
>
> 次のコマンドを使用して、外部インスタンスが使用するオペレーティングシステムおよびシステムアーキテクチャを決定できます。
>
> ```
> cat /etc/os-release
> uname -m
> ```

オペレーティングシステム (アーキテクチャ)	ecs-init パッケージ
CentOS 7 (x86-64) CentOS 8 (x86_64) CentOS Stream 9 (x86_64) SUSE Enterprise Server 15 (x86_64) RHEL 7 (x86_64) RHEL 8 (x86_64)	amazon-ecs-init-latest.x86_64.rpm
CentOS 7 (aarch64) CentOS 8 (aarch64) CentOS Stream 9 (aarch64)	amazon-ecs-init-latest.aarch64.rpm

オペレーティングシステム (アーキテクチャ)	ecs-init パッケージ
RHEL 7 (aarch64)	
Debian 9 (x86_64) Debian 10 (x86_64) Debian 11 (x86_64) Debian 12 (x86_64) Ubuntu 18 (x86_64) Ubuntu 20 (x86_64) Ubuntu 22 (x86_64) Ubuntu 24 (x86_64)	`amazon-ecs-init-latest.amd64.deb`
Debian 9 (aarch64) Debian 10 (aarch64) Debian 11 (aarch64) Debian 12 (aarch64) Ubuntu 18 (aarch64) Ubuntu 20 (aarch64) Ubuntu 22 (aarch64) Ubuntu 24 (aarch64)	`amazon-ecs-init-latest.arm64.deb`

Amazon ECS エージェントを更新するには、以下の手順に従います。

Amazon ECS エージェントを更新するには

1. 実行している Amazon ECS エージェントのバージョンを確認します。

```
curl -s 127.0.0.1:51678/v1/metadata | python3 -mjson.tool
```

2. オペレーティングシステムとシステムアーキテクチャー用のecs-initパッケージをダウンロードします。Amazon ECS がecs-initパッケージファイルを、各リージョンの Amazon S3 バケットに作成します。置き換えることを確認します。*<region>*コマンド内の識別子を、地理的に最も近いリージョン名 (例:us-west-2) に置き換えることを確認します。

   amazon-ecs-init-latest.x86_64.rpm

   ```
   curl -o amazon-ecs-init.rpm https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.x86_64.rpm
   ```

   amazon-ecs-init-latest.aarch64.rpm

   ```
   curl -o amazon-ecs-init.rpm https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.aarch64.rpm
   ```

   amazon-ecs-init-latest.amd64.deb

   ```
   curl -o amazon-ecs-init.deb https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.amd64.deb
   ```

   amazon-ecs-init-latest.arm64.deb

   ```
   curl -o amazon-ecs-init.deb https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.arm64.deb
   ```

3. (オプション) PGP 署名を使用して、ecs-initパッケージファイルの妥当性を検証します。

   a. GnuPG をダウンロードし、インストールします。GNUpg の詳細については、「GnuPG ウェブサイト」を参照してください。Linux システムの場合は、お使いの Linux ディストリビューションでパッケージマネージャーを使用して gpg をインストールします。

   b. Amazon ECS PGP パブリックキーを取得します。

      ```
      gpg --keyserver hkp://keys.gnupg.net:80 --recv BCE9D9A42D51784F
      ```

   c. ecs-init パッケージ署名ファイルをダウンロードします。署名は、ASCII でデタッチ済みの PGP 署名で、拡張子が .asc のファイルに保存されています。Amazon ECS は、各リー

ジョンの Amazon S3 バケットに署名ファイルを提供します。置き換えることを確認します。*<region>*コマンド内の識別子を、地理的に最も近いリージョン名 (例:us-west-2) に置き換えることを確認します。

amazon-ecs-init-latest.x86_64.rpm

```
curl -o amazon-ecs-init.rpm.asc https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.x86_64.rpm.asc
```

amazon-ecs-init-latest.aarch64.rpm

```
curl -o amazon-ecs-init.rpm.asc https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.aarch64.rpm.asc
```

amazon-ecs-init-latest.amd64.deb

```
curl -o amazon-ecs-init.deb.asc https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.amd64.deb.asc
```

amazon-ecs-init-latest.arm64.deb

```
curl -o amazon-ecs-init.deb.asc https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.arm64.deb.asc
```

d. キーを使用してecs-initパッケージファイルを検証します。

rpmパッケージ向け

```
gpg --verify amazon-ecs-init.rpm.asc ./amazon-ecs-init.rpm
```

debパッケージ向け

```
gpg --verify amazon-ecs-init.deb.asc ./amazon-ecs-init.deb
```

予想される出力は次のようになります。

```
gpg: Signature made Fri 14 May 2021 09:31:36 PM UTC
gpg:                using RSA key 50DECCC4710E61AF
gpg: Good signature from "Amazon ECS <ecs-security@amazon.com>" [unknown]
```

```
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: F34C 3DDA E729 26B0 79BE  AEC6 BCE9 D9A4 2D51 784F
     Subkey fingerprint: D64B B6F9 0CF3 77E9 B5FB  346F 50DE CCC4 710E 61AF
```

4. ecs-init パッケージをインストールします。

   rpmパッケージ向けは、CentOS 7、CentOS 8、および RHEL 7 に搭載されています。

   ```
   sudo yum install -y ./amazon-ecs-init.rpm
   ```

   SUSE Enterprise Server 15 の**rpm**パッケージ向け

   ```
   sudo zypper install -y --allow-unsigned-rpm ./amazon-ecs-init.rpm
   ```

   debパッケージ向け

   ```
   sudo dpkg -i ./amazon-ecs-init.deb
   ```

5. ecs サービスを再起動します。

   ```
   sudo systemctl restart ecs
   ```

6. Amazon ECS エージェントのバージョンが更新されたことを確認します。

   ```
   curl -s 127.0.0.1:51678/v1/metadata | python3 -mjson.tool
   ```

Amazon ECS クラスターを更新する

次のクラスタープロパティを変更できます。

- デフォルトのキャパシティプロバイダーを設定する

  各クラスターには、1 つ以上のキャパシティプロバイダーがあり、さらにオプションとしてキャパシティプロバイダー戦略があります。キャパシティープロバイダー戦略は、クラスターの複数のキャパシティープロバイダー間にタスクを分散する方法を決定します。スタンドアロンタスクを実行するか、サービスを作成するときは、クラスターのデフォルトのキャパシティプロバイダー戦略か、クラスターのデフォルト戦略をオーバーライドするキャパシティプロバイダー戦略のいずれかを使用します。

- Container Insights をオンにします。

  CloudWatch Container Insights は、コンテナ化されたアプリケーションとマイクロサービスのメトリクスとログを収集、集約、要約します。Container Insights は、問題の迅速な特定と解決に使用するコンテナの再起動失敗などの診断情報も提供します。詳細については、「the section called "オブザーバビリティが強化された Container Insights を使用し、Amazon ECS コンテナを監視する"」を参照してください。
- クラスターを識別しやすいようにタグを追加します。

手順

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。
3. [Clusters] (クラスター) ページで、クラスターを選択します。
4. [クラスター: ***name***] ページで、[クラスターを更新] を選択します。
5. デフォルトのキャパシティプロバイダーを設定するには、[デフォルトのキャパシティプロバイダー戦略] で [さらに追加] を選択します。

   a. [キャパシティプロバイダー] で、キャパシティプロバイダーを選択します。
   b. (オプション) [ベース] で、キャパシティプロバイダーで実行されるタスクの最小数を入力します。

      1 つのキャパシティプロバイダーには、1 つの [ベース] 値のみを設定できます。
   c. (オプション) [重み] で、指定されたキャパシティプロバイダーを使用する、起動されたタスクの合計数の相対的な割合を入力します。
   d. (オプション) 追加のキャパシティプロバイダーについて、ステップを繰り返します。
6. Container Insights をオンまたはオフにするには、[モニタリング] を展開し、[Container Insights を使用] をオンにします。
7. クラスターを識別しやすくするには、[タグ] を展開し、タグを設定します。

   [タグの追加] [タグの追加] を選択して、以下を実行します。

   - [キー] にはキー名を入力します。
   - [値] にキー値を入力します。

   [タグを削除] タグのキーと値の右側にある [削除] を選択します。

8. [更新] を選択します。

Amazon ECS クラスターを削除する

クラスターの使用を終了する場合は、クラスターを削除できます。クラスターを削除すると、INACTIVE 状態に移行します。INACTIVE ステータスのクラスターは、一定期間アカウント内で検出可能なままになる場合があります。ただし、この動作は今後変更される可能性があるため、INACTIVE クラスターの永続化に頼るべきではありません。

クラスターを削除する前に、次のオペレーションを実行する必要があります。

- クラスター内のすべてのサービスを削除します。詳細については、「the section called “サービスの削除”」を参照してください。
- 現在実行中のタスクをすべて停止します。詳細については、「the section called “タスクの停止”」を参照してください。
- クラスターに登録されているすべてのコンテナインスタンスの登録を解除します。詳細については、「the section called “コンテナインスタンスの登録解除”」を参照してください。
- 名前空間を削除します。詳細については、AWS Cloud Map デベロッパーガイドの名前空間の削除を参照してください。

手順

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションバーから、使用するリージョンを選択します。
3. ナビゲーションペインで [Clusters] (クラスター) を選択します。
4. [Clusters] (クラスター) ページで、削除するクラスターを選択します。
5. [Delete Cluster] を選択します。

   クラスターに関連付けられているすべてのリソースを削除しなかった場合は、メッセージが表示されます。
6. 確認ボックスに [delete ***cluster name***] (削除 クラスター名) と入力してください。

Amazon ECS コンテナインスタンスの登録を解除する

> **Important**
>
> このトピックは、Amazon EC2 で作成されたコンテナインスタンスのみを対象としています。外部インスタンスの登録を解除する方法については、Amazon ECS 外部インスタンスの登録を解除する を参照してください。

Amazon EC2 でバックアップされたコンテナインスタンスの使用を終了する場合は、そのインスタンスをクラスターから登録解除できます。登録解除後、コンテナインスタンスは新しいタスクを受けることができなくなります。

登録解除するときにコンテナインスタンスでタスクが実行されている場合、インスタンスを削除するかタスクが他の手段で停止するまで、これらのタスクは実行されたままになります。ただし、これらのタスクは孤立しています (Amazon ECS によるモニタリングや情報収集の対象外になります)。コンテナインスタンス上の孤立したタスクが Amazon ECS サービスに含まれる場合、サービススケジューラは、可能であれば、別のコンテナインスタンスでそのタスクの別のコピーを開始します。Application Load Balancer ターゲットグループに登録されている、孤立したサービスタスクのコンテナは、すべてその登録が解除されます。これらはロードバランサーまたはターゲットグループの設定に従って Connection Draining が開始されます。awsvpc ネットワークモードを使用している孤立したタスク、その Elastic Network Interface は削除されます。

登録解除後に、コンテナインスタンスを別の用途に使用する予定の場合は、登録解除前に、コンテナインスタンスで実行中のすべてのタスクを停止する必要があります。これにより、孤立したタスクによってリソースが消費されなくなります。

コンテナインスタンスの登録を解除するときは、以下の考慮事項に注意してください。

- 各コンテナインスタンスには、それぞれに固有の状態情報があるため、1 つのクラスターから登録解除して別のクラスターに再登録しないでください。コンテナインスタンスリソースを再配置するには、1 つのクラスターからコンテナインスタンスを終了し、新しいクラスターで新しいコンテナインスタンスを起動することをお勧めします。詳細については、「Amazon EC2 ユーザーガイド」の「インスタンスの終了」および Amazon ECS Linux コンテナインスタンスの起動 を参照してください。
- コンテナインスタンスが Auto Scaling グループまたはCloudFormationスタックで管理されている場合は、Auto Scaling グループを更新するか、CloudFormationスタックでインスタンスを終了しま

す。それ以外の場合は、Auto Scaling グループまたはCloudFormationにより、インスタンス終了後に新しいインスタンスが作成されます。

- 接続された Amazon ECS コンテナエージェントを使用して実行中のコンテナインスタンスを削除する場合は、エージェントによってクラスターからインスタンスが自動的に登録解除されます。停止したコンテナインスタンス、または切断されたエージェントがあるインスタンスは、終了時に自動的に登録解除されません。
- コンテナインスタンスを登録解除すると、インスタンスがクラスターから削除されますが、Amazon EC2 インスタンスは終了しません。インスタンスの使用を終了する場合は、必ずインスタンスを終了して課金を停止します。詳細については、「Amazon EC2 ユーザーガイド」の「インスタンスの終了」を参照してください。

手順

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションバーから、外部インスタンスが存在するリージョンを選択します。
3. ナビゲーションペインで [Clusters] (クラスター) を選択し、インスタンスをホストするクラスターを選択します。
4. [Cluster : ***name***] (クラスター: 名前) のページで、[Infrastructure] (インフラストラクチャ) タブを選択します。
5. [Container instances] (コンテナインスタンス)で、登録解除するインスタンス ID を選択します。コンテナインスタンスの詳細ページにリダイレクトされます。
6. [Container Instance : ***id***] (コンテナインスタンス: id) ページで、[Deregister] (登録解除) を選択します。
7. 確認画面で [登録解除] を選択します。
8. コンテナインスタンスの使用を終了する場合は、基になる Amazon EC2 インスタンスを終了します。詳細については、「Amazon EC2 ユーザーガイド」の「インスタンスの終了」を参照してください。

Amazon ECS コンテナインスタンスをドレインする

クラスターからコンテナインスタンスを削除する必要がある場合があります。例えば、システム更新を実行したり、クラスターキャパシティーをスケールダウンしたりする場合などです。Amazon ECS では、コンテナインスタンスをDRAINING ステータスに遷移する能力を提供します。これ

は、コンテナインスタンスのドレインと呼ばれます。コンテナインスタンスを DRAINING に設定すると、Amazon ECS によって新規タスクがそのコンテナインスタンスに配置されなくなります。

サービスのドレイニング動作

PENDING 状態にあるサービスの一部であるタスクは、直ちに停止されます。クラスター内に利用可能なコンテナインスタンス容量がある場合、サービススケジューラによって置き換えタスクが開始されます。十分なコンテナインスタンス容量がない場合、問題を示すサービスイベントメッセージが送信されます。

RUNNING 状態にあるコンテナインスタンス上のサービスの一部であるタスクは、STOPPED 状態に移行します。サービススケジューラは、サービスのデプロイタイプ、デプロイ設定パラメータ、minimumHealthyPercent および maximumPercent に従って、タスクを置き換えようとします。詳細については、「Amazon ECS サービス」および「Amazon ECS サービス定義パラメータ」を参照してください。

- minimumHealthyPercent が 100% を下回っている場合、タスクの代替中、スケジューラは一時的に desiredCount を無視できます。例えば、desiredCount が 4 つのタスクの場合、最小値 50% でスケジューラは 2 つの既存タスクを停止してから 2 つの新規タスクを開始できます。最小が 100% の場合、サービススケジューラは、代替タスクが正常な状態と見なされるまで既存タスクを削除できません。ロードバランサーを使用しないサービスのタスクが RUNNING 状態にある場合、正常な状態と見なされます。ロードバランサーを使用するサービスのタスクは、RUNNING 状態にあり、そのタスクをホストするコンテナインスタンスがロードバランサーによって正常と報告された場合に、正常であると見なされます。

  > **Important**
  >
  > スポットインスタンスを使用していて、minimumHealthyPercent が 100% 以上の場合、サービスには、スポットインスタンスが終了する前にタスクを置き換えるための十分な時間がありません。

- maximumPercent パラメータは、タスクの置き換え中に実行できるタスク数の上限を表します。これは、置き換えのバッチサイズを定義するために使用できます。例えば、desiredCount が 4 つのタスクで、最大が 200% であればドレインされる 4 つのタスクを停止する前に 4 つの新規タスクを開始できます (これを行うために必要なクラスターリソースを使用できる場合)。最大が 100% の場合、代替タスクは、ドレインするタスクが停止するまで開始できません。

> **Important**
>
> `minimumHealthyPercent` と `maximumPercent` の両方が 100% の場合、サービスは既存のタスクを削除できず、代替タスクを開始することもできません。これにより、コンテナインスタンスのドレインの成功を防止し、新たなデプロイが防止されます。

スタンドアロンタスクのドレイニング動作

`PENDING` または `RUNNING` 状態のスタンドアロンタスクは影響を受けません。自分で停止するか、手動で停止するまで待つ必要があります。コンテナインスタンスは `DRAINING` ステータスのままです。

Amazon ECS マネージドインスタンスのドレイニング動作

Amazon ECS マネージドインスタンスの終了は、コストを最適化し、システムの健全性を維持しながら、ワークロードを正常に移行します。終了システムには、インスタンスの終了に関する 3 つの異なる決定パスがあり、それぞれのタイミングの特性やお客様に影響する事項が異なります。

お客様が開始した終了

コンテナインスタンスをすぐにサービスから削除する必要がある場合に、インスタンスの削除を直接制御できます。`force` リクエストパラメータを true に設定して、`deregister-container-instance` を実行します。つまり、実行中のワークロードがあっても、即時終了が必要になります。

システムによるアイドル終了

Amazon ECS マネージドインスタンスは、タスクを実行していないアイドル状態の Amazon ECS コンテナインスタンスを終了することで、コストを継続的にモニタリングし、プロアクティブに最適化します。ECS はヒューリスティック遅延を使用して、コンテナインスタンスが終了する前に新しく起動されたタスクを取得する時間を提供します。これは、`scaleInAfter` Amazon ECS マネージドインスタンスのキャパシティープロバイダー設定パラメータを使用してカスタマイズできます。

インフラストラクチャの更新の終了

Amazon ECS マネージドインスタンスでは、ワークロードの可用性を維持しながら、マネージドコンテナインスタンスのソフトウェアを自動的に管理および更新することで、セキュリティとコ

ンプライアンスを確実にします。詳細については、Amazon ECS マネージドインスタンスでのパッチ適用を参照してください。

終了システムは、ワークロードの継続性とインフラストラクチャ管理要件のバランスを取る、2 段階のアプローチを実行します。

フェーズ 1: 正常な完了期間

このフェーズでは、システムはワークロードの継続性を優先する、正常なドレイニング戦略を実行します。サービスタスクは、通常の Amazon ECS スケジューリングプロセスによって正常にドレインされます。スタンドアロンタスクは自然に完了する可能性があるため、実行が継続されます。システムは、自然な完了プロセスを通じて停止状態に達するように、すべてのタスクをモニタリングします。

フェーズ 2: 厳格な期限の適用

正常な完了が、許容可能な期間内に終了目標を達成しない場合、システムは厳格な期限の適用を実行します。厳格な期限では、通常ドレイン開始日時に 7 日を加えて設定されるため、運用要件は維持されますが、正常な完了にはかなりの時間がかかります。強制には、強制登録解除手順の自動実行と、完了ステータスを考慮しない残りのすべてのタスクの即時終了が含まれます。

インスタンスで実行されているすべてのタスクが `STOPPED` 状態に移行すると、コンテナインスタンスのドレインが完了します。コンテナインスタンスは、再びアクティブ化または削除されるまで、`DRAINING` 状態のままです。コンテナインスタンス上のタスクの状態を確認するには、ListTasks オペレーションを `containerInstance` パラメータと共に使用して、インスタンス上のタスクのリストを取得した後、各 Amazon リソースネーム (ARN) または各タスクの ID で DescribeTasks オペレーションを実行して、タスクの状態を確認します。

コンテナインスタンスがタスクのホスティングを再開する準備ができたら、コンテナインスタンスの状態を `DRAINING` から `ACTIVE` に変更します。Amazon ECS サービススケジューラは、コンテナインスタンスを再度検討してタスクを配置します。

手順

次の手順に従って、新しい AWS マネジメントコンソール を使用してコンテナインスタンスをドレインする設定ができます。

また、UpdateContainerInstancesState API アクションまたは update-container-instances-state コマンドを使用して、コンテナインスタンスのステータスを `DRAINING` に変更することも可能です。

AWS マネジメントコンソール

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。
3. [Clusters] (クラスター) ページで、インスタンスをホストするクラスターを選択します。
4. [Cluster : ***name***] (クラスター: 名前) のページで、[Infrastructure] (インフラストラクチャ) タブを選択します。次に、[Container instances] (コンテナインスタンス) タブを選択し、ドレインしたい各コンテナインスタンスのチェックボックスをオンにします。
5. [アクション]、[ドレイン] の順に選択します。

Auto Scaling グループ外のインスタンスのタイプまたはサイズを変更する

AWS では、イミュータブルなインフラストラクチャを維持することをお勧めします。インスタンスサイズを変更する必要がある場合は、次のいずれかを実行できます。

- 水平方向にスケーリングし、インスタンスを追加します。次に、それらのインスタンスに追加のタスクを配置します。または、
- 垂直方向にスケーリングし、新しい大規模/小規模インスタンスを起動して、古いインスタンスをドレインします。

これらのアプローチは、いずれもアプリケーションの可用性への影響を最小限に抑えるのに役立ちます。

別の方法を使用してインスタンスを変更した場合、次のエラーが表示されることがあります。

```
Container instance type changes are not supported.
```

このエラーが表示されたら、次の手順を実行します。

1. 目的のインスタンスタイプで新しいインスタンスを起動します。
2. 古いインスタンスタイプをドレインします。詳細については、「the section called “コンテナインスタンスのドレイン”」を参照してください。

Amazon ECS EC2 コンテナインスタンス

Amazon ECS エージェントは、クラスターに登録されているすべてのコンテナインスタンスで実行されるプロセスです。これにより、コンテナインスタンスと Amazon ECS の間の通信が容易になります。

> **Note**
>
> Linux コンテナインスタンスでは、エージェントコンテナは `/lib`、`/lib64`、`/proc` などの最上位ディレクトリをマウントします。これは、Amazon EBS ボリューム、`awsvpc` ネットワークモード、Amazon ECS Service Connect、FireLens – Amazon ECS といった、ECS の様々な機能を利用するために必要です。

Amazon ECSコンテナエージェントの各バージョンは、異なる機能セットをサポートし、以前のバージョンのバグ修正を提供します。可能であれば、最新バージョンの Amazon ECSコンテナエージェントを使用することを常にお勧めします。コンテナエージェントを最新バージョンに更新するには、「Amazon ECS コンテナエージェントをアップデートする」を参照してください。

Amazon ECS コンテナエージェントには `amazon-ecs-pause` イメージが含まれています。Amazon ECS は、`awsvpc` ネットワークモードを使用するタスクにこのイメージを使用します。

どの機能と拡張が各エージェントリリースに含まれているか確認するには、https://github.com/aws/amazon-ecs-agent/releases を参照してください。

> **Important**
>
> 信頼できるメトリクスの最小 Docker バージョンは、Amazon ECS 最適化 AMI `20220607` 以降に含まれる Docker バージョン `v20.10.13` 以降です。
> バージョン `1.20.0` 以降のAmazon ECSエージェントでは、`18.01.0` より前のバージョンの Docker のサポートが廃止されました。

ライフサイクル

Amazon ECS コンテナエージェントが Amazon EC2 インスタンスをクラスターに登録すると、Amazon EC2 インスタンスのステータスは、`ACTIVE` として、エージェントの接続ステータス

は、TRUE としてレポートされます。このコンテナインスタンスはタスクの実行リクエストを受けることができます。

コンテナインスタンスを (終了ではなく) 停止した場合、ステータスは ACTIVE のままになりますが、エージェント接続ステータスは数分以内に FALSE になります。コンテナインスタンスで実行されていたタスクはすべて停止します。コンテナインスタンスを再び開始すると、コンテナエージェントは、Amazon ECS サービスと再接続し、インスタンスでタスクを再実行できるようになります。

コンテナインスタンスのステータスを DRAINING に変更すると、新しいタスクはそのコンテナインスタンスに配置されません。そのコンテナインスタンスで実行されているサービスタスクは、可能な場合は削除され、システム更新を実行できるようになります。詳細については、「Amazon ECS コンテナインスタンスをドレインする」を参照してください。

コンテナインスタンスを登録解除または終了した場合、コンテナインスタンスのステータスは直ちに INACTIVE に変わり、コンテナインスタンスを一覧表示しても、そのコンテナインスタンスはレポートされなくなります。ただし、終了後 1 時間は、コンテナインスタンスの内容を表示できます。1 時間後、インスタンスの内容は表示できなくなります。

インスタンスを手動でドレインさせるか、Auto Scaling グループのライフサイクルフックを構築して、インスタンスの状態を DRAINING にさせます。Auto Scaling ライフサイクルフックの詳細については、「Amazon EC2 Auto Scaling ライフサイクルフック」を参照してください。

Docker サポート

Amazon ECS は、Amazon Linux で公開された Docker の直近の 2 つの主要バージョンをサポートしています。現在、これには Docker 20.10.x と Docker 25.x が含まれます。

Amazon ECS に必要な最低限の Docker バージョンは、GitHub の Amazon ECS エージェント仕様ファイルにあります。

Amazon ECS に最適化された AMI を使用する場合、Docker は Amazon ECS コンテナエージェントで動作するように事前にインストールおよび設定されています。AMI には、Amazon ECS でテストおよびサポートされている Docker バージョンが含まれています。

> **Note**
>
> Amazon ECS は複数の Docker バージョンをサポートしていますが、最高の互換性とサポートを確保するために、Amazon ECS 最適化 AMI に付属する Docker バージョンを使用することをお勧めします。

Amazon ECS に最適化された AMI

Amazon ECS に最適化された AMI の詳細については、「Amazon ECS に最適化された Linux AMI」を参照してください。

追加情報

次のページでは、変更に関する追加情報を説明します。

- GitHub の Amazon ECS Agent の変更ログ
- Amazon Linux 2 リリースノート。
- Docker ドキュメントの「Docker Engine リリースノート」
- NVIDIA ドキュメントの「NVIDIA ドライバードキュメント」

Amazon ECS コンテナエージェントの設定

適用対象: EC2 インスタンス

Amazon ECS コンテナエージェントでは、多数の設定オプションがサポートされており、そのほとんどは環境変数を介して設定します。

コンテナインスタンスが Amazon ECS に最適化されたAMIの Linux バリアントを使用して起動された場合は、これらの環境変数を `/etc/ecs/ecs.config` ファイルに設定してからエージェントを再び開始できます。起動時に Amazon EC2 ユーザーデータを使用して、コンテナインスタンスにこれらの設定変数を作成することもできます。詳細については、「Amazon ECS Linux コンテナインスタンスをブートストラップしてデータを渡す」を参照してください。

コンテナインスタンスが Amazon ECS に最適化された AMI の Windows バリアントを使用して起動された場合は、PowerShell SetEnvironmentVariable コマンドを使用して、これらの環境変数を設定してからエージェントを再び開始できます。詳細については、「Amazon EC2 ユーザーガイド」の「ユーザーデータ入力を使用して EC2 インスタンスを起動するときにコマンドを実行する」および「the section called “コンテナインスタンスのブートストラップ”」を参照してください。

Amazon ECS コンテナエージェントを手動で開始する場合 (Amazon ECSに最適化されていない AMI の場合)、これらの環境変数は、エージェントの起動に使用する docker run コマンドで使用できます。これらの変数は構文 `--env=`*`VARIABLE_NAME`*`=`*`VARIABLE_VALUE`* で使用します。プライベートリポジトリの認証情報など、機密性の高い情報の場合は、エージェントの環境変数をファイルに保存

し、--env-file *path_to_env_file* オプションを使用して、それらすべてを一度に渡す必要があります。以下のコマンドを使用して変数を追加できます。

```
sudo systemctl stop ecs
sudo vi /etc/ecs/ecs.config
# And add the environment variables with VARIABLE_NAME=VARIABLE_VALUE format.
sudo systemctl start ecs
```

ホスト PID 名前空間を使用した Amazon ECS エージェントの実行

デフォルトで、Amazon ECS エージェントは独自の PID 名前空間で実行されます。以下の設定で、ホスト PID 名前空間で実行するように Amazon ECS エージェントを設定できます。

- SELinux 強制モードが有効になっています。
- Docker の SELinux セキュリティポリシーが true に設定されています。

この動作を設定するには、/etc/ecs/ecs.config ファイルで ECS_AGENT_PID_NAMESPACE_HOST 環境変数を true に設定します。この変数を有効にすると、ecs-init はホストの PID 名前空間 (--pid=host) で Amazon ECS エージェントコンテナを起動します。これにより、エージェントは SELinux 強制環境で適切に自身をブートストラップできます。この機能は、Amazon ECS エージェントバージョン 1.94.0 以降で使用できます。

この機能を有効にするには、/etc/ecs/ecs.config ファイルに次の行を追加します。

```
ECS_AGENT_PID_NAMESPACE_HOST=true
```

この変更を行った後に、Amazon ECS エージェントを再起動して変更を有効にします。

```
sudo systemctl restart ecs
```

以下の機能は、SELinux 強制モードが有効で、Docker セキュリティポリシーが true に設定されている場合でも、ECS_AGENT_PID_NAMESPACE_HOST=true が設定されていても動作しません。

- Amazon ECS Exec
- Amazon EBS タスクアタッチ
- Service Connect
- FireLensAmazon ECS の

使用できるパラメータ

使用可能な Amazon ECS コンテナエージェントの設定パラメータについては、GitHub の「Amazon ECS コンテナエージェント」を参照してください。

Amazon S3 に Amazon ECS コンテナインスタンスの設定を保存する

Amazon ECS コンテナエージェントの設定は、環境変数によって制御されます。Amazon ECS に最適化されたAMIの Linux バリアントは、コンテナエージェントの起動時に `/etc/ecs/ecs.config` でこれらの変数を検索し、見つかった変数に応じてエージェントを設定します。`ECS_CLUSTER` などの機密性の低い環境変数は、Amazon EC2 のユーザーデータを経由して起動時にコンテナインスタンスに渡され、そのままこのファイルに書き込まれます。ただし、AWS 認証情報や `ECS_ENGINE_AUTH_DATA` 変数など機密性の高い情報は、ユーザーデータでインスタンスに渡したり、`/etc/ecs/ecs.config` ファイルに表示される可能性のある方法で `.bash_history` に書き込んだりしないでください。

設定情報を Amazon S3 のプライベートバケットに保存し、コンテナインスタンスの IAM ロールに読み取り専用アクセス権限を付与するのが、コンテナインスタンスの起動時に設定を許可する安全で便利な方法です。`ecs.config` ファイルのコピーはプライベートバケットに保存できます。その後、AWS CLI Amazon EC2 ユーザーデータを使用してをインストールし、インスタンスの起動時に設定情報を `/etc/ecs/ecs.config` にコピーできます。

`ecs.config` Amazon S3 のファイルを保存するには

1. コンテナインスタンスロール (ecsInstanceRole) に、Amazon S3 への読み取り専用アクセス権を持つためのアクセス許可を付与する必要があります。これを行うには、AmazonS3ReadOnlyAccess を `ecsInstanceRole` ロールに割り当てます。ポリシーをロールにアタッチする方法については、「AWS Identity and Access Management ユーザーガイド」の「ロールに対するアクセス許可を更新する」を参照してください。
2. 次の形式を使用して、有効な Amazon ECS エージェントの設定変数を含む `ecs.config` ファイルを作成します。この例では、プライベートレジストリ認証を設定しています。詳細については、「Amazon ECS での AWS 以外のコンテナイメージの使用」を参照してください。

```
ECS_ENGINE_AUTH_TYPE=dockercfg
ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":
{"auth":"zq212MzEXAMPLE7o6T25Dk0i","email":"email@example.com"}}
```

> **Note**
>
> 利用可能な Amazon ECS エージェントの設定変数の完全なリストについては、GitHub の「Amazon ECS コンテナエージェント」を参照してください。

3. 設定ファイルを保存するには、Amazon S3 内にプライベートバケットを作成します。詳細については、「Amazon Simple Storage Service ユーザーガイド」の「バケットの作成」を参照してください。
4. `ecs.config` ファイルを S3 バケットにアップロードします。詳細については、Amazon Simple Storage Service 開発者ガイドの「オブジェクトのアップロード」を参照してください。

起動時に **`ecs.config`** ファイルを Amazon S3 からロードするには

1. このセクションの上記の手順を完了して、読み取り専用 Amazon S3 アクセス権限をコンテナインスタンスに許可し、 `ecs.config` ファイルをプライベート S3 バケットに保存します。
2. 新しいコンテナインスタンスを起動し、EC2 ユーザーデータで次のサンプルスクリプトを使用します。このスクリプトは AWS CLI をインストールし、設定ファイルを `/etc/ecs/ecs.config` にコピーします。詳細については、「Amazon ECS Linux コンテナインスタンスの起動」を参照してください。

```
#!/bin/bash
yum install -y aws-cli
aws s3 cp s3://your_bucket_name/ecs.config /etc/ecs/ecs.config
```

Amazon ECS コンテナエージェントをインストールする

Amazon ECS クラスターに Amazon EC2 インスタンスを登録する必要があり、そのインスタンスが Amazon ECS に最適化された AMI に基づく AMI を使用していない場合は、次の手順を使用して Amazon ECS コンテナエージェントを手動でインストールできます。これを行うには、リージョンの Amazon S3 バケットのいずれかから、または Amazon Elastic Container Registry Public からエージェントをダウンロードできます。リージョンの Amazon S3 バケットのいずれかからダウンロードする場合は、必要に応じて、PGP 署名を使用してコンテナエージェントファイルの有効性を検証できます。

> **Note**
>
> Amazon ECS と Docker サービスの両方の systemd ユニットには、両方のサービスを開始する前に cloud-init が終了するのを待つディレクティブがあります。Amazon EC2ユーザーデータの実行が終了するまで、cloud-init プロセスは終了したと見なされません。したがって、Amazon EC2 ユーザーデータを介して Amazon ECS または Docker を起動すると、デッドロックが発生する可能性があります。Amazon EC2 ユーザーデータを使用してコンテナエージェントを起動するには、systemctl enable --now --no-block ecs.service を使用できます。

非 Amazon Linux EC2 インスタンスに Amazon ECS コンテナエージェントをインストールする

Amazon EC2 インスタンスに Amazon ECS コンテナエージェントをインストールするには、リージョンの Amazon S3 バケットのいずれかからエージェントをダウンロードしてインストールできます。

> **Note**
>
> Amazon Linux AMI 以外を使用する場合、Amazon ECS エージェントがタスクレベルのリソース制限をサポートするためには、Amazon EC2 インスタンスは cgroup ドライバーの cgroupfs サポートが必要です。詳細については、GitHub で Amazon ECS エージェントについて参照してください。

各システムアーキテクチャの最新 Amazon ECS コンテナエージェントファイルは、リージョン別に参考として以下の表に示します。

リージョン	リージョン名	Amazon ECS init deb ファイル	Amazon ECS init rpm ファイル
us-east-2	米国東部 (オハイオ)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)

リージョン	リージョン名	Amazon ECS init deb ファイル	Amazon ECS init rpm ファイル
us-east-1	米国東部 (バージニア北部)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)
us-west-1	米国西部 (北カリフォルニア)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)
us-west-2	米国西部 (オレゴン)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)
ap-east-1	アジアパシフィック (香港)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)
ap-northeast-1	アジアパシフィック (東京)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)
ap-northeast-2	アジアパシフィック (ソウル)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)

リージョン	リージョン名	Amazon ECS init deb ファイル	Amazon ECS init rpm ファイル
ap-south-1	アジアパシフィック (ムンバイ)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)
ap-southeast-1	アジアパシフィック (シンガポール)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)
ap-southeast-2	アジアパシフィック (シドニー)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)
ca-central-1	カナダ (中部)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)
eu-central-1	欧州 (フランクフルト)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)
eu-west-1	欧州 (アイルランド)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)

リージョン	リージョン名	Amazon ECS init deb ファイル	Amazon ECS init rpm ファイル
eu-west-2	欧州 (ロンドン)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)
eu-west-3	欧州 (パリ)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)
sa-east-1	南米 (サンパウロ)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 Amazon ECS init aarch64 (aarch64)
us-gov-east-1	AWS GovCloud (米国東部)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)
us-gov-west-1	AWS GovCloud (米国西部)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)

非 Amazon Linux AMI を使用した Amazon EC2 インスタンスに Amazon ECS コンテナエージェントをインストールするには

1. Amazon ECSへのアクセスを許可する IAM; ロールを使用して Amazon EC2インスタンスを起動します。詳細については、「Amazon ECS コンテナインスタンスの IAM ロール」を参照してください。

2. インスタンスに接続します。
3. サポートされているバージョンの Docker をインスタンスにインストールします。詳細については、「Docker サポート」を参照してください。
4. Docker のバージョンをチェックして、システムがバージョンの最小要件を満たしていることを確認します。Docker のサポートの詳細については、「Amazon ECS EC2 コンテナインスタンス」を参照してください。

```
docker --version
```

5. オペレーティングシステムとシステムアーキテクチャ用の適切な Amazon ECS エージェントファイルをダウンロードし、インストールします。

   deb アーキテクチャの場合:

```
ubuntu:~$ curl -O https://s3.us-west-2.amazonaws.com/amazon-ecs-agent-us-west-2/amazon-ecs-init-latest.amd64.deb
ubuntu:~$ sudo dpkg -i amazon-ecs-init-latest.amd64.deb
```

   rpm アーキテクチャの場合:

```
fedora:~$ curl -O https://s3.us-west-2.amazonaws.com/amazon-ecs-agent-us-west-2/amazon-ecs-init-latest.x86_64.rpm
fedora:~$ sudo yum localinstall -y amazon-ecs-init-latest.x86_64.rpm
```

6. /lib/systemd/system/ecs.service ファイルを編集し、[Unit] セクションの最後に次の行を追加します。

```
After=cloud-final.service
```

7. (オプション) インスタンスを default クラスター以外のクラスターに登録するには、/etc/ecs/ecs.config ファイルを編集して以下の内容を追加します。次の例は MyCluster クラスターを指定します。

```
ECS_CLUSTER=MyCluster
```

   これらや他のエージェントランタイムオプションの詳細については、「Amazon ECS コンテナエージェントの設定」を参照してください。

> **Note**
>
> オプションで、エージェント環境変数を Amazon S3 に保存できます (これらの環境変数は、Amazon EC2 ユーザーデータを使用して、起動時にコンテナインスタンスにダウンロードできます)。プライベートリポジトリの認証情報などの機密情報については、この方法を推奨します。詳細については、「Amazon S3 に Amazon ECS コンテナインスタンスの設定を保存する」および「Amazon ECS での AWS 以外のコンテナイメージの使用」を参照してください。

8. ecs サービスを開始します。

```
ubuntu:~$ sudo systemctl start ecs
```

ホストネットワークモードで Amazon ECS エージェントを実行する

Amazon ECSコンテナエージェントを実行している場合、ecs-init は host ネットワークモードでコンテナエージェントのコンテナを作成します。これは、コンテナエージェントのコンテナでサポートされている唯一のネットワークモードです。

これにより、コンテナエージェントにより開始されたコンテナの Amazon EC2 インスタンスのメタデータサービスエンドポイント (http://169.254.169.254) へのアクセスをブロックできます。これにより、コンテナはコンテナインスタンスプロファイルから IAM ロール認証情報にアクセスできなくなります。また、タスクでは IAM タスクロールの認証情報のみが使用されます。詳細については、「Amazon ECS タスクの IAM ロール」を参照してください。

また、コンテナエージェントが docker0 ブリッジ上の接続とネットワークトラフィックで競合しないようにもできます。

Amazon ECS コンテナエージェントのログ設定パラメータ

Amazon ECS コンテナエージェントは、コンテナインスタンスのログを保存します。

コンテナエージェントバージョン 1.36.0 以降の場合、デフォルトでは、ログは Linux インスタンスの /var/log/ecs/ecs-agent.log および Windows インスタンスの C:\ProgramData\Amazon\ECS\log\ecs-agent.log にあります。

コンテナエージェントバージョン 1.35.0 以前の場合、デフォルトでは、ログは Linux インスタンスの `/var/log/ecs/ecs-agent.log.timestamp` および Windows インスタンスの `C:\ProgramData\Amazon\ECS\log\ecs-agent.log.timestamp` にあります。

デフォルトでは、エージェントログは 1 時間ごとにローテーションされ、最大 24 個のログが保存されます。

以下のコンテナエージェントの設定変数は、エージェントのデフォルトのログ記録動作を変更するために使用できます。使用可能なすべての設定パラメータの詳細については、Amazon ECS コンテナエージェントの設定 または GitHub の「Amazon ECS Agent README」を参照してください。

コンテナエージェントバージョン 1.36.0 以降では、`logfmt` 形式を使用した場合のログファイルは以下の例のようになります。

```
level=info time=2019-12-12T23:43:29Z msg="Loading configuration" module=agent.go
level=info time=2019-12-12T23:43:29Z msg="Image excluded from cleanup: amazon/amazon-
ecs-agent:latest" module=parse.go
level=info time=2019-12-12T23:43:29Z msg="Image excluded from cleanup: amazon/amazon-
ecs-pause:0.1.0" module=parse.go
level=info time=2019-12-12T23:43:29Z msg="Amazon ECS agent Version: 1.36.0, Commit:
 ca640387" module=agent.go
level=info time=2019-12-12T23:43:29Z msg="Creating root ecs cgroup: /ecs"
 module=init_linux.go
level=info time=2019-12-12T23:43:29Z msg="Creating cgroup /ecs"
 module=cgroup_controller_linux.go
level=info time=2019-12-12T23:43:29Z msg="Loading state!" module=statemanager.go
level=info time=2019-12-12T23:43:29Z msg="Event stream ContainerChange start
 listening..." module=eventstream.go
level=info time=2019-12-12T23:43:29Z msg="Restored cluster 'auto-robc'" module=agent.go
level=info time=2019-12-12T23:43:29Z msg="Restored from checkpoint file. I
 am running as 'arn:aws:ecs:us-west-2:0123456789:container-instance/auto-
robc/3330a8a91d15464ea30662d5840164cd' in cluster 'auto-robc'" module=agent.go
```

JSON 形式を使用した場合のログファイルは以下の例のようになります。

```
{"time": "2019-11-07T22:52:02Z", "level": "info", "msg": "Starting Amazon Elastic
 Container Service Agent", "module": "engine.go"}
```

プライベート Docker イメージ用の Amazon ECS コンテナインスタンスを設定する

Amazon ECS コンテナエージェントは、基本認証を使用してプライベートレジストリで認証できます。プライベートレジストリ認証を有効にすると、タスク定義のプライベート Docker イメージを使用できます。この機能は、EC2 を使用するタスクでのみサポートされています。

プライベートレジストリ認証を有効にする別の方法では、AWS Secrets Manager を使用して、プライベートレジストリ認証情報を安全に保存してから、コンテナの定義で参照します。これにより、プライベートリポジトリのイメージをタスクで使用できるようになります。このメソッドは、EC2 または Fargate のいずれかを使用するタスクをサポートします。詳細については、「Amazon ECS での AWS 以外のコンテナイメージの使用」を参照してください。

Amazon ECS コンテナエージェントでは、起動時に次の 2 つの環境変数を検索します。

- `ECS_ENGINE_AUTH_TYPE`。送信する認証データのタイプを指定します。
- `ECS_ENGINE_AUTH_DATA`。実際の認証情報が含まれます。

Amazon ECSに最適化されたAMIのLinuxバリアントは、コンテナインスタンスの起動時、およびサービスが開始されるたびに（sudo start ecs コマンドを使用して）、`/etc/ecs/ecs.config` ファイルをスキャンしてこれらの変数を探します。Amazon ECSに最適化されていないAMIは、これらの環境変数をファイルに保存し、`--env-file` *`path_to_env_file`*オプションを付けてコンテナエージェントを起動するdocker runコマンドに渡す必要があります。

> **Important**
>
> これらの認証環境変数を、 Amazon EC2 ユーザーデータを使用してインスタンス起動時に挿入することや、`--env` オプションを使用して docker run コマンドに渡すことはお勧めしません。これらの方法は、認証情報などの機密データには適していません。認証情報をコンテナインスタンスに安全に追加する方法については、「Amazon S3 に Amazon ECS コンテナインスタンスの設定を保存する」を参照してください。

認証形式

プライベートレジストリ認証には、`dockercfg` と `docker` の 2 つの形式を使用できます。

dockercfg 認証形式

dockercfg 形式は、docker login コマンドの実行時に作成した設定ファイルに保存されている認証情報を使用します。このファイルを作成するには、ローカルシステムで docker login を実行し、レジストリのユーザー名、パスワード、および E メールアドレスを入力します。コンテナインスタンスにログインして、そこでコマンドを実行することもできます。Docker のバージョンによって、ファイルは ~/.dockercfg または ~/.docker/config.json として保存されます。

```
cat ~/.docker/config.json
```

出力:

```
{
  "auths": {
    "https://index.docker.io/v1/": {
      "auth": "zq212MzEXAMPLE7o6T25Dk0i"
    }
  }
}
```

Important

Docker の新しいバージョンは、外部 auths オブジェクトを使用して、前の例に示したような設定ファイルを作成します。Amazon ECS エージェントは、auths オブジェクトを持たない、次の形式の dockercfg 認証データのみをサポートします。jq ユーティリティをインストール済みである場合、このデータは cat ~/.docker/config.json | jq .auths コマンドを使用して抽出できます。

```
cat ~/.docker/config.json | jq .auths
```

出力:

```
{
  "https://index.docker.io/v1/": {
    "auth": "zq212MzEXAMPLE7o6T25Dk0i",
    "email": "email@example.com"
  }
}
```

上記の例では、以下の環境変数を、Amazon ECS コンテナエージェントがランタイム時にロードする環境変数ファイル (Amazon ECSに最適化された AMI の場合は `/etc/ecs/ecs.config`) に追加しなければなりません。Amazon ECS に最適化された AMI を使用せず、 docker run を使用して手動でエージェントを開始する場合は、エージェント開始時に `--env-file` *`path_to_env_file`* オプションを使用して環境変数ファイルを指定します。

```
ECS_ENGINE_AUTH_TYPE=dockercfg
ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":
{"auth":"zq212MzEXAMPLE7o6T25Dk0i","email":"email@example.com"}}
```

次の構文を使用して複数のプライベートレジストリを設定できます。

```
ECS_ENGINE_AUTH_TYPE=dockercfg
ECS_ENGINE_AUTH_DATA={"repo.example-01.com":
{"auth":"zq212MzEXAMPLE7o6T25Dk0i","email":"email@example-01.com"},"repo.example-02.com":
{"auth":"fQ172MzEXAMPLEoF7225DU0j","email":"email@example-02.com"}}
```

docker 認証形式

docker 形式は、エージェントが認証する必要があるレジストリサーバーの JSON 表現を使用します。また、そのレジストリに必要な認証パラメータ (ユーザー名、パスワード、およびそのアカウントの E メールアドレスなど) も含まれます。Docker Hub アカウントの場合、JSON 表現は次のようになります。

```
{
  "https://index.docker.io/v1/": {
    "username": "my_name",
    "password": "my_password",
    "email": "email@example.com"
  }
}
```

この例では、以下の環境変数を、Amazon ECS コンテナエージェントが実行時にロードする環境変数ファイル (`/etc/ecs/ecs.config` Amazon ECSに最適化された AMI の場合は) に追加する必要があります。Amazon ECS に最適化された AMI を使用せず、 docker run を使用して手動でエージェントを開始する場合は、エージェント開始時に `--env-file` *`path_to_env_file`* オプションを使用して環境変数ファイルを指定します。

```
ECS_ENGINE_AUTH_TYPE=docker
```

```
ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":
{"username":"my_name","password":"my_password","email":"email@example.com"}}
```

次の構文を使用して複数のプライベートレジストリを設定できます。

```
ECS_ENGINE_AUTH_TYPE=docker
ECS_ENGINE_AUTH_DATA={"repo.example-01.com":
{"username":"my_name","password":"my_password","email":"email@example-01.com"},"repo.example-02
{"username":"another_name","password":"another_password","email":"email@example-02.com"}}
```

手順

以下の手順を使用して、コンテナインスタンスのプライベートレジストリをオン状態にします。

Amazon ECS に最適化された AMI のプライベートレジストリを有効にするには

1. SSH を使用してコンテナインスタンスにログインします。
2. /etc/ecs/ecs.config ファイルを開いて、レジストリとアカウントの ECS_ENGINE_AUTH_TYPE 値と ECS_ENGINE_AUTH_DATA 値を追加します。

   ```
   sudo vi /etc/ecs/ecs.config
   ```

   この例では Docker Hub のユーザーアカウントが認証されます。

   ```
   ECS_ENGINE_AUTH_TYPE=docker
   ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":
   {"username":"my_name","password":"my_password","email":"email@example.com"}}
   ```

3. エージェントが ECS_DATADIR 環境変数を使用して状態を保存しているかどうかを確認します。

   ```
   docker inspect ecs-agent | grep ECS_DATADIR
   ```

   出力:

   ```
   "ECS_DATADIR=/data",
   ```

> **Important**
>
> 前のコマンドで ECS_DATADIR 環境変数が返されない場合は、エージェントを停止する前に、このコンテナインスタンスで実行されているタスクをすべて停止する必要があります。より新しいエージェントは ECS_DATADIR 環境変数を使用して状態を保存するため、タスクが実行中でも問題なく停止および起動できます。詳細については、「Amazon ECS コンテナエージェントをアップデートする」を参照してください。

4. ecs サービスを停止します。

```
sudo stop ecs
```

出力:

```
ecs stop/waiting
```

5. ecs サービスを再起動します。

- Amazon ECS に最適化された Amazon Linux 2 AMI の場合:

```
sudo systemctl restart ecs
```

- Amazon ECS に最適化された Amazon Linux AMI の場合:

```
sudo stop ecs && sudo start ecs
```

6. (オプション) エージェント詳細分析 API オペレーションをクエリして、エージェントが実行されていることを確認し、新しいコンテナインスタンスに関する情報の一部を表示できます。詳細については、「the section called “コンテナの詳細分析”」を参照してください。

```
curl http://localhost:51678/v1/metadata
```

Amazon ECS タスクとイメージの自動クリーンアップ

タスクがコンテナインスタンスに配置されるたびに、Amazon ECS コンテナエージェントによって、タスクが参照するイメージがリポジトリの指定されたタグの最新版であるかどうかが確認されます。そうでない場合、デフォルトの動作では、エージェントがそれぞれのリポジトリからイメージ

を取得します。タスクやサービスで頻繁にイメージを更新する場合は、コンテナインスタンスのストレージが、もう使用しておらず、今後使用する予定もない Docker イメージですぐにいっぱいになってしまいます。例えば、継続的な統合や継続的なデプロイ (CI/CD) パイプラインを使用している場合です。

> **Note**
>
> Amazon ECS エージェントイメージのプル動作は、`ECS_IMAGE_PULL_BEHAVIOR` パラメータを使用してカスタマイズできます。詳細については、「Amazon ECS コンテナエージェントの設定」を参照してください。

同様に、停止したタスクに属するコンテナも、ログ情報、データボリューム、その他の生成物でコンテナインスタンスストレージを消費します。これらの生成物は、コンテナが予期せずに停止した場合のデバッグには役に立ちますが、このストレージのほとんどが一定期間後は安全に解放できます。

デフォルトでは、Amazon ECS コンテナエージェントによって、停止したタスクとコンテナインスタンスのどのタスクでも使用されていない Docker イメージがクリーンアップされます。

> **Note**
>
> 自動イメージクリーンアップ機能には、Amazon ECS コンテナエージェントのバージョン 1.13.0 以上が必要です。エージェントを最新バージョンに更新するには、「Amazon ECS コンテナエージェントをアップデートする」を参照してください。

以下のエージェント設定変数を使用して、タスクとイメージの自動クリーンアップの使い勝手を調整できます。コンテナインスタンスでこれらの変数を設定する方法の詳細については、「Amazon ECS コンテナエージェントの設定」を参照してください。

`ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION`

停止したタスクのコンテナを削除する際のデフォルトの待機時間。1 未満に設定すると、その値は無視されます。デフォルトでは、このパラメータは 3 時間に設定されていますが、アプリケーションでの必要に応じて、最短で 1 秒まで短縮できます。

> **Note**
>
> イメージクリーンアッププロセスは、そのイメージを参照するコンテナがある限りイメージを削除できません。コンテナが削除されると、参照されていない画像は、いずれも画像クリーンアップの設定パラメーターに基づいてクリーンアップの対象となります。

`ECS_DISABLE_IMAGE_CLEANUP`

この変数を `true` に設定した場合は、コンテナインスタンスでのイメージの自動クリーンアップはオフになり、イメージの自動的な削除は実行されません。

`ECS_IMAGE_CLEANUP_INTERVAL`

この変数は、イメージの自動クリーンアッププロセスが削除するイメージをチェックする頻度を指定します。デフォルトでは 30 分ごとですが、より頻繁にイメージを削除するには最短で 10 分まで短くできます。

`ECS_IMAGE_MINIMUM_CLEANUP_AGE`

この変数は、イメージが取得されてから、削除の対象になるまでの期間の最短時間を指定します。これは、取得されたばかりのイメージがクリーンアップされるのを防ぐために使用されます。デフォルトは 1 時間です。

`ECS_NUM_IMAGES_DELETE_PER_CYCLE`

この変数は、1 つのクリーンアップサイクルで削除するイメージの数を指定します。デフォルトは 5 で、最小は 1 です。

Amazon ECS コンテナエージェントが実行中でありイメージの自動クリーンアップがオフになっていない場合、エージェントは `ECS_IMAGE_CLEANUP_INTERVAL` 変数で決定された頻度で、実行中または停止されたコンテナから参照されていない Docker イメージをチェックします。使用されていないイメージが見つかり、それらが `ECS_IMAGE_MINIMUM_CLEANUP_AGE` 変数で指定された最短クリーンアップ時間よりも古い場合、エージェントは `ECS_NUM_IMAGES_DELETE_PER_CYCLE` 変数で指定された最大数までイメージを削除します。最後に参照された日時が最も古いイメージから順に削除されます。イメージが削除されると、エージェントは次の間隔まで待ってから、再度プロセスを繰り返します。

Amazon ECS でコンテナをスケジュールする

Amazon Elastic Container Service (Amazon ECS) は、コンテナ化されたワークロードに柔軟なスケジューリング機能を提供する共有状態のオプティミスティックな並列システムです。Amazon ECS スケジューラはAmazon ECS API と同じクラスターの状態情報を利用して、適切な配置を決定します。

Amazon ECS は、長時間実行されるタスクおよびアプリケーションにサービススケジューラを提供します。バッチジョブまたは単一実行タスクに対してスタンドアロンタスクまたはスケジュールされたタスクを手動で実行することもできます。ニーズに最適なタスクを実行するためのタスク配置戦略や制約を指定できます。例えば、タスクを複数のアベイラビリティーゾーンにまたがって実行するか、単一のアベイラビリティーゾーン内で実行するかを指定できます。さらに、オプションとして、独自のカスタムまたはサードパーティーのスケジューラでタスクを統合することもできます。

オプション	どのようなときに使うか	詳細情報
サービス	サービススケジューラは、長期実行するステートレスサービスやアプリケーションに適しています。サービススケジューラはオプションで、タスクがElastic Load Balancing ロードバランサーに登録されていることも確認します。サービススケジューラで維持されているサービスを更新できます。これには、新しいタスク定義のデプロイや、実行中のタスクの必要数の変更が含まれる場合があります。デフォルトでは、サービススケジューラによってタスクは複数のアベイラビリティーゾーン間で分散されます。ただし、タスク配置戦略と制約を	Amazon ECS サービス

オプション	どのようなときに使うか	詳細情報
	使用すると、タスク配置の決定をカスタマイズできます。	
スタンドアロンのタスク	スタンドアロンタスクは、作業を実行してから停止するバッチジョブなどのプロセスに適しています。例えば、キューに作業が入ったときに RunTask を呼び出す処理を入れることができます。タスクはキューから作業を引き出し、作業を実行して、その後終了します。RunTaskを使用すると、デフォルトのタスク配置戦略でクラスター全体にタスクをランダムに分散させることができます。これにより、単一のインスタンスが不均衡な数のタスクを取得する可能性を最小限に抑えることができます。	Amazon ECS スタンドアロンタスク

オプション	どのようなときに使うか	詳細情報
スケジュールされたタスク	スケジュールされたタスクは、クラスター内で設定された間隔で実行するタスクがある場合に適しており、EventBridge Scheduler を使用してスケジュールを作成できます。バックアップオペレーションまたはログスキャンのタスクを実行できます。作成する EventBridge Scheduler のスケジュールは、指定した時間にクラスター内の 1 つ以上のタスクを実行できます。スケジュールされたイベントは、特定の間隔 (*N* 分、時間または日ごとに実行されます) に設定できます。それ以外の場合、より複雑なスケジューリングには、`cron` 表現を使用できます。	Amazon EventBridge スケジューラを使用して Amazon ECS タスクをスケジュールする

コンピューティングオプション

Amazon ECS では、タスクまたはサービスを実行するインフラストラクチャを指定できます。キャパシティープロバイダー戦略または起動タイプを使用できます。

Fargate の場合、キャパシティープロバイダーは Fargate と Fargate Spot です。EC2 の場合、キャパシティープロバイダーは登録されたコンテナインスタンスを持つ Auto Scaling グループです。

キャパシティープロバイダー戦略は、クラスターに関連付けられたキャパシティープロバイダー全体にタスクを分散します。

キャパシティプロバイダー戦略では、クラスターに既に関連付けられており、`ACTIVE` または `UPDATING` ステータスを持つキャパシティプロバイダーのみを使用できます。クラスターの作成時に、キャパシティプロバイダーをクラスターに関連付けることができます。

キャパシティプロバイダー戦略では、オプションの ベース 値は、指定されたキャパシティプロバイダーで実行されるタスクの最小限の数を指定します。キャパシティープロバイダー戦略では、ベースを定義できるキャパシティープロバイダーは 1 つだけです。

ウェイト値は、指定したキャパシティプロバイダーを使用する起動済みタスクの総数に対する相対的な割合を決定します。次の例を考えます。2 つのキャパシティプロバイダーを含む戦略があり、両方の重みが 1 であるとします。ベースの割合が達すると、タスクは 2 つのキャパシティプロバイダー間で均等に分割されます。同じロジックを使用して、capacityProviderA に 1 の重みを指定し、capacityProviderB に 4 の重みを指定するとします。そして、capacityProviderA を使うタスクが 1 つ実行されるごとに、capacityProviderB を使うタスクが 4 つ実行されることになります。

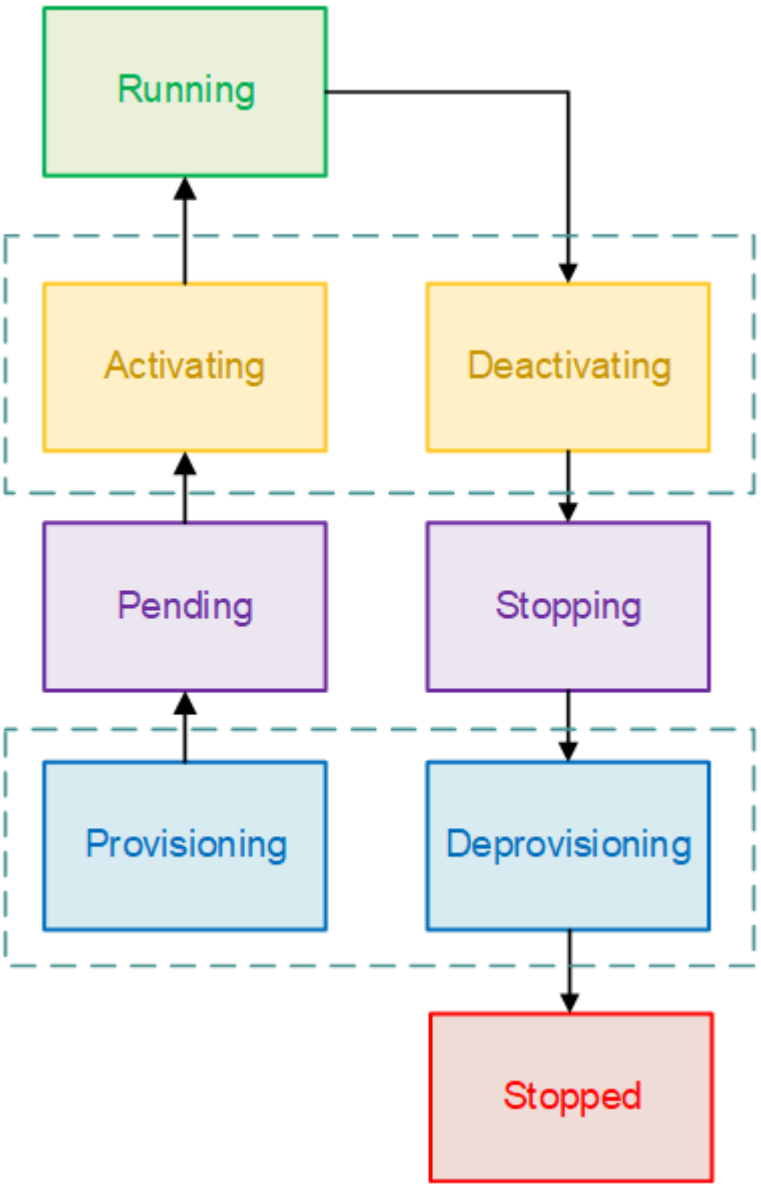
コンピューティングオプションでは、Fargate またはクラスターに手動で登録した Amazon EC2 インスタンスでタスクを直接起動します。

Amazon ECS タスクライフサイクル

タスクが手動またはサービスの一部として開始されると、自己終了するか手動で停止されるまでに、複数の状態を経由します。バッチジョブとして実行され、PENDING から RUNNING、その後 STOPPED と自然に進行するタスクもあります。他のタスクの場合は、サービスの一部であることもありますが、無制限に実行され続けるか、必要に応じてスケールアップまたはスケールダウンします。

タスクの停止や、スケールアップまたはスケールダウンのためのサービスの必要数の更新など、タスクのステータスの変更がリクエストされた場合、Amazon ECS コンテナエージェントはこれらの変更をタスクの最後の既知のステータス (lastStatus) およびタスクの目的のステータス (desiredStatus) として追跡します。タスクの最後の既知のステータスと目的のステータスの両方が、コンソールでまたは API あるいは AWS CLI でタスクを記述することにより、表示できます。

以下のフローチャートは、タスクのライフサイクルのフローを示しています。



ライフサイクル状態

それぞれのタスクのライフサイクル状態の説明は、以下のとおりです。

PROVISIONING

Amazon ECS は、タスクを起動する前に追加のステップを実行する必要があります。例えば、awsvpc ネットワークモードを使用するタスクの場合、Elastic Network Interface をプロビジョニングする必要があります。

保留中

これは、Amazon ECS が他の操作を実行するためにコンテナエージェントで待機している遷移状態です。タスクは、そのタスクに利用可能なリソースがあるまで保留状態のままです。

アクティブ化中

これは、タスクが起動された後で、タスクが RUNNING 状態に移行する前に、Amazon ECS が追加の手順を実行する必要がある移行状態です。これは、Amazon ECS がコンテナイメージをプルし、コンテナを作成し、タスクネットワークを設定し、ロードバランサーのターゲットグループを登録し、サービス検出を設定する状態です。

RUNNING

タスクを正常に実行中です。

非アクティブ化中

これは、タスクを停止する前に Amazon ECS が追加のステップを実行する必要がある移行状態です。例えば、Elastic Load Balancings ターゲットグループを使用するように設定されたサービスの一部であるタスクの場合、この状態でターゲットグループの登録解除が行われます。

停止中

これは、Amazon ECS が他の操作を実行するためにコンテナエージェントで待機している遷移状態です。

Linux コンテナの場合、コンテナエージェントはコンテナイメージで定義された停止シグナルを送信して、STOPSIGNAL の指示を使用してアプリケーションを終了およびシャットダウンする必要があることを通知します。デフォルトでは、SIGTERM です。次に、タスク定義で設定された StopTimeout 期間待機してから SIGKILL を送信します。

プロビジョン解除中

Amazon ECS は、タスクが停止された後で、タスクが STOPPED 状態に移行する前に、追加の手順を実行する必要があります。例えば、awsvpc ネットワークモードを使用するタスクの場合、Elastic Network Interface をデタッチして削除する必要があります。

停止

タスクは正常に停止しました。

エラーが原因でタスクが停止した場合は、「Amazon ECS の停止したタスクのエラーを表示する」を参照してください。

削除済み

これは、タスクが停止したときの移行状態です。この状態はコンソールには表示されませんが、`describe-tasks` に表示されます。

Amazon ECS がタスクをコンテナインスタンスに配置する方法

タスク配置を使用して、アベイラビリティーゾーンやインスタンスタイプなど、特定の基準を満たすコンテナインスタンスにタスクを配置するように Amazon ECS を設定できます。

以下はタスク配置のコンポーネントです。

- タスク配置戦略 - タスク配置またはタスクの終了でコンテナインスタンスを選択するためのアルゴリズムです。例えば、Amazon ECS でランダムにコンテナインスタンスを選択することも、インスタンスのグループ間に均等に分散されているタスクを持つコンテナインスタンスを選択することもできます。
- タスクグループ - データベースタスクなどの関連タスクのグループ。
- タスク配置制約事項 - コンテナインスタンスにタスクを配置するために満たす必要があるルールです。この制約が満たされない場合、タスクは配置されず、`PENDING` 状態のままになります。たとえば、制約を使用して、特定のインスタンスタイプにのみタスクを割り当てることができます。

Amazon ECS には、容量オプションごとに異なるアルゴリズムがあります。

Amazon ECS マネージドインスタンス

Amazon ECS マネージドインスタンスで実行されるタスクの場合、Amazon ECS はタスクを配置する場所と、タスク数をスケールダウンするときに終了するタスクを決定する必要があります。Amazon ECS は、キャパシティプロバイダーの起動テンプレートで指定されたインスタンス要件、CPU やメモリなどのタスク定義で指定された要件、およびタスクの配置に関する制約に基づいて、この決定を行います。

> **Note**
>
> Amazon ECS マネージドインスタンスはタスク配置戦略をサポートしていません。Amazon ECS は、アクセス可能なアベイラビリティーゾーンにタスクを分散するよう最善を尽くします。

Amazon ECSがタスクを配置する際は、以下のプロセスでコンテナインスタンスを選択します。

1. タスク定義の CPU、GPU、メモリ、ポートの要件を満たすコンテナインスタンスを識別します。
2. タスク配置の制約事項を満たすコンテナインスタンスを識別します。
3. キャパシティプロバイダーの起動テンプレートで指定されたインスタンスの要件を満たしているコンテナインスタンスを特定します。
4. タスクを配置するコンテナインスタンスを選択します。

EC2

EC2 起動タイプを使用するタスクには、Amazon ECS ではタスク定義で指定されている CPU やメモリなどの要件に基づいてタスクを配置する場所を決定する必要があります。同様に、タスク数を減らすときも、Amazon ECS でどのタスクを終了させるか決定する必要があります。タスク配置の戦略と制約を適用することで、Amazon ECS がタスクを配置および終了する方法をカスタマイズできます。

デフォルトのタスク配置戦略は、タスクを手動 (スタンドアロンタスク) で実行するのか、それともサービス内で実行するのかによって異なります。Amazon ECS サービスの一部として実行されるタスクの場合、タスク配置戦略は `attribute:ecs.availability-zone` を使用した `spread` です。サービス内にないタスクには、デフォルトのタスク配置の制約はありません。詳細については、「Amazon ECS でコンテナをスケジュールする」を参照してください。

> **Note**
>
> タスク配置戦略はベストエフォートです。Amazon ECSは、最適な配置オプションが利用できない場合でも、タスクの配置を試みます。ただし、タスク配置の制約が有効な場合、タスクを配置できないことがあります。

タスク配置戦略と制約は併用できます。例えば、タスク配置戦略とタスク配置制約を使用して、アベイラビリティーゾーン間でタスクを分散し、各アベイラビリティーゾーン内のメモリに基づいてビンパックタスクを分散できます。ただし、G2 インスタンスのみです。

Amazon ECSがタスクを配置する際は、以下のプロセスでコンテナインスタンスを選択します。

1. タスク定義の CPU、GPU、メモリ、ポートの要件を満たすコンテナインスタンスを識別します。
2. タスク配置の制約事項を満たすコンテナインスタンスを識別します。

3. タスク配置戦略を満たすコンテナインスタンスを識別します。
4. タスクを配置するコンテナインスタンスを選択します。

Fargate

タスクの配置戦略および制約事項は、Fargate を使用しているタスクをサポートしていません。Fargate は、アクセス可能なアベイラビリティーゾーンにタスクを分散するよう最善を尽くします。キャパシティープロバイダーに Fargate と Fargate Spot の両方が含まれている場合、スプレッドの動作はキャパシティープロバイダーごとに異なります。

Amazon ECS マネージドインスタンスの自動スケーリングとタスク配置

Amazon ECS マネージドインスタンスは、インテリジェントなアルゴリズムを使用してクラスターの容量を自動的にスケーリングし、インフラストラクチャ全体にタスクを効率的に配置します。これらのアルゴリズムの仕組みを理解することで、サービス設定を最適化し、配置動作をトラブルシューティングできます。

タスク配置アルゴリズム

Amazon ECS マネージドインスタンスは、タスクのスケジュールを設定する際に可用性、リソース使用率、ネットワーク要件のバランスを確保する高度な配置アルゴリズムを使用します。

分散アベイラビリティーゾーン

デフォルトでは、Amazon ECS マネージドインスタンスは複数のアベイラビリティーゾーンにタスクを分散することで可用性を優先します。

- 複数のタスクが配置されているサービスの場合、Amazon ECS マネージドインスタンスは、可能であれば異なるアベイラビリティーゾーンの少なくとも 3 つのインスタンスに分散します。
- この動作は耐障害性を実現しますが、インスタンスあたりのリソース使用率が低下する可能性があります。
- 拡散アベイラビリティーゾーンは、ビンのパッキングの最適化よりも優先されます

ビンのパッキング動作

Amazon ECS マネージドインスタンスは、リソース使用率を最大化するためにビンパッキングを実行できますが、この動作はネットワーク設定の影響を受けます。

- ビンのパッキングを実現するには、単一のサブネットを使用するようにサービスを設定します。
- マルチサブネット設定では、リソース密度よりもアベイラビリティーゾーンの分散が優先されます。
- ビンのパッキングは、スケーリングイベント中よりも初回サービス起動時に実施される可能性が高くなります。

ENI の密度に関する考慮事項

awsvpc ネットワークモードを使用するサービスの場合、Amazon ECS マネージドインスタンスは配置を決定する際に Elastic Network Interface (ENI) の密度を考慮します。

- awsvpc モードの各タスクには専用の ENI が必要です
- インスタンスタイプには、タスク密度に影響するさまざまな ENI 制限があります
- Amazon ECS マネージドインスタンスは、ターゲットインスタンスを選択するときに ENI の可用性を考慮します

> **Note**
>
> ENI 密度の計算は、配置の決定を最適化するために継続的に改善されています。

キャパシティプロバイダーの決定ロジック

Amazon ECS マネージドインスタンスのキャパシティプロバイダーは、次の複数の要因に基づいてスケーリングと配置を決定します。

リソースの要件

: 保留中のタスクの CPU、メモリ、ネットワークに関する要件

インスタンスの可用性

: 既存のインスタンス全体の現在の容量と使用率

ネットワーキングの制約

: サブネット設定と ENI の可用性

アベイラビリティーゾーンの分散

: 複数のアベイラビリティーゾーンにわたる耐障害性の維持

設定オプション

サブネットの選択戦略

サブネット設定は、タスク配置の動作に大きな影響を与えます。

複数のサブネット (デフォルト)

高可用性を実現するためにアベイラビリティーゾーンの拡散を優先する

インスタンスあたりのリソース使用率が低下する可能性があります

耐障害性を必要とする本番稼働用ワークロードに推奨

単一サブネット

ビンパッキングを有効にしてリソース使用率を高める

タスクを 1 つのアベイラビリティーゾーンに集約することで耐障害性を低減する

開発またはコスト最適化ワークロードに適しています

ネットワークモードに関する考慮事項

選択したネットワークモードは、配置の決定に影響します。

- awsvpc モード – 各タスクには専用の ENI が必要であり、インスタンスあたりのタスク密度が制限されます。
- host モード – タスクはホストのネットワークを直接使用し、配置は主にリソースの可用性によって決定されます。

CPU アーキテクチャに関する考慮事項

タスク定義で指定した cpuArchitecture は、特定のアーキテクチャにタスクを配置するために使用されます。cpuArchitecture を指定しない場合、Amazon ECS はキャパシティプロバイダーの設定に基づいて、使用可能な CPU アーキテクチャにタスクを配置しようとします。X86_64 または ARM64 のどちらかを指定できます。

タスク配置のトラブルシューティング

一般的な配置パターン

予想される配置パターンを理解すると、通常の動作と潜在的な問題を区別できるようになります。

分散ディストリビューション

部分的な使用率で複数のインスタンスに分散されたタスク

複数のサブネットを使用する場合の通常の動作

リソース効率よりも可用性が優先されることを示します

集約された配置

使用率の高い少数のインスタンスに配置された複数のタスク

単一サブネット設定を使用する場合に想定されます

最初のサービス起動中に発生する可能性があります

不均等な分散

一部のインスタンスの使用率が非常に高い一方で、他のインスタンスについては使用率が低い状態が継続します。

ENI の制限またはリソースの制約を示している場合があります

インスタンスタイプとネットワーク設定の確認を検討する

配置動作の最適化

特定の要件に合わせてタスク配置を最適化するには:

1. コスト最適化のニーズと照らし合わせて可用性要件を評価する
2. 優先順位に基づいて適切なサブネット設定を選択する
3. ネットワークモードに適した ENI 容量を持つインスタンスタイプを選択する
4. 配置パターンをモニタリングし、必要に応じて設定を調整する

ベストプラクティス

- 本稼働ワークロードの場合 – 異なるアベイラビリティーゾーン間で複数のサブネットを使用して高可用性を確保し、リソース使用率のトレードオフを受け入れます。
- 開発またはテストの場合 – リソース使用率を最大化し、コストを削減するために単一サブネット設定を検討する
- **awsvpc** モードの場合 – 配置の制約に抵触することを回避するために、十分な ENI 容量を持つインスタンスタイプを選択する

- コスト最適化の場合 – 使用率パターンをモニタリングし、可用性と効率性のバランスを取るようにサービス設定を調整する
- トラブルシューティングの場合 — 予期しない配置パターンを調査するときに、サブネット設定とネットワークモードを確認する

戦略を使用して Amazon ECS タスク配置を定義する

EC2 起動タイプを使用するタスクには、Amazon ECS ではタスク定義で指定されている CPU やメモリなどの要件に基づいてタスクを配置する場所を決定する必要があります。同様に、タスク数を減らすときも、Amazon ECS でどのタスクを終了させるか決定する必要があります。タスク配置の戦略と制約を適用することで、Amazon ECS がタスクを配置および終了する方法をカスタマイズできます。

デフォルトのタスク配置戦略は、タスクを手動 (スタンドアロンタスク) で実行するのか、それともサービス内で実行するのかによって異なります。Amazon ECS サービスの一部として実行されるタスクの場合、タスク配置戦略は `attribute:ecs.availability-zone` を使用した `spread` です。サービス内にないタスクには、デフォルトのタスク配置の制約はありません。詳細については、「Amazon ECS でコンテナをスケジュールする」を参照してください。

> **Note**
>
> タスク配置戦略はベストエフォートです。Amazon ECSは、最適な配置オプションが利用できない場合でも、タスクの配置を試みます。ただし、タスク配置の制約が有効な場合、タスクを配置できないことがあります。

タスク配置戦略と制約は併用できます。例えば、タスク配置戦略とタスク配置制約を使用して、アベイラビリティーゾーン間でタスクを分散し、各アベイラビリティーゾーン内のメモリに基づいてビンパックタスクを分散できます。ただし、G2 インスタンスのみです。

Amazon ECSがタスクを配置する際は、以下のプロセスでコンテナインスタンスを選択します。

1. タスク定義の CPU、GPU、メモリ、ポートの要件を満たすコンテナインスタンスを識別します。
2. タスク配置の制約事項を満たすコンテナインスタンスを識別します。
3. タスク配置戦略を満たすコンテナインスタンスを識別します。
4. タスクを配置するコンテナインスタンスを選択します。

タスク配置戦略は、サービス定義または placementStrategy パラメータを使用したタスク定義で指定できます。

```
"placementStrategy": [
    {
        "field": "The field to apply the placement strategy against",
        "type": "The placement strategy to use"
    }
]
```

タスクを実行するとき (RunTask)、新しいサービスを作成するとき (CreateService)、または既存のサービスを更新するとき (UpdateService) に戦略を指定できます。

次の表は利用可能なタイプとフィールドを説明しています。

型	有効なフィールド値	
binpack タスクはコンテナインスタンスに配置され、未使用の CPU またはメモリを最小にします。この戦略は、使用中のコンテナインスタンスの数を最小限に抑えます。 この戦略が使用されてスケールインアクションが実行されると、Amazon ECS はタスクを終了します。タスクが終了した後にコンテナインスタンスに残されたリソース量に基づいてこれが実行されます。タスクの終了後に利用可能なリソースが最も多く残るコンテナインスタンスが、そのタスクを終了されます。	• cpu • メモリ	
random	使用されていない	

型	有効なフィールド値	
タスクはランダムに配置されます。		
`spread` タスクは指定された値に基づいて均等に配置されます。サービスタスクはそのサービスからのタスクに基づいて分散されます。スタンドアロンタスクは、同じタスクグループからのタスクに基づいて分散されます。タスクグループの詳細については、「Amazon ECS タスクをグループ化する」を参照してください。 spread 戦略が使用されてスケールインアクションが実行されると、Amazon ECS は、アベイラビリティーゾーン間のバランスを維持するタスクを選択して終了します。アベイラビリティーゾーン内では、タスクはランダムに選択されます。	• `instanceId` (または同じ効果のある `host`) • `attribute:ecs.availability-zone` のようなコンテナインスタンスに適用される任意のプラットフォームまたはカスタム属性	

タスク配置の戦略は、既存のサービスに対しても更新できます。詳細については、「Amazon ECS がタスクをコンテナインスタンスに配置する方法」を参照してください。

実行する順序で戦略の配列を作成することで、複数の戦略を使用するタスク配置戦略を作成できます。例えば、複数のアベイラビリティーゾーンにタスクを分散し、各アベイラビリティーゾーン内のメモリに基づいてタスクをビンパックする場合、アベイラビリティーゾーン戦略を指定し、その後にメモリ戦略を指定します。戦略の例については、「Amazon ECS タスク配置戦略の例」を参照してください。

Amazon ECS タスク配置戦略の例

次のアクションを使用してタスク配置戦略を指定できます。CreateService、UpdateService、および RunTask。

例

- 複数のアベイラビリティーゾーンでタスクを均等に分散する
- すべてのインスタンスでタスクを均等に分散する
- メモリに基づいてタスクをビンパックする
- タスクをランダムに配置します。
- 複数のアベイラビリティーゾーンでタスクを均等に分散し、各アベイラビリティーゾーン内で複数のインスタンスでタスクを均等に分散する
- 複数のアベイラビリティーゾーンでタスクを均等に分散し、各アベイラビリティーゾーン内でメモリに基づいてタスクをビンパックする
- 複数のインスタンスでタスクを均等に分散し、メモリに基づいてタスクをビンパックする

複数のアベイラビリティーゾーンでタスクを均等に分散する

次の戦略は、アベイラビリティーゾーン間でタスクを均等に分散します。

```
"placementStrategy": [
    {
        "field": "attribute:ecs.availability-zone",
        "type": "spread"
    }
]
```

すべてのインスタンスでタスクを均等に分散する

次の戦略は、すべてのインスタンス間でタスクを均等に分散します。

```
"placementStrategy": [
    {
        "field": "instanceId",
        "type": "spread"
    }
]
```

メモリに基づいてタスクをビンパックする

次の戦略はメモリに基づいてタスクをビンパックします。

```
"placementStrategy": [
    {
        "field": "memory",
        "type": "binpack"
    }
]
```

タスクをランダムに配置します。

次の戦略はタスクをランダムに配置します。

```
"placementStrategy": [
    {
        "type": "random"
    }
]
```

複数のアベイラビリティーゾーンでタスクを均等に分散し、各アベイラビリティーゾーン内で複数のインスタンスでタスクを均等に分散する

次の戦略は、アベイラビリティーゾーン間でタスクを均等に分散し、次に各アベイラビリティーゾーン内でインスタンスを均等に分散します。

```
"placementStrategy": [
    {
        "field": "attribute:ecs.availability-zone",
        "type": "spread"
    },
    {
        "field": "instanceId",
        "type": "spread"
    }
]
```

複数のアベイラビリティーゾーンでタスクを均等に分散し、各アベイラビリティーゾーン内でメモリに基づいてタスクをビンパックする

次の戦略は、アベイラビリティーゾーン間でタスクを均等に分散し、次に各アベイラビリティーゾーン内でメモリに基づいてタスクをビンパックします。

```
"placementStrategy": [
    {
        "field": "attribute:ecs.availability-zone",
        "type": "spread"
    },
    {
        "field": "memory",
        "type": "binpack"
    }
]
```

複数のインスタンスでタスクを均等に分散し、メモリに基づいてタスクをビンパックする

次の戦略は、すべてのインスタンスでタスクを均等に分散し、各インスタンス内のメモリに基づいてタスクをビンパックします。

```
"placementStrategy": [
    {
        "field": "instanceId",
        "type": "spread"
    },
    {
        "field": "memory",
        "type": "binpack"
    }
]
```

Amazon ECS タスクをグループ化する

一連の関連するタスクを識別してタスクグループで配置できます。同じタスクグループ名のすべてのタスクは、spreadタスクの配置戦略の使用時に、セットとして見なされます。例えば、データベースおよびウェブサーバーなど、1 つのクラスターで異なるアプリケーションを実行していることを想定します。データベースを確実にアベイラビリティーゾーン間に均等に分散するには、それらのデータベースをdatabasesという名前のタスクグループに追加し、spreadタスク配置戦略を使用し

ます。詳細については、「戦略を使用して Amazon ECS タスク配置を定義する」を参照してください。

タスクグループはタスク配置の制約にも使用できます。memberOf 制約でタスクグループを指定したとき、タスクは指定されたタスクグループ内のコンテナインスタンスにのみ送信されます。例については、Amazon ECS タスク配置の制約事項の例を参照してください。

デフォルトでは、カスタムタスクグループ名が指定されていない場合、スタンドアロンタスクはタスク定義ファミリ名 (例えば family:my-task-definition) をタスクグループ名として使用します。サービスの一部として起動されたタスクは、サービス名をタスクグループ名として使用し、変更することはできません。

タスクグループには次の要件が適用されます。

- タスクグループ名は 255 文字以下である必要があります。
- 各タスクを 1 つのグループと同一にできます。
- タスクを起動後、タスクグループを変更することはできません。

Amazon ECS がタスクに使用するコンテナインスタンスを定義する

タスク配置の制約事項は、Amazon ECS がインスタンス上でタスクを実行できるかどうかを判断するために使用するコンテナインスタンスに関するルールです。少なくとも 1 つのコンテナインスタンスが制約に一致する必要があります。制約に一致するインスタンスがない場合、タスクは PENDING 状態のままになります。新しいサービスを作成するか、既存のサービスを更新するときに、サービスのタスク用にタスク配置制約を指定できます。

placementConstraint パラメータを使用して、サービス定義、タスク定義、またはタスクでタスク配置制約事項を指定できます。

```
"placementConstraints": [
    {
        "expression": "The expression that defines the task placement constraints",
        "type": "The placement constraint type to use"
    }
]
```

次の表は、パラメータの使用法の説明です。

[Constraint type (制約タイプ)]	次の場合に指定できます。	
`distinctInstance` それぞれのアクティブタスクを別々のコンテナインスタンスに配置します。 Amazon ECS は、タスク配置におけるタスクの目的のステータスを確認します。例えば、既存のタスクの目的のステータスが `STOPPED` の場合 (ただし、最後のステータスが目的のステータスではない場合)、`distinctInstance` 配置の制約事項にもかかわらず、新しい受信タスクを同じインスタンスに配置することができます。したがって、同じインスタンスで最後のステータスが `RUNNING` のタスクが 2 つ表示される場合があります。 **Important** タスクの強力な分離を求めているお客様には、Fargate を使用することをお勧めします。Fargate はハードウェア仮想化環境で各タスクを実行します。これにより、これらのコンテナ化されたワークロードがネットワー	• タスク RunTask を実行する • 新しいサービス CreateService の作成、	

[Constraint type (制約タイプ)]	次の場合に指定できます。	
クインターフェイス、Fargate の一時ストレージ、CPU、またはメモリを他のタスクと共有しないことが保証されます。		
`memberOf` 式を満たすコンテナインスタンスにタスクを配置します。	• タスク RunTask を実行する • 新しいサービス CreateService の作成、 • 新しいタスク定義 RegisterTaskDefinition の作成 • タスク定義 RegisterTaskDefinition の新しいバージョンの作成 • サービス UpdateService の更新	

`memberOf` 制約事項タイプを使用すると、Amazon ECS がタスクを配置できるコンテナインスタンスを定義するクラスタークエリ言語を使用して式を作成できます。この式は、コンテナインスタンスを属性別にグループ化する方法です。この式は、`placementConstraint` の `expression` パラメータに含まれます。

Amazon ECS コンテナインスタンス属性

コンテナインスタンスに属性と呼ばれるカスタムメタデータを追加できます。各属性には名前とオプションの文字列値があります。Amazon ECSが提供する組み込み属性を使用することも、カスタム属性を定義することもできます。

次のセクションでは、組み込み型、オプション型、カスタム型属性のサンプルが含まれています。

組み込み属性

Amazon ECSは次の属性を自動的にコンテナインスタンスに適用します。

`ecs.ami-id`

インスタンスの起動に使用される AMI の ID。この属性の値の例は「`ami-1234abcd`」です。

`ecs.availability-zone`

インスタンスのアベイラビリティーゾーン。この属性の値の例は「`us-east-1a`」です。

`ecs.instance-type`

インスタンスのインスタンスタイプ。この属性の値の例は「`g2.2xlarge`」です。

`ecs.os-type`

インスタンスのオペレーティングシステム。この属性に指定できる値は `linux` と `windows` です。

`ecs.os-family`

インスタンスのオペレーティングシステムのバージョン。

Linux インスタンスの場合、有効な値は `LINUX` です。Windows インスタンスの場合、ECS は値を `WINDOWS_SERVER_<OS_Release>_<FULL or CORE>` 形式で設定します。有効な値は、`WINDOWS_SERVER_2022_FULL`、`WINDOWS_SERVER_2022_CORE`、`WINDOWS_SERVER_20H2_COR` および `WINDOWS_SERVER_2016_FULL` です。

これは、Windows コンテナおよび Windows containers on AWS Fargate にとって重要です。なぜなら、すべての Windows コンテナの OS バージョンがホストのそれと一致する必要があるからです。コンテナイメージの Windows バージョンがホストと異なる場合、コンテナは起動しません。詳細については、マイクロソフトのドキュメント Web サイトの「Windows コンテナバージョンの互換性」を参照してください。

クラスターが複数の Windows バージョンを実行している場合、次の配置制約を使用して、同じバージョンで実行されている EC2 インスタンスにタスクが配置されるようにすることができます: `memberOf(attribute:ecs.os-family == WINDOWS_SERVER_<OS_Release>_<FULL or CORE>)`。詳細については、「the section called “Amazon ECS に最適化された Windows AMI メタデータを取得する”」を参照してください。

`ecs.cpu-architecture`

インスタンスの CPU アーキテクチャ。この属性の値の例は `x86_64` と `arm64` です。

`ecs.vpc-id`

インスタンスが起動された VPC。この属性の値の例は「`vpc-1234abcd`」です。

`ecs.subnet-id`

インスタンスが使用しているサブネット。この属性の値の例は「`subnet-1234abcd`」です。

> **Note**
>
> Amazon ECS マネージドインスタンスは、次の属性のサブセットをサポートしています。
>
> - `ecs.subnet-id`
> - `ecs.availability-zone`
> - `ecs.instance-type`
> - `ecs.cpu-architecture`

オプションの属性

Amazon ECS では、コンテナインスタンスに次の属性を追加することができます。

`ecs.awsvpc-trunk-id`

この属性が存在する場合、インスタンスにトランクネットワークインターフェイスがあります。詳細については、「Amazon ECS Linux コンテナインスタンスのネットワークインターフェイスを増やす」を参照してください。

`ecs.outpost-arn`

この属性が存在する場合は、Outpost の Amazon リソースネーム (ARN) が含まれます。詳細については、「the section called “AWS OutpostsのAmazon Elastic Container Service”」を参照してください。

`ecs.capability.external`

この属性が存在する場合、インスタンスは外部インスタンスとして識別されます。詳細については、「外部インスタンスの Amazon ECS クラスター」を参照してください。

カスタム属性

コンテナインスタンスに、カスタム属性を適用できます。例えば、「stack」という名前で「prod」という値の属性を定義できます。

カスタム属性を指定するとき、次の点を考慮する必要があります。

- name は、1 ~ 128 文字で指定する必要があります。文字 (大文字と小文字)、数字、ハイフン、アンダースコア、スラッシュ、バックスラッシュ、ピリオドを使用できます。
- value は、1 ~ 128 文字で指定する必要があります。文字 (大文字と小文字)、数字、ハイフン、アンダースコア、ピリオド、アットマーク (@)、スラッシュ、バックスラッシュ、コロン、またはスペースを使用できます。値の先頭または末尾にホワイトスペースを含めることはできません。

Amazon ECS タスク用のコンテナインスタンスを定義する式を作成する

クラスタークエリは、オブジェクトをグループ化できる式です。例えば、アベイラビリティーゾーン、インスタンスタイプ、カスタムメタデータなどの属性でコンテナインスタンスをグループ化できます。詳細については、「Amazon ECS コンテナインスタンス属性」を参照してください。

コンテナインスタンスのグループを定義した後、グループに基づいてコンテナインスタンスのタスクを配置するように Amazon ECS をカスタマイズできます。詳細については「Amazon ECS タスクとしてのアプリケーションの実行」および「Amazon ECS のローリング更新デプロイの作成」を参照してください。また、コンテナインスタンスをリスト表示する際にグループフィルタを適用できます。

式の構文

式の構文は次のとおりです。

```
subject operator [argument]
```

件名

評価する属性またはフィールド。

`agentConnected`

Amazon ECS コンテナエージェント接続ステータスでコンテナインスタンスを選択します。このフィルタを使用して、切断されたコンテナエージェントでインスタンスを検索します。

有効な演算子: equals (==)、not_equals (!=)、in、not_in (!in)、matches (=~)、not_matches (!~)

`agentVersion`

Amazon ECS コンテナエージェントバージョンでコンテナインスタンスを選択します。このフィルタを使用して、Amazon ECS コンテナエージェントの古くなったバージョンを実行しているインスタンスを検索します。

有効な演算子: equals (==)、not_equals (!=)、greater_than (>)、greater_than_equal (>=)、less_than (<)、less_than_equal (<=)

`attribute:attribute-name`

属性でコンテナインスタンスを選択します。詳細については、「Amazon ECS コンテナインスタンス属性」を参照してください。

`ec2InstanceId`

Amazon EC2 インスタンス ID で、コンテナインスタンスを選択します。

有効な演算子: equals (==)、not_equals (!=)、in、not_in (!in)、matches (=~)、not_matches (!~)

`registeredAt`

コンテナインスタンス登録日で、コンテナインスタンスを選択します。このフィルタを使用して、新しく登録されたインスタンスまたは古いインスタンスを検索します。

有効な演算子: equals (==)、not_equals (!=)、greater_than (>)、greater_than_equal (>=)、less_than (<)、less_than_equal (<=)

有効な日付形式: 2018-06-18T22:28:28+00:00、2018-06-18T22:28:28Z、2018-06-18T22:28:28、2018-06-18

`runningTasksCount`

実行中のタスクの数でコンテナインスタンスを選択します。このフィルタを使用して、空またはほぼ空 (実行中のタスクがほぼない) のインスタンスを検索します。

有効な演算子: equals (==)、not_equals (!=)、greater_than (>)、greater_than_equal (>=)、less_than (<)、less_than_equal (<=)

`task:group`

コンテナインスタンスをタスクグループで選択します。詳細については、「Amazon ECS タスクをグループ化する」を参照してください。

オペレーター

比較演算子。以下の演算子がサポートされています。

演算子	説明
==, が	文字列が等価

演算子	説明
!=、not_equals	文字列が不等価
>、greater_than	以上
>=、greater_than_equal	以上
<、less_than	未満
<=、less_than_equal	以下
exists	対象が存在
!exists、not_exists	サブジェクトが存在しません
情報	引数リストの値
!in、not_in	引数リストにない値
=~、matches	パターンが一致
!~、not_matches	パターンが不一致

> **Note**
>
> 1 つの式に括弧を含めることはできません。ただし、複合式で優先順位を指定するために括弧を使用できます。

引数

多くの演算子は、引数がリテラル値です。

in および not_in 演算子は、引数として引数リストを想定しています。次のように引数リストを指定します。

```
[argument1, argument2, ..., argumentN]
```

matches と not_matches 演算子は、Java の正規表現構文に準拠する引数を想定しています。詳細については、java.util.regex.Pattern を参照してください。

複合式

次のブール演算子を使用して式を結合できます。

- &&, および
- ||, または
- !, 先頭の文字に

括弧を使用して、優先度を指定できます。

```
(expression1 or expression2) and expression3
```

式の例

以下に式の例を示します。

例: 文字列が等価

次の式は指定されたインスタンスタイプのインスタンスを選択します。

```
attribute:ecs.instance-type == t2.small
```

例: 引数リスト

次の式は、アベイラビリティーゾーンが us-east-1a または us-east-1b のインスタンスを選択します。

```
attribute:ecs.availability-zone in [us-east-1a, us-east-1b]
```

例: 複合式

次の表現は、us-east-1d アベイラビリティーゾーンにはない G2 インスタンスを選択します。

```
attribute:ecs.instance-type =~ g2.* and attribute:ecs.availability-zone != us-east-1d
```

例: タスクのアフィニティ

次の式は、service:production グループのタスクをホストするインスタンスを選択します。

```
task:group == service:production
```

例: タスクのアンチアフィニティ

次の表現は、データベースグループでタスクをホストしないインスタンスを選択します。

```
not(task:group == database)
```

例: 実行中のタスクの数

次の式は、1 つのタスクのみを実行しているインスタンスを選択します。

```
runningTasksCount == 1
```

Amazon ECS コンテナエージェントバージョン

次の式は、コンテナエージェントバージョン 1.14.5 より前のバージョンを実行しているインスタンスを選択します。

```
agentVersion < 1.14.5
```

例: インスタンス登録時

次の式は、2018 年 2 月 13 日以前に登録されているインスタンスを選択します。

```
registeredAt < 2018-02-13
```

例: Amazon EC2 インスタンス ID

次の式は、以下の Amazon EC2 インスタンス ID のインスタンスを選択します。

```
ec2InstanceId in ['i-abcd1234', 'i-wxyx7890']
```

Amazon ECS タスク配置の制約事項の例

以下はタスク配置の制約事項の例です。

この例では、`memberOf` 制約を使用して T2 インスタンスにタスクを配置します。次のアクションを使用して指定できます。CreateService、UpdateService、RegisterTaskDefinition、および RunTask。

```
"placementConstraints": [
    {
        "expression": "attribute:ecs.instance-type =~ t2.*",
        "type": "memberOf"
    }
]
```

この例では、memberOf 制約を使用して、指定されたタスク配置方法に従って、daemon-service タスクグループ内のデーモンサービスタスクを持つインスタンスに、レプリカタスクを配置します。この制約により、デーモンサービスタスクはレプリカサービスタスクよりも前に EC2 インスタンスに配置されます。

daemon-service を デーモンサービスの名前に置き換えます。

```
"placementConstraints": [
    {
        "expression": "task:group == service:daemon-service",
        "type": "memberOf"
    }
]
```

この例では、memberOf 制約を使用して、指定されたタスク配置方法に従って、databases タスクグループ内の他のタスクとともにインスタンスにタスクを配置します。タスクグループの詳細については、「Amazon ECS タスクをグループ化する」を参照してください。次のアクションを使用して指定できます。CreateService、UpdateService、RegisterTaskDefinition、および RunTask。

```
"placementConstraints": [
    {
        "expression": "task:group == databases",
        "type": "memberOf"
    }
]
```

distinctInstance 制約は、グループ内の各タスクを別のインスタンスに配置します。次のアクションを使用して指定できます。CreateService、UpdateService、および RunTask。

Amazon ECS は、タスク配置におけるタスクの目的のステータスを確認します。例えば、既存のタスクの目的のステータスが STOPPED の場合 (ただし、最後のステータスが目的のステータスではない場合)、distinctInstance 配置の制約事項にもかかわらず、新しい受信タスクを同じインスタ

ンスに配置することができます。したがって、同じインスタンスで最後のステータスが RUNNING のタスクが 2 つ表示される場合があります。

```
"placementConstraints": [
    {
        "type": "distinctInstance"
    }
]
```

Amazon ECS スタンドアロンタスク

バッチプロセスなど、何らかの処理を実行した後に停止するアプリケーションがある場合、アプリケーションをタスクとして実行できます。タスクを 1 回実行する場合は、コンソール、AWS CLI、API または SDK を使用できます。

レートベース、cron ベース、または 1 回限りのスケジュールでアプリケーションを実行する必要がある場合は、EventBridge Scheduler を使用してスケジュールを作成できます。

タスクワークフロー

Amazon ECS タスク (スタンドアロンタスクまたは Amazon ECS サービス) を起動すると、タスクが作成され、最初に PROVISIONING 状態に移行されます。タスクが PROVISIONING 状態になると、Amazon ECS はタスクを配置するためのコンピューティング能力を見つける必要があるため、タスクもコンテナも存在しません。

Amazon ECS は、起動タイプまたはキャパシティプロバイダーの設定に基づいて、タスクに適したコンピューティング性能を選択します。キャパシティプロバイダーとキャパシティプロバイダー戦略は、Fargate と EC2 の両方で使用できます。Fargate を使用すると、クラスター容量のプロビジョニング、設定、スケーリングについて心配する必要はありません。Fargate は、お客様のタスクに必要なすべてのインフラストラクチャ管理を行います。EC2 の場合、Amazon EC2 インスタンスをクラスターに登録してクラスター容量を管理するか、クラスターの自動スケーリングを使用してコンピューティングキャパシティ管理を簡素化することができます。クラスター自動スケーリングは、クラスター容量を動的にスケーリングするため、ユーザーは実行中のタスクに集中できます。Amazon ECS は、タスク定義で指定した要件 (CPU やメモリなど)、および配置の制約事項と戦略に基づいて、タスクを配置する場所を決定します。詳細については、「Amazon ECS がタスクをコンテナインスタンスに配置する方法」を参照してください。

マネージドスケーリングが有効になっているキャパシティープロバイダーを使用すると、コンピューティングキャパシティーの不足が原因で開始できないタスクは、すぐに失敗するのではな

く、PROVISIONING 状態に移行されます。タスクを配置する容量が決まったら、Amazon ECS は必要なアタッチメント (awsvpc モード内のタスク用の Elastic Network Interface (ENI) など) をプロビジョニングします。Amazon ECS コンテナエージェントを使用してコンテナイメージをプルし、コンテナを起動します。プロビジョニングが完了し、関連するコンテナが起動すると、Amazon ECS はタスクを RUNNING 状態に移行します。タスクの状態の詳細については、「Amazon ECS タスクライフサイクル」を参照してください。

Amazon ECS タスクとしてのアプリケーションの実行

1 回限りのプロセス用のタスクは AWS マネジメントコンソール を使用して作成できます。

スタンドアロンタスクを作成するには (AWS マネジメントコンソール)

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
2. Amazon ECS コンソールでは、クラスターの詳細ページまたはタスク定義リビジョンリストからスタンドアロンタスクを作成できます。選択したリソースページに応じて、次の手順を使用してスタンドアロンタスクを作成します。

サービスの起動元	Steps
クラスター詳細ページ	a. [Clusters] ページで、サービスを作成するクラスターを選択します。 b. [タスク] タブで、[タスクの実行] を選択します。
タスク定義のリビジョンページ	a. [タスク定義] ページで、タスク定義ファミリを選択して、そのファミリーのリビジョンを表示します。 b. 使用するリビジョンを選択します。 c. [デプロイ] メニューから、[タスクの実行] を選択します。

3. [既存のクラスター] で、クラスターを選択します。

[クラスターの作成] を選択して、新しいクラスターでタスクを実行します。

4. クラスターのインフラストラクチャ全体にタスクを分散する方法を選択します。[コンピューティング設定] で、オプションを選択します。キャパシティプロバイダー戦略を使用するには、キャパシティプロバイダーをクラスターレベルで設定する必要があります。

   キャパシティプロバイダーを使用するようにクラスターを構成していない場合は、代わりに起動タイプを使用してください。

   Amazon ECS マネージドインスタンスでワークロードを実行する場合は、キャパシティプロバイダーの戦略オプションを使用する必要があります。

ディストリビューションの方法	Steps	
キャパシティープロバイダー戦略	a. [Compute options] (コンピューティングオプション) セクションで、[Capacity provider strategy] (キャパシティープロバイダー戦略) を選択します。 b. 戦略を選択: • デフォルトのキャパシティープロバイダー戦略を使用するには、[Use cluster default] (デフォルトのクラスターを使用) を選択します。 • クラスターにデフォルトのキャパシティープロバイダー戦略がない場合、またはカスタム戦略を使用する場合は、[Use custom] (カスタムを	

ディストリビューションの方法	Steps	
	使用)、[Add capacity provider strategy] (キャパシティプロバイダー戦略の追加)を選択し、[Base] (ベース)、[Capacity provider] (キャパシティープロバイダー)、[Weight] (ウェイト) を指定して、カスタムキャパシティープロバイダー戦略を定義します。 **Note** 戦略でキャパシティープロバイダーを使用するには、キャパシティープロバイダーをクラスターに関連付ける必要があります。	

ディストリビューションの方法	Steps	
起動タイプ	a. [Compute options] (コンピューティングオプション) セクションで、[Launch type] (起動タイプ) を選択します。 b. [Launch type] (起動タイプ) で、起動タイプを選択します。 c. (オプション) Fargate を使用する場合は、[プラットフォームバージョン] で、使用するプラットフォームのバージョンを指定します。プラットフォームバージョンが指定されない場合、LATEST プラットフォームバージョンが使用されます。	

5. [デプロイメント設定] で、次の操作を行います。

   a. [タスク定義]に、タスク定義を入力します。

   > **Important**
   > コンソールは、選択を検証し、選択したタスク定義ファミリーおよびリビジョンが、定義されたコンピューティング設定と互換性があることを確認します。

   b. [Desired tasks] (必要なタスク) で、起動するタスクの数を入力します。

   c. [タスクグループ]に、タスクグループの名前を入力します。

6. タスク定義で awsvpc ネットワークモードを使用している場合、[Networking] (ネットワーク) を展開します。カスタム設定を指定するには、次のステップを実行します。

   a. VPC の場合、使用する VPC を選択します。

b. [Subnets] (サブネット) で、タスクスケジューラがタスクを配置するときに考慮する VPC 内のサブネットを 1 つ以上選択します。

c. [セキュリティグループ] で、既存のセキュリティグループを選択することも、新しいセキュリティグループを作成することもできます。既存のセキュリティグループを使用するには、セキュリティグループを選択し、次のステップに進みます。新しいセキュリティグループを作成するには、[Create a new security group (新しいセキュリティグループの作成)] を選択します。セキュリティグループの名前、説明を指定してから、セキュリティグループのインバウンドルールを 1 つ以上追加する必要があります。

d. [Public IP] (パブリック IP) では、タスクの Elastic Network Interface (ENI) にパブリック IP アドレスを自動的に割り当てるかどうかを選択します。

   AWS Fargate タスクをパブリックサブネットで実行するときに、そのタスクにパブリック IP アドレスを割り当てて、インターネットへのルートを持つようにすることができます。このフィールドを使用して EC2 タスクにパブリック IP を割り当てることはできません。詳細については、「Fargate の Amazon ECS タスクのネットワーキングオプション」および「Amazon ECS タスクにネットワークインターフェイスを割り当てる」を参照してください。

7. デプロイ時の設定と互換性のあるデータボリュームをタスクで使用する場合は、[ボリューム] を拡張してボリュームを設定できます。

   ボリューム名とボリュームタイプはタスク定義リビジョンの作成時に設定され、スタンドアロンタスクの実行時には変更できません。ボリューム名とタイプを更新するには、新しいタスク定義リビジョンを作成し、その新しいリビジョンを使用してタスクを実行する必要があります。

このボリュームタイプを設定するには	この操作を行います	
Amazon EBS	a. [EBS ボリュームタイプ] には、タスクにアタッチする EBS ボリュームのタイプを選択します。 b. [サイズ (GiB)] には、ボリュームサイズの有効な値をギビバイト (GiB) 単位で入力します。最小で 1 GiB、最大で 16,384	

このボリュームタイプを設定するには	この操作を行います	
	GiB のボリュームサイズを指定できます。スナップショット ID を指定しない限り、この値は必須です。 c. [IOPS] には、ボリュームが提供する入力/出力オペレーションの数 (IOPS) の最大数を入力します。この値は io1、io2、および gp3 ボリュームタイプでのみ設定できます。 d. [スループット (MiB/秒)] には、ボリュームが提供すべきスループットをメビバイト/秒 (MiBps または MiB/s) で入力します。この値は gp3 ボリュームタイプでのみ設定できます。 e. [スナップショット ID] には、既存の Amazon EBS ボリュームスナップショットを選択するか、スナップショットからボリュームを作成する場合はスナップショットの ARN を入力します。新しい空のボリュームは、スナップショット ID を選択または入力しないで作成することもできます。	

このボリュームタイプを設定するには	この操作を行います	
	f. スナップショット ID を指定すると、ボリューム初期化レート (MiB/秒) を指定できます。100 から 300 の値 (MiB/秒単位) を入力して、スナップショット ID を使用してボリュームを作成する際のスナップショットからのデータ読込み速度を指定します。 g. タスクの終了後にタスクに接続するように構成されたボリュームを保持する場合は、[終了ポリシー] のチェックボックスを選択解除します。デフォルトでは、タスクにアタッチされている EBS ボリュームは、タスクが終了すると削除されます。 h. [ファイルシステムのタイプ] では、ボリューム上のデータストレージおよび取得に使用するファイルシステムの種類を選択します。オペレーティングシステムのデフォルトまたは特定のファイルシステムタイプのいずれかを選択できます。Linux のデフォルトは XFS です。スナップショットから作成	

このボリュームタイプを設定するには	この操作を行います	
	されたボリュームでは、スナップショットの作成時にボリュームが使用していたのと同じファイルシステムタイプを指定する必要があります。ファイルシステムのタイプが一致しない場合、タスクは開始できません。 i. [インフラストラクチャロール] には、Amazon ECS がタスクの Amazon EBS ボリュームを管理できるようにするために必要な権限を持つ IAM ロールを選択します。`AmazonECSInfrastructureRolePolicyForVolumes` 管理ポリシーをロールにアタッチすることも、このポリシーをガイドとして使用して、特定のニーズを満たすアクセス権限を持つ独自のポリシーを作成してアタッチすることもできます。必要なアクセス権限の詳細については、「Amazon ECS インフラストラクチャ IAM ロール」を参照してください。	

このボリュームタイプを設定するには	この操作を行います
	j. デフォルト設定で Amazon EBS 暗号化を使用する場合は、[暗号化] で [デフォルト] を選択します。アカウントにデフォルトで暗号化が設定されている場合、ボリュームは設定で指定されている AWS Key Management Service (AWS KMS) キーで暗号化されます。[デフォルト] を選択し、Amazon EBS のデフォルト暗号化がオンになっていない場合、ボリュームは暗号化されません。 [カスタム] を選択した場合は、ボリューム暗号化に任意の AWS KMS key を指定できます。 [なし] を選択すると、デフォルトで暗号化が設定されている場合や、暗号化されたスナップショットからボリュームを作成した場合を除き、ボリュームは暗号化されません。 k. [暗号化] に [カスタム] を選択した場合は、使用する AWS KMS key を指定する必要があり

このボリュームタイプを設定するには	この操作を行います	
	ます。[KMS キー] には、AWS KMS key を選択するか、キーの ARN を入力します。対称型のカスタマー管理キーを使用してボリュームを暗号化する場合は、AWS KMS key ポリシーに適切な権限が定義されていることを確認してください。詳細については、「Amazon EBS ボリュームのデータ暗号化」を参照してください。 l. (オプション) [タグ] では、タスク定義からタグを伝達するか、独自のタグを指定することで、Amazon EBS ボリュームにタグを追加できます。 タスク定義からタグを伝達する場合は、[タグの伝播元] で [タスク定義] を選択します。[伝播しない] を選択した場合、または値を選択しなかった場合、タグは伝播されません。 独自のタグを指定する場合は、[タグの追加] を選択し、追加する各タグのキーと値を指定します。	

このボリュームタイプを設定するには	この操作を行います	
	Amazon EBS ボリュームのタグ付けの詳細については、「Amazon EBS ボリュームのタグ付け」を参照してください。	

8. (オプション) デフォルト以外のタスク配置戦略を使用するには、[Task Placement] (タスクの配置) を展開し、以下のオプションから選択します。

   詳細については、「Amazon ECS がタスクをコンテナインスタンスに配置する方法」を参照してください。

   - [AZ バランススプレッド] - アベイラビリティーゾーン間およびアベイラビリティーゾーン内のコンテナインスタンス間でタスクを分散します。
   - [AZ Balanced BinPack] - 利用可能な最小メモリでアベイラビリティーゾーン間およびコンテナインスタンス間でタスクを分散します。
   - [ビンパック] - CPU またはメモリの最小利用可能量に基づいてタスクを配置します。
   - [ホストごとに 1 つのタスク] - 各コンテナインスタンスのサービスから最大 1 タスクを配置します。
   - [カスタム] - 独自のタスク配置戦略を定義します。

   [Custom] (カスタム) を選択した場合、タスクを配置するアルゴリズムと、タスク配置時に考慮されるルールを定義します。

   - [Strategy] (方針) にとって [Type] (タイプ) そして [Field] (フィールド) で、アルゴリズムとアルゴリズムに使用するエンティティを選択します。

     最大 5 個の戦略を入力できます。
   - [制約] の [タイプ] および [式] で、制約のルールと属性を選択します。

     例えば、T2 インスタンスにタスクを配置する制約を設定するには、[Expression] (表現) で、[attribute:ecs.instance-type =~ t2.*] と入力します。

     最大 10 個の制約を入力できます。

9. (オプション) タスク IAM ロール、またはタスク定義で定義されているタスク実行ロールをオーバーライドするには、[Task overrides] (タスクのの上書き) を展開し、以下のステップを実行します。

   a. [タスクロール] で、このタスクの IAM ロールを選択します。詳細については、「Amazon ECS タスクの IAM ロール」を参照してください。

      `ecs-tasks.amazonaws.com`信頼関係を持つロールのみが表示されます。タスクの IAM ロールを作成する方法については、「タスクの IAM ロールを作成する」を参照してください。

   b. [タスク実行ロール] で、タスク実行ロールを選択します。詳細については、「Amazon ECS タスク実行IAM ロール」を参照してください。

10. (オプション) コンテナコマンドと環境変数をオーバーライドするには、[Container Overrides] (コンテナの上書き) を展開し、コンテナを展開します。

    - タスク定義コマンド以外のコンテナにコマンドを送信するには、[コマンドの上書き] で Docker コマンドを入力します。
    - [Add Environment Variable] (環境変数の追加) で、環境変数を追加します。Keyに、環境変数の名前を入力します。Value(値) で、(文字列を囲む二重引用符 (" ") なしで) 環境値の文字列値を入力します。

      AWS は文字列を二重引用符 (" ") で囲み、次の形式で文字列をコンテナに渡します。

      ```
      MY_ENV_VAR="This variable contains a string."
      ```

11. (オプション) タスクを識別しやすくするには、[Tags] (タグ) セクションを展開し、タグを設定します。

    Amazon ECS で、新しく起動されたすべてのタスクに、クラスター名とタスク定義のタグで自動でタグ付けするには、[Turn on Amazon ECS managed tags] (Amazon ECS で管理されたタグを有効にする) を選択し、[Task definition] (タスク定義) を選択します。

    タグを追加または削除します。

    - [タグを追加] [Add tag] (タグを追加) を選択し、以下を実行します。
      - [キー] にはキー名を入力します。
      - [値] にキー値を入力します。
    - [タグの削除] タグの横にある [タグの削除] を選択します。

12. [作成] を選択します。

Amazon EventBridge スケジューラを使用して Amazon ECS タスクをスケジュールする

EventBridge スケジューラーはサーバーレススケジューラであり、一元化されたマネージドサービスからタスクを作成、実行、管理できます。イベントバスやルールに依存しない、1 回限りの定期的なスケジューリング機能を提供します。EventBridge スケジューラは高度にカスタマイズ可能で、EventBridge のスケジュールされたルールよりもスケーラビリティが高く、より幅広いターゲット API オペレーションと AWS のサービスを使用できます。EventBridge スケジューラには、EventBridge スケジューラコンソールでタスクに設定できる以下のスケジュールが用意されています。

- レートベース
- cron ベース

  cron ベースのスケジュールはどのタイムゾーンでも設定できます。
- 1 回限りのスケジュール

  1 回限りのスケジュールはどのタイムゾーンでも設定できます。

Amazon ECS は Amazon EventBridge スケジューラを使用してスケジュールできます。

Amazon ECS コンソールでスケジュールされたタスクを作成できますが、現在、EventBridge Scheduler コンソールではさらに多くの機能を提供しています。

タスクをスケジュールする前に、次のステップを完了します。

1. タスクが実行されるサブネット ID とサブネットのセキュリティグループ ID を取得するには、VPC コンソールを使用します。詳細については、「VPC のサブネット」と「Amazon VPC ユーザーガイド」の「セキュリティグループを使用して AWS リソースへのトラフィックを制御する」を参照してください。
2. EventBridge スケジューラの実行ロールを設定します。詳細については、「Amazon EventBridge スケジューラユーザーガイド」の「実行ロールを設定する」を参照してください。
3. キャパシティープロバイダー戦略を使用してタスクを実行するには、キャパシティープロバイダーをクラスターに関連付ける必要があります。

コンソールを使用して新しいスケジュールを作成するには

1. Amazon EventBridge スケジューラコンソール (https://console.aws.amazon.com/scheduler/home) を開きます。
2. [スケジュール] ページで、[スケジュールを作成] を選択します。
3. [スケジュールの詳細を指定] ページの [スケジュールの名前と説明] セクションで、次を実行します。

   a. [スケジュール名] で、スケジュールの名前を入力します。例えば、**MyTestSchedule**。

   b. (オプション) [説明] で、スケジュールの説明を入力します。例えば、**TestSchedule**。

   c. [スケジュールグループ] で、スケジュールグループを選択します。グループがない場合は、[デフォルト] を選択します。スケジュールグループを作成するには、[独自のスケジュールを作成] を選択します。

      スケジュールグループを使用して、スケジュールのグループにタグを追加します。
4. スケジュールオプションを選択します。

頻度	手順
回限りのスケジュール 1 回限りのスケジュールは、指定した日時に 1 回だけターゲットを呼び出します。	[日付と時刻] で、次を実行します。 • 有効な日付を YYYY/MM/DD 形式で入力します。 • タイムスタンプを 24 時間 (hh:mm) 形式で入力します。 • [タイムゾーン] で、タイムゾーンを選択します。
繰り返しのスケジュール 繰り返しのスケジュールは、cron 式またはレート式を使用して指定したレートでターゲットを呼び出します。	a. [スケジュールの種類] では、次のいずれかを実行します。 • Cron 式を使用してスケジュールを定義するには、[cron ベースのスケ

頻度	手順	
	ジュール] を選択して Cron 式を入力します。 • Rate 式を使用してスケジュールを定義するには、[rate ベースのスケジュール] を選択して Rate 式を入力します。 cron および rate 式の詳細については、「Amazon EventBridge スケジューラユーザーガイド」の「EventBridge スケジューラのスケジュールタイプ」を参照してください。 b. [フレックスタイムウィンドウ] で、[オフ] を選択してオプションをオフにするか、事前定義された時間枠のいずれかを選択します。例えば、[15 分] を選択し、1 時間に 1 回ターゲットを呼び出す繰り返しのスケジュールを設定した場合、スケジュールは毎時の開始後 15 分以内に実行されます。	

5. (オプション) 前のステップで [定期的なスケジュール] を選択した場合は、[時間枠] セクションで次を実行します。

   a. [タイムゾーン] で、タイムゾーンを選択します。

b. [開始日時] で、有効な日付を YYYY/MM/DD 形式で入力してから、タイムスタンプを 24 時間 (hh:mm) 形式で指定します。

c. [終了日時] で、有効な日付を YYYY/MM/DD 形式で入力してから、タイムスタンプを 24 時間 (hh:mm) 形式で指定します。

6. [次へ] を選択します。

7. [ターゲットを選択] ページで、次の操作を行います。

a. [すべての API] を選択し、検索ボックスで ECS と入力します。

b. [Amazon ECS] を選択します。

c. 検索ボックスで RunTask と入力し、[RunTask] を選択します。

d. [ECS クラスター] で、クラスターを選択します。

e. [ECS タスク] で、タスクに使用するタスク定義を選択します。

f. クラスターのインフラストラクチャ全体にタスクを分散する方法を選択します。コンピューティングオプション を展開し、次のいずれかのオプションを選択します

コンピューティングオプション	Steps	
キャパシティープロバイダー戦略	a. [キャパシティープロバイダー戦略] を選択します。 b. 戦略を選択: • デフォルトのキャパシティープロバイダー戦略を使用するには、[デフォルトのクラスターを使用] を選択します。 • カスタム戦略を使用するには、[カスタムを使用] を選択します。次に、ベース、キャパシティープロバイ	

コンピューティングオプション	Steps	
	ダー、およびウェイトを入力します。 EC2 の場合、キャパシティープロバイダーは Auto Scaling グループです。	
起動タイプ	a. [起動タイプ] を選択します。 b. [Launch type] (起動タイプ) で、起動タイプを選択します。 c. Fargate が指定されている場合は、[プラットフォームバージョン] で、使用するプラットフォームのバージョンを指定します。	

g. [サブネット] で、タスクを実行するサブネット ID を入力します。

h. [セキュリティグループ] で、サブネットのセキュリティグループ ID を入力します。

i. (オプション) デフォルト以外のタスク配置戦略を使用するには、[配置制約] を展開し、制約を入力します。

詳細については、「Amazon ECS がタスクをコンテナインスタンスに配置する方法」を参照してください。

j. (オプション) タスクを識別しやすくするために、[タグ] でタグを設定します。

新しく起動されたすべてのタスクに対して、Amazon ECS がタスク定義タグを自動的にタグ付けするようにするには、[Amazon ECS マネージドタグを有効にする] を選択します。

8. [次へ] を選択します。

9. [Settings] (設定) ページで、以下の操作を行います。

a. スケジュールをオンにするには、[スケジュールの状態] で [スケジュールを有効にする] をオンに切り替えます。

b. スケジュールの再試行ポリシーを設定するには、[再試行ポリシーとデッドレターキュー (DLQ)] で次を実行します。

- [再試行] を切り替えてオンにします。
- [イベントの最大保持時間] で、EventBridge スケジューラが未処理のイベントを保持しなければならない最大の [時間] と [分] を入力します。
- 最大 24 時間です。
- [最大再試行回数] で、ターゲットがエラーを返した場合に EventBridge スケジューラがスケジュールを再試行する最大回数を入力します。

  再試行の最大値は 185 です。

再試行ポリシーを使用すると、スケジュールがそのターゲットの呼び出しに失敗した場合、EventBridge スケジューラはスケジュールを再実行します。設定されている場合は、スケジュールの最大保持時間と再試行を設定する必要があります。

c. EventBridge スケジューラが未配信のイベントを保存する場所を選択します。

[デッドレターキュー (DLQ)] オプション	手順	
保存しない	[None] を選択します。	
スケジュールを作成しようとしている同じ AWS アカウント にイベントを保存する	a. [自分の AWS アカウント の Amazon SQS キューを DLQ として選択] を選択します。 b. Amazon SQS キューの Amazon リソースネーム (ARN) を選択します。	

[デッドレターキュー (DLQ)] オプション	手順
スケジュールを作成しようとしているのとは別の AWS アカウント にイベントを保存する	a. [他の AWS アカウント の Amazon SQS キューを DLQ として指定] を選択します。 b. Amazon SQS キューの Amazon リソースネーム (ARN) を入力します。

d. カスタマーマネージドキーを使用してターゲットの入力を暗号化するには、[暗号化] で [暗号化設定をカスタマイズする (高度)] を選択します。

   このオプションを選択した場合は、既存の KMS キー ARN を入力するか、[AWS KMS key を作成] を選択して AWS KMS コンソールに移動します。EventBridge スケジューラが保管中のデータを暗号化する方法の詳細については、「Amazon EventBridge スケジューラユーザーガイド」の「保管中の暗号化」を参照してください。

e. [許可] で、[既存のロールを使用] を選択してから、ロールを選択します。

   EventBridge スケジューラに新しい実行ロールを作成させるには、[このスケジュールの新しいロールを作成] を選択します。その後、[ロール名] で名前を入力します。このオプションを選択すると、EventBridge スケジューラは、テンプレート化されたターゲットに必要な許可をロールにアタッチします。

10. [次へ] を選択します。
11. [スケジュールの確認と作成] ページで、スケジュールの詳細を確認します。各セクションで、そのステップに戻って詳細を編集するには、[編集] を選択します。
12. [スケジュールを作成] を選択します。

    [スケジュール] ページで、新規および既存のスケジュールのリストを表示できます。[ステータス] 列で、新しいスケジュールが [有効] になっていることを確認します。

次のステップ

EventBridge スケジューラコンソールまたは AWS CLI を使用し、スケジュールを管理できます。詳細については、「Amazon EventBridge スケジューラユーザーガイド」の「スケジュールの管理」を参照してください。

Amazon ECS タスクの停止

スタンドアロンタスクを実行し続ける必要がなくなった場合は、タスクを停止できます。Amazon ECS コンソールでは、1 つ以上のタスクを簡単に停止できます。

スタンドアロンの停止したタスクは、再起動できません。

サービスを停止したい場合は、「コンソールを使用して Amazon ECS サービスの削除」を参照してください。

スタンドアロンタスクを停止するには (AWS マネジメントコンソール)

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。
3. [クラスター] ページで、クラスターを選択してクラスターの詳細ページに移動します。
4. クラスターの詳細ページで、[タスク] タブを選択します。
5. [起動タイプのフィルター] リストを使用して、起動タイプごとにタスクをフィルターできます。

停止するタスク	Steps
1 つ以上	a. タスクを選択し、[停止]、[選択したものを停止] を選択します。 b. [タスク停止確認ページ] で [停止] を選択します。
すべて	**Important** コンソールを使用してすべてのタスクを停止することを選択した場合、Amazon ECS はすべてのスタンドアロンタスクとサービスの一部であるタスクを停止します。したがって、こ

停止するタスク	Steps
	のオプションを使用する場合は注意が必要です。 a. [停止]、[すべてを停止] の順に選択します。 b. [タスク停止確認ページ] で、[すべてのタスクを停止] を入力してから、[停止] を選択します。

Amazon ECS サービス

Amazon ECS サービスを使用すると、Amazon ECS クラスター内で、タスク定義の指定した数のインスタンスを同時に実行して維持できます。タスクの 1 つが失敗または停止した場合、Amazon ECS サービススケジューラはタスク定義の別のインスタンスを起動してそれを置き換えます。これは、サービスで必要な数のタスクを維持するのに役立ちます。

オプションで、ロードバランサーの背後でサービスを実行することもできます。ロードバランサーは、サービスに関連付けられたタスク間でトラフィックを分散させます。

長時間実行されるステートレスサービスおよびアプリケーションには、サービススケジューラを使用することをお勧めします。サービススケジューラにより、指定したスケジューリング戦略が順守され、タスクが失敗したときに タスクが再スケジュールされます。例えば、基盤となるインフラストラクチャに障害が発生した場合、サービススケジューラはタスクを再スケジュールします。タスク配置の戦略と制約を使用して、スケジューラがタスクを配置および終了する方法をカスタマイズできます。サービス内のタスクが停止すると、スケジューラはタスクを置き換えるために新しいタスクを起動します。このプロセスは、サービスが使用するスケジュール戦略に基づいて、サービスがタスクの必要数に達するまで続行されます。

また、コンテナのヘルスチェックまたはロードバランサーのターゲットグループのヘルスチェックが失敗すると、サービススケジューラーによって、異常であると判断されたタスクが置き換えられます。この置き換え動作は、`maximumPercent` および `desiredCount` のサービス定義パラメータに

よって異なります。タスクが異常とマークされた場合、サービススケジューラーによってまず置き換えタスクが開始されます。次に以下が発生します。

- 置き換えタスクのヘルスステータスが HEALTHY になると、サービススケジューラーが異常のあるタスクを停止します。
- 置き換えタスクのヘルスステータスが UNHEALTHY の場合、スケジューラーは異常のある置き換えタスクまたは既存の異常タスクのいずれかを停止して、タスク総数が desiredCount と等しくなるようにします。

maximumPercent パラメーターによって、置き換えタスクを先に開始できないようにスケジューラーが制限されている場合、スケジューラーは異常のあるタスクをランダムに 1 つずつ停止して容量を解放してから置き換えタスクを開始します。異常のあるタスクがすべて正常なタスクに置き換えられるまで、起動と停止のプロセスが続きます。異常なタスクがすべて置き換えられ、正常なタスクだけが実行中になると、合計タスク数が desiredCount を超える場合、タスク数が desiredCount になるまで、正常なタスクが無作為に停止されます。maximumPercent および desiredCount の詳細については、「サービス定義パラメータ」を参照してください。

サービススケジューラには、タスクが繰り返し起動に失敗した場合にタスクを再起動する頻度を調整するロジックが含まれています。RUNNING 状態にならずにタスクが停止すると、サービススケジューラは起動の試行の速度を遅くし始め、サービスイベントメッセージを送信します。この動作により、問題を解決する前に、失敗したタスクに不要なリソースが使用されるのを防ぐことができます。サービスが更新されると、サービススケジューラは正常なスケジューリング動作を再開します。詳細については、「Amazon ECS サービスのスロットルロジック」および「Amazon ECS のサービスイベントメッセージを表示する」を参照してください。

インフラストラクチャコンピューティングオプション

タスクを分散するコンピューティングオプションが 2 つあります。

- [capacity provider strategy] (キャパシティープロバイダー戦略) により、Amazon ECS がタスクを 1 つまたは複数のキャパシティプロバイダーに分散させます。

  Amazon ECS マネージドインスタンスでワークロードを実行する場合は、キャパシティプロバイダーの戦略オプションを使用する必要があります。

  [capacity provider strategy] (キャパシティープロバイダー戦略) では、コンソールはデフォルトでコンピューティングオプションを選択します。次に、コンソールがデフォルトを選択するときに使用する順序について説明します。

- クラスターにデフォルトのキャパシティープロバイダー戦略が定義されている場合は、その戦略が選択されます。
- クラスターにデフォルトのキャパシティープロバイダー戦略が定義されていないが、Fargate キャパシティープロバイダーをクラスターに追加している場合は、キャパシティープロバイダーを使用するカスタム FARGATE キャパシティープロバイダー戦略が選択されます。
- クラスターにデフォルトのキャパシティープロバイダー戦略が定義されていないが、クラスターに 1 つ以上の Auto Scaling グループキャパシティープロバイダーが追加されている場合は、[Use custom (アドバンスト)] オプションが選択され、戦略を手動で定義する必要があります。
- クラスターにデフォルトのキャパシティープロバイダー戦略が定義されておらず、クラスターにキャパシティープロバイダーが追加されていない場合は、Fargate起動タイプが選択されます。

- [起動タイプ] により、Amazon ECS は Fargate またはクラスターに登録された EC2 インスタンスのいずれかでタスクを直接起動します。

  Amazon ECS マネージドインスタンスでワークロードを実行する場合は、キャパシティプロバイダーの戦略オプションを使用する必要があります。

  デフォルトでは、サービスはクラスター VPC のサブネットで開始されます。

サービスのオートスケーリング

サービスの自動スケーリングは、Amazon ECS サービスで必要なタスク数を自動的に増減する機能です。Amazon ECS は アプリケーション Auto Scaling サービスを活用してこの機能を提供します。

詳細については、「Amazon ECS サービスを自動的にスケールする」を参照してください。

サービスの負荷分散

AWS Fargate でホストされている Amazon ECS サービスでは、Application Load Balancer、Network Load Balancer、および Gateway Load Balancer がサポートされています。次の表を使用して、使用するロードバランサーのタイプを確認してください。

ロードバランサーのタイプ	以下の場合に使用	
Application Load Balancer	HTTP/HTTPS (またはレイヤー 7) トラフィックをルーティングします。	

ロードバランサーのタイプ	以下の場合に使用	
	Amazon Load Balancer は Amazon ECS サービスでの使用に便利な複数の機能を提供しています。 • 各サービスは、複数のロードバランサーからトラフィックを送信し、複数のターゲットグループを指定することにより複数のロードバランシングポートを公開できます。 • バインドマウントは、Fargate インスタンスと Amazon EC2 インスタンスの両方でホストされているタスクでサポートされています。 • Application Load Balancer により、コンテナが動的ホストポートマッピングを使用できるようになります (同じサービスから複数のタスクがコンテナインスタンスごとに許可するため)。 • Application Load Balancer では、パスベースのルーティングと優先ルールをサポートしています (複数のサービスが 1 つの Application Load Balancer で同じリスナーポートを使用するため)。	

ロードバランサーのタイプ	以下の場合に使用
Network Load Balancer	TCP または UDP (またはレイヤー 4) トラフィックをルーティングします。
Gateway Load Balancer	TCP または UDP (またはレイヤー 4) トラフィックをルーティングします。 ファイアウォール、侵入検知および防止システム、ディープパケットインスペクションシステムなどの仮想アプライアンスを使用します。

詳細については、「ロードバランサーを使用して Amazon ECS サービストラフィックを分散する」を参照してください。

相互接続サービス

Amazon ECS サービスとして実行される他のアプリケーションにアプリケーションを接続する必要がある場合、Amazon ECS ではロードバランサーを使用せずに、次の方法で実行できます。

- Service Connect – 短い名前と標準ポートを使用した自動検出によるサービス間の通信を実現します。
- サービス検出 – サービス検出では、Route 53 を使用してサービスの名前空間を作成し、DNS を介して検出できるようにします。
- Amazon VPC Lattice – VPC Lattice は、複数のアカウントおよび VPC 間でサービスを接続、保護、モニタリングするフルマネージドのアプリケーションネットワーキングサービスです。関連するコストが発生します。

詳細については、「Amazon ECS サービスを相互接続する」を参照してください。

Amazon ECS サービスデプロイコントローラーと戦略

サービスをデプロイする前に、デプロイするためのオプションとサービスが使用する機能を確認してください。

スケジュール戦略

利用できる 2 つのサービススケジューラ戦略があります。

- REPLICA - レプリカスケジュール戦略では、クラスター全体で必要数のタスクを配置して維持します。デフォルトでは、サービススケジューラによってタスクはアベイラビリティーゾーン間に分散されます。タスク配置の戦略と制約を使用すると、タスク配置の決定をカスタマイズできます。詳細については、「レプリカスケジューリング戦略」を参照してください。
- DAEMON - デーモンのスケジュール戦略では、指定したすべてのタスク配置制約を満たすクラスター内のアクティブなコンテナインスタンスごとに、1 つのタスクのみをデプロイします。この戦略を使用する場合、タスクの必要数や配置戦略、サービスの自動スケーリングポリシーを指定する必要はありません。詳細については、「デーモンスケジューリング戦略」を参照してください。

> **Note**
>
> Fargate タスクは DAEMON スケジュール戦略をサポートしていません。

レプリカスケジューリング戦略

レプリカスケジュール戦略では、クラスターに必要数のタスクを配置して維持します。

Fargate でタスクを実行するサービスについては、サービススケジューラは、新しいタスクの起動時や実行中タスクの停止時に、アベイラビリティーゾーン間の負荷バランスを維持するよう最大限試みます。タスク置き換え戦略や制約を指定する必要はありません。

EC2 インスタンスでタスクを実行するサービスを作成する場合、オプションでタスク配置戦略と制約を指定して、タスク配置に関する決定をカスタマイズできます。タスク配置戦略または制約が指定されていない場合、デフォルトでは、サービススケジューラはタスクをアベイラビリティーゾーン全体に分散します。サービススケジューラは、次のロジックを使用します。

- クラスター内のどのコンテナインスタンスがサービスのタスク定義をサポートできるか (必要な CPU、メモリ、ポート、コンテナインスタンス属性がなど) を判断します。
- サービスに定義された配置の制約を満たすコンテナインスタンスを特定します。

- デーモンサービスに依存するレプリカサービス (たとえば、タスクがログを使用する前に実行する必要があるデーモンログルータスクなど) がある場合は、デーモンサービスタスクがレプリカサービスタスクよりも前に EC2 インスタンスに配置されるようにするタスク置き換え制約を作成します。詳細については、「Amazon ECS タスク配置の制約事項の例」を参照してください。
- 配置戦略が定義されている場合は、その戦略を使用して残りの候補からインスタンスを選択します。
- 定義された配置戦略がない場合は、次のロジックを使用してクラスター内のアベイラビリティーゾーン全体にタスクが配分されます。
  - 有効なコンテナインスタンスをソートします。それぞれのアベイラビリティーゾーンで、このサービスの実行中のタスクの数が最も少ないインスタンスを優先します。例えば、ゾーン A に実行中のサービスタスクが 1 つあり、ゾーン B と C に実行中のサービスタスクがない場合、ゾーン B または C のいずれかの有効なコンテナインスタンスが配置に最適と見なされます。
  - 前のステップに基づいて、新しいサービスタスクを最適なアベイラビリティーゾーン内の有効なコンテナインスタンスに配置します。このサービスで実行中のタスク数が最も少ないコンテナインスタンスを優先します。

サービスで高い利用可能性を確保できるようになるため、サービス再調整機能は `REPLICA` 戦略を使用する際に使用することをお勧めします。

デーモンスケジューリング戦略

デーモンのスケジュール戦略では、指定したすべてのタスク配置制約を満たすクラスター内のアクティブなコンテナインスタンスごとに、1 つのタスクのみをデプロイします。サービススケジューラは、実行中のタスクのタスク配置の制約事項を評価し、配置制約を満たさないタスクを停止します。この戦略を使用する場合、タスクの必要数や配置戦略を指定したり、サービス自動スケーリングポリシーを使用したりする必要はありません。

Amazon ECS は、CPU、メモリ、およびネットワークインターフェイスを含むコンテナインスタンスのコンピューティングリソースをデーモンタスク用に予約します。他のレプリカサービスを使用してクラスターでデーモンサービスを起動すると、Amazon ECS はデーモンタスクを優先します。これは、デーモンタスクがインスタンス上で起動する最初のタスクであり、すべてのレプリカタスクが停止した後に停止する最後のタスクであることを意味します。この戦略により、保留中のレプリカ・タスクでリソースが使用されず、デーモンタスクでリソースを使用できるようになります。

デーモンサービススケジューラは `DRAINING` ステータスのインスタンスにはタスクを配置しません。コンテナインスタンスが `DRAINING` ステータスに移行すると、そのインスタンス上のデーモンタスクは停止します。サービススケジューラはまた、新しいコンテナインスタンスがいつクラスター

に追加されるかをモニタリングし、追加されたらそれらのインスタンスにデーモンタスクを追加します。

デプロイメント設定を指定する場合、`maximumPercent` パラメータの値は `100` (パーセンテージとして指定) である必要があります。これは、設定されていない場合に使用されるデフォルト値です。`minimumHealthyPercent` パラメータのデフォルト値は `0` (パーセンテージで指定) です。

デーモンサービスの配置制約を変更するには、サービスを再起動する必要があります。Amazon ECS は、デーモンタスクの対象となるインスタンスに予約されているリソースを動的に更新します。既存のインスタンスの場合、スケジューラはインスタンスにタスクを配置しようとします。

タスク定義でタスクサイズまたはコンテナリソースの予約が変更されると、新しいデプロイが開始されます。新しいデプロイは、サービスを更新したり、タスク定義の異なるリビジョンを設定したりするときにも開始されます。Amazon ECS は、デーモンの更新された CPU およびメモリ予約をピックアップし、デーモンのタスクでその容量をブロックします。

上記のいずれかの場合に十分なリソースがない場合、次のことが起こります。

- タスクの配置が失敗します。
- CloudWatch イベントが生成されます。
- Amazon ECS は、リソースが使用可能になるのを待って、インスタンスでタスクのスケジュールを試行します。
- Amazon ECS は、配置制約基準を満たさなくなったリザーブドインスタンスを解放し、対応するデーモンタスクを停止します。

デーモンのスケジューリング戦略は、次の場合に使用できます。

- アプリケーションコンテナの実行
- ログ記録、モニタリング、トレースタスク用のサポートコンテナの実行

Fargate 起動タイプ、`CODE_DEPLOY` または `EXTERNAL` のデプロイメントコントローラータイプを使用するタスクは、デーモンスケジューリング戦略をサポートしません。

サービススケジューラがタスクの実行を停止する場合、アベイラビリティーゾーン間のクラスターの負荷バランスを維持します。スケジューラは、次のロジックを使用します。

- 配置戦略が定義されている場合、その戦略を使用して終了するタスクを選択します。例えば、サービスにアベイラビリティーゾーンの spread 戦略が定義されている場合、1 つのタスクが選択され、残りのタスクは最適に分散されたまま残ります。
- 配置戦略が定義されていない場合は、次のロジックを使用してクラスター内のアベイラビリティーゾーン全体への配分を維持します。
  - 有効なコンテナインスタンスをソートします。それぞれのアベイラビリティーゾーンで、このサービスの実行中のタスクの数が最も多いインスタンスを優先します。例えば、実行中のサービスタスクがゾーン A には 1 つ、ゾーン B とゾーン C にはそれぞれ 2 つずつある場合、ゾーン B またはゾーン C のいずれかのコンテナインスタンスが終了に最適と見なされます。
  - 前のステップに基づいて、最適なアベイラビリティーゾーン内のコンテナインスタンスで、タスクを停止します。このサービスで実行中のタスク数が最も多いコンテナインスタンスを優先します。

デプロイコントローラー

デプロイコントローラーは、サービスにタスクがデプロイされる方法を決定するメカニズムです。有効なオプションは次のとおりです。

- ECS

  ECS デプロイコントローラーを使用するサービスを作成する際、次のデプロイ戦略から選択できます。

  - ROLLING: ローリング更新 (ROLLING) デプロイ戦略を使用するサービスを作成すると、Amazon ECS サービススケジューラによって現在実行中のタスクが新しいタスクに置き換えられます。ローリング更新中にサービスに対して Amazon ECS が追加または削除するタスク数は、サービスデプロイ設定により制御されます。

    ローリング更新のデプロイは、次のシナリオに最適です。

    - サービスの段階的な更新: サービス全体を一度にオフラインにせず、サービスを段階的に更新する必要がある場合。
    - 制限のあるリソース要件: 2 つの完全な環境を同時に実行することで生じる追加のリソースコストを回避したい場合 (ブルー/グリーンデプロイで必要)。
    - 許容されるデプロイ時間: ローリング更新はタスクを 1 つずつ置き換えるため、アプリケーションがより長いデプロイプロセスを許容できる場合。
    - インスタントロールバックが不要: サービスで数秒ではなく数分かかるロールバックプロセスを許容できる場合。

- シンプルなデプロイプロセス: 複数の環境、ターゲットグループ、リスナーを管理する複雑さのないシンプルなデプロイアプローチを求める場合。
- ロードバランサーの要件なし: サービスにロードバランサー、Application Load Balancer、Network Load Balancer、Service Connect (ブルー/グリーンデプロイに必要) を使用しない場合または必要としない場合。
- ステートフルアプリケーション: アプリケーションで、2 つの並列環境の実行を困難にする状態が維持される場合。
- コストに敏感: デプロイ中に重複する環境を実行しないことで、デプロイコストを最小限に抑えたい場合。

ローリング更新はサービスのデフォルトのデプロイ戦略であり、多くの一般的なアプリケーションシナリオでデプロイの安全性とリソース効率のバランスを実現します。

- BLUE_GREEN: ブルー/グリーンデプロイ戦略 (BLUE_GREEN) はブルーおよびグリーンという 2 つの同一の本番環境を実行することで、ダウンタイムおよびリスクを軽減するリリース方法です。Amazon ECS ブルー/グリーンデプロイを使用すると、本番トラフィックをルーティングする前に、新しいサービスリビジョンを検証できます。このアプローチによって変更をデプロイするより安全な方法が実現され、必要に応じてすばやくロールバックできます。

  Amazon ECS ブルー/グリーンデプロイは次のシナリオに最適です。

  - サービス検証: 本番トラフィックをルーティングする前に、新しいサービスリビジョンを検証する必要がある場合
  - ダウンタイムなし: サービスにダウンタイムのないデプロイが必要な場合
  - インスタントロールバック: 問題が検出された場合、すばやくロールバックする機能が必要な場合
  - ロードバランサーの要件: サービスが Application Load Balancer、Network Load Balancer、Service Connect を使用する場合
- LINEAR: リニアデプロイ戦略 (LINEAR) は、トラフィックを現在の本番稼働環境から新しい環境に、指定した期間にわたって同じ割合で徐々に移行します。Amazon ECS リニアデプロイでは、トラフィックの移行のペースを制御し、本番稼働トラフィックの量が増えるにつれて新しいサービスリビジョンを検証できます。

  Amazon ECS リニアデプロイは次のシナリオに最適です。

  - 段階的な検証: トラフィックの増加に伴って新しいサービスバージョンを段階的に検証する必要がある場合

  - パフォーマンスモニタリング: デプロイ中にメトリクスとパフォーマンスをモニタリングするための時間が必要な場合
  - リスクの最小化: 新しいバージョンを本番稼働トラフィックに段階的に公開してリスクを最小限に抑える必要がある場合
  - ロードバランサーの要件: サービスが Application Load Balancer、Network Load Balancer、Service Connect を使用する場合
- CANARY: カナリアデプロイ戦略 (CANARY) では、まずトラフィックのごく一部を新しいサービスリビジョンに移行し、指定した期間が経過した後に残りのトラフィックをすべて一度に移行します。これにより、完全なデプロイを実施する前にユーザーのサブセットで新しいバージョンをテストできます。

  Amazon ECS カナリアデプロイは次のシナリオに最適です。
  - 機能テスト: 完全なロールアウト前に少数のユーザーサブセットで新機能をテストする必要がある場合
  - 本番稼働用検証: 実際の本番稼働トラフィックでパフォーマンスと機能を検証する必要がある場合
  - ブラスト半径制御: 新しいバージョンで問題が見つかった場合にブラスト半径を最小限に抑える必要がある場合
  - ロードバランサーの要件: サービスが Application Load Balancer、Network Load Balancer、Service Connect を使用する場合
- 外部

  サードパーティーのデプロイコントローラーを使用します。
- ブルー/グリーンデプロイ (AWS CodeDeploy を搭載)

  CodeDeploy によってアプリケーションの更新されたバージョンが新しい置き換えタスクセットとしてインストールされ、本番トラフィックが元のアプリケーションタスクセットから置き換えタスクセットに再ルーティングされます。デプロイが正常に完了すると、元のタスクセットは終了します。本番トラフィックをルーティングする前に、サービスの新しいデプロイを検証するためにこのデプロイコントローラーを使用します。

Amazon ECS デプロイの失敗検出

Amazon ECS には、デプロイの失敗を検出する 2 つの方法があります。

- デプロイサーキットブレーカー

- CloudWatch アラーム

どちらの方法も、失敗したデプロイを既知の最後の正常な状態に自動的にロールバックするように設定できます。

以下の点を考慮してください。

- どちらの方法も、ローリング更新デプロイおよびブルー/グリーンデプロイのタイプのみをサポートしています。
- サービスが早期成功基準を使用する場合、Amazon ECS がデプロイを完了するまで、デプロイサーキットブレーカーと CloudWatch アラームのロールバックが適用されます。デプロイが完了すると、サービスはロールバックされなくなります。詳細については、「早期成功基準を使用した Amazon ECS ローリングデプロイの早期完了」を参照してください。
- 両方の方法を使用すると、いずれかによってデプロイの失敗がトリガーされる可能性があります。
- ロールバック方法には、COMPLETED 状態の前のデプロイが必要です。
- EventBridge イベントは、デプロイ状態の変更に対して生成されます。

Amazon ECS デプロイサーキットブレーカーが障害を検出する方法

デプロイサーキットブレーカーは、タスクが定常状態に到達したかどうかを判断する、ローリング更新メカニズムです。デプロイサーキットブレーカーには、デプロイが失敗した場合に、 COMPLETED 状態のデプロイに自動的にロールバックするオプションがあります。回路ブレーカーが障害をカウントする方法と、トリガーとなるしきい値をカスタマイズできるため、ロールバック動作をアプリケーションのスタートアップ特性とタスク障害に対する許容度に合わせることができます。

サービスデプロイの状態が変わったとき、Amazon ECS はサービスデプロイ状態変更イベントを EventBridge に送信します。これにより、サービスデプロイの状態を監視するためのプログラムによる方法がもたらされます。詳細については、「Amazon ECS サービスデプロイ状態変更イベント」を参照してください。デプロイを開始するための手動アクションを実行できるように、eventName が SERVICE_DEPLOYMENT_FAILED の EventBridge ルールをを使用して作成および監視することをお勧めします。詳細については、「Amazon EventBridge ユーザーガイド」の「EventBridge の開始方法」を参照してください。

デプロイサーキットブレーカーは、デプロイが失敗したと判断すると、COMPLETED 状態にある最新のデプロイを探します。このデプロイをロールバックデプロイとして使用します。ロールバックが開始されると、デプロイは COMPLETED から IN_PROGRESS に変わります。つまり、デプロイは COMPLETED 状態になるまで次のロールバックの対象にはなりません。デプロイサーキットブレー

カーが COMPLETED 状態のデプロイを見つけられない場合、サーキットブレーカーは新しいタスクを起動せず、デプロイは停止します。

サービスを作成すると、スケジューラーは起動に失敗したタスクを 2 段階で追跡します。

- ステージ 1 - スケジューラーはタスクが RUNNING 状態に移行するかどうかを監視します。
  - 成功 - RUNNING 状態に移行したタスクが複数あるため、デプロイメントが COMPLETED 状態に移行する可能性があります。障害基準はスキップされ、サーキットブレーカーはステージ 2 に移行します。
  - 失敗 - デフォルトでは、RUNNING 状態に移行しない連続タスクは失敗しきい値 (`resetOnHealthyTask` は `true`) にカウントされます。`resetOnHealthyTask` が `false` の場合、正常なタスクが失敗の間に開始されるかどうかに関係なく、すべてのタスクの失敗が累積されます。
- ステージ 2 - RUNNING 状態のタスクが 1 つ以上あると、デプロイメントはこの段階に入ります。サーキットブレーカーは、評価対象の現在のデプロイのタスクのヘルスチェックをチェックします。検証済みのヘルスチェックは、Elastic Load Balancing、AWS Cloud Map サービスヘルスチェック、コンテナヘルスチェックです。
  - 成功 - ヘルスチェックに合格した実行状態のタスクが少なくとも 1 つあります。
  - 失敗 - ヘルスチェックに失敗したために置き換えられたタスクが失敗のしきい値に達しました。

サービスでデプロイサーキットブレーカーメソッドを使用する際には、以下を考慮してください。EventBridge がルールを生成します。

- この `DescribeServices` レスポンスは、デプロイの状態、`rolloutState` および `rolloutStateReason` についての洞察を提供します。新しいデプロイが開始されると、ロールアウトの状態は `IN_PROGRESS` 状態から始まります。サービスが定常状態になると、ロールアウトの状態は `COMPLETED` に移行します。サービスが定常状態にならず、サーキットブレーカーがオンになっている場合、デプロイは `FAILED` 状態に移行します。`FAILED` 状態のデプロイでは、新しいタスクは起動されません。
- 開始および完了したデプロイに対して送信されるサービスデプロイの状態変更イベントに加えて、Amazon ECS はサーキットブレーカーがオンになったデプロイが失敗した場合にもイベントを送信します。これらのイベントは、デプロイが失敗した理由やロールバックのためにデプロイが開始されたかどうかについての詳細情報を提供します。詳細については、「Amazon ECS サービスデプロイ状態変更イベント」を参照してください。

- 以前のデプロイが失敗し、ロールバックが発生したために新しいデプロイが開始された場合、サービスデプロイ状態変更イベントの reason フィールドには、ロールバックのためにデプロイが開始されたことが示されます。
- デプロイサーキットブレーカーは、ローリング更新 (ECS) デプロイコントローラーを使用する Amazon ECS サービスでのみサポートされています。
- Amazon ECS コンソールを使用するか、CloudWatch オプションと共にデプロイサーキットブレーカーを使用する場合は AWS CLI を使用する必要があります。詳細については、「AWS Command Line Interfaceリファレンス」の「the section called “定義済みのパラメータを使用したサービスの作成”」および「create-service」を参照してください。

以下の create-service AWS CLI の例は、デプロイサーキットブレーカーと共にロールバックオプションが使用されている場合の Linux サービスの作成方法を示しています。

```
aws ecs create-service \
     --service-name MyService \
     --deployment-controller type=ECS \
     --desired-count 3 \
     --deployment-configuration "deploymentCircuitBreaker={enable=true,rollback=true}"
 \
     --task-definition sample-fargate:1 \
     --launch-type FARGATE \
     --platform-family LINUX \
     --platform-version 1.4.0 \
     --network-configuration
 "awsvpcConfiguration={subnets=[subnet-12344321],securityGroups=[sg-12344321],assignPublicIp=EN
```

次の create-service AWS CLI 例は、固定障害数 5 と累積障害追跡を使用するカスタムデプロイサーキットブレーカー設定でサービスを作成する方法を示しています。

```
aws ecs create-service \
     --service-name MyService \
     --deployment-controller type=ECS \
     --desired-count 10 \
     --deployment-configuration
 "deploymentCircuitBreaker={enable=true,rollback=true,resetOnHealthyTask=false,thresholdConfigu
 \
     --task-definition sample-fargate:1 \
     --launch-type FARGATE \
     --platform-family LINUX \
     --platform-version 1.4.0 \
```

```
    --network-configuration
"awsvpcConfiguration={subnets=[subnet-12344321],securityGroups=[sg-12344321],assignPublicIp=EN
```

例:

デプロイ 1 は COMPLETED 状態にあります。

デプロイ 2 は起動できないため、サーキットブレーカーはデプロイ 1 にロールバックされます。デプロイ 1 は IN_PROGRESS 状態に移行します。

デプロイ 3 が開始され、COMPLETED 状態のデプロイがないため、デプロイ 3 はロールバックやタスクの起動ができません。

失敗しきい値

デプロイサーキットブレーカは、デプロイを FAILED 状態にいつ移行するかを決定するためのしきい値を計算します。障害のカウント方法としきい値自体の両方を設定できます。

失敗回数モード

resetOnHealthyTask の設定は、デプロイ中にサーキットブレーカーがタスクの失敗をカウントする方法を制御します。

true (デフォルト)

失敗回数は、タスクが正常な状態に達するたびに 0 にリセットされます。連続した失敗のみがしきい値にカウントされます。このモードは、安定する前に断続的なスタートアップ障害が発生する可能性があるアプリケーションに役立ちます。

false

タスクの失敗は、デプロイ全体で累積されます。正常なタスクが失敗の間に開始されても、失敗回数はリセットされません。このモードでは、障害パターンが問題のあるデプロイを示している場合に、より迅速に検出できます。

しきい値の設定

thresholdConfiguration の設定では、サーキットブレーカーがトリガーされるタイミングを定義します。これには、しきい値の計算方法を決定する type と、使用する割合または数を指定する value が含まれます。

BOUNDED_PERCENT (デフォルト)

Amazon ECS は、障害しきい値を計算するために、最新のサービスの必要数を value で乗算します。結果は最小値 3、最大値 200 に制限されます。これは、デフォルト値が 50 のデフォルトタイプです。

UNBOUNDED_PERCENT

Amazon ECS は、障害しきい値を計算するために、最新のサービスの必要数を value で乗算します。結果に最小値や最大値の制限はありません。このタイプは、200 個の上限なしで比例しきい値を必要とする必要数が多いサービスに使用します。

COUNT

Amazon ECS は、 を障害のしきい値として value を直接使用します。しきい値は、サービスの必要数に関係なく固定されたままです。許容できる障害数を正確に指定したい場合、例えば開発環境でより迅速なロールバックを行うための低いしきい値を設定したい場合などに、このタイプを使用します。

パーセンテージタイプ (BOUNDED_PERCENT および UNBOUNDED_PERCENT) の場合、value の有効範囲は 1 ~ 100 です。デプロイ中、Amazon ECS は計算において常に最新のサービス要求数を使用します。

BOUNDED_PERCENT がしきい値を計算する方法

デフォルトの BOUNDED_PERCENT のしきい値タイプを使用すると、デプロイサーキットブレーカーは次の式を使用してしきい値を計算します。

```
Minimum threshold (3) <= (value/100) * desired task count => Maximum threshold (200)
```

計算の結果が最小値 3 を下回った場合、しきい値は 3 に設定されます。結果が 200 より大きい場合、しきい値は 200 に設定されます。それ以外の場合、しきい値は計算された値 (切り上げ) に設定されます。

次の表は、デフォルト値の 50 を使用した例を示しています。

希望タスク数	計算	Threshold
1	`3 <= 0.5 * 1 => 200`	3 (計算値は最小値より小さい)

希望タスク数	計算	Threshold
25	`3 <= 0.5 * 25 => 200`	13 (値は切り上げられます)
400	`3 <= 0.5 * 400 => 200`	200 (計算値は最大値を超えています)
800	`3 <= 0.5 * 800 => 200`	200 (計算値は最大値を超えています)

UNBOUNDED_PERCENT では、同じ計算が適用されますが、最小値と最大値の制限はありません。例えば、目標件数が 800 で値が 50 のサービスの場合、しきい値は 400 になります。

デプロイステータスチェックには 2 つの段階があります。

1. デプロイサーキットブレーカは、デプロイの一部であるタスクを監視して RUNNING 状態のタスクを確認します。スケジューラは、現在のデプロイのタスクが RUNNING 状態のときに失敗条件を無視して次のステージに進みます。タスクが RUNNING 状態に到達できなかった場合、デプロイサーキットブレーカは故障数を 1 つ増やします。障害カウントがしきい値に等しい場合、デプロイは FAILED としてマークされます。
2. RUNNING 状態のタスクが 1 つ以上ある場合、このステージが開始されます。デプロイサーキットブレーカは、現在のデプロイのタスクの次のリソースにヘルスチェックを実行します。
   - Elastic Load Balancing ロードバランサー
   - AWS Cloud Map のサービス
   - Amazon ECS コンテナヘルスチェック

   タスクのヘルスチェックが失敗した場合、デプロイサーキットブレーカーは障害数を 1 つ増やします。障害カウントがしきい値に等しい場合、デプロイは FAILED としてマークされます。

たとえば、しきい値が 3 の場合、サーキットブレーカーは故障回数が 0 に設定された状態で起動します。タスクが RUNNING 状態に到達できなかった場合、デプロイサーキットブレーカは故障数を 1 つ増やします。障害カウントが 3 である場合、デプロイは FAILED としてマークされます。

ロールバックオプションの使用方法に関するその他の例については、「Amazon ECS デプロイサーキットブレーカーのお知らせ」を参照してください。

CloudWatch アラームが Amazon ECS のデプロイ障害を検出する方法

指定された CloudWatch アラームが ALARM 状態になったことを検出したとき、デプロイが失敗に設定されるように Amazon ECS を設定できます。

オプションとして、失敗したデプロイを最後に完了したデプロイにロールバックするように設定できます。

以下の `create-service` AWS CLI の例は、デプロイアラームと共にロールバックオプションが使用されている場合の Linux サービスの作成方法を示しています。

```
aws ecs create-service \
     --service-name MyService \
     --deployment-controller type=ECS \
     --desired-count 3 \
     --deployment-configuration
 "alarms={alarmNames=[alarm1Name,alarm2Name],enable=true,rollback=true}" \
     --task-definition sample-fargate:1 \
     --launch-type FARGATE \
     --platform-family LINUX \
     --platform-version 1.4.0 \
     --network-configuration
 "awsvpcConfiguration={subnets=[subnet-12344321],securityGroups=[sg-12344321],assignPublicIp=EN
```

サービスで Amazon CloudWatch アラームメソッドを使用する場合は、以下を考慮してください。

- 本番トラフィックが移行した後に、ブルーおよびグリーンのサービスリビジョンの両方が同時に実行される期間。Amazon ECS は、デプロイに関連するアラーム設定に基づいてこの期間を計算します。この値は設定できません。
- `deploymentConfiguration` リクエストパラメータには `alarms` のデータ型が含まれるようになっています。アラーム名、メソッド使用の有無、アラームがデプロイの失敗を示したときのロールバック開始の有無について指定できます。詳細については、「Amazon Elastic Container Service API リファレンス」の「CreateService」を参照してください。
- この `DescribeServices` レスポンスは、デプロイの状態、`rolloutState` および `rolloutStateReason` についての洞察を提供します。新しいデプロイが開始されると、ロールアウトの状態は `IN_PROGRESS` 状態から始まります。サービスが定常状態になりベイク時間が完了すると、ロールアウトの状態は `COMPLETED` に移行します。サービスが定常状態にならず、アラームが `ALARM` 状態になった場合、デプロイは `FAILED` 状態に移行します。`FAILED` 状態のデプロイでは、新しいタスクは起動されません。

- 開始および完了したデプロイに対して Amazon ECS から送信されるサービスデプロイの状態変更イベントに加えて、Amazon ECS はアラームを使用するデプロイが失敗した場合にもイベントを送信します。これらのイベントは、デプロイが失敗した理由やロールバックのためにデプロイが開始されたかどうかについての詳細情報を提供します。詳細については、「Amazon ECS サービスデプロイ状態変更イベント」を参照してください。
- 以前のデプロイが失敗し、ロールバックがオンになったために新しいデプロイが開始された場合、サービスデプロイ状態変更イベントの `reason` フィールドには、ロールバックのためにデプロイが開始されたことが示されます。
- 障害検出にデプロイサーキットブレーカーと Amazon CloudWatch アラームを使用する場合、どちらかのメソッドの基準が満たされ次第、どちらかがデプロイの失敗を開始します。デプロイの失敗を開始したメソッドでロールバックオプションが使用されている場合は、ロールバックが発生します。
- Amazon CloudWatch アラームは、ローリング更新 (ECS) デプロイコントローラーを使用する Amazon ECS サービスでのみサポートされています。
- このオプションは、Amazon ECS コンソールまたは AWS CLI を使用して設定できます。詳細については、「AWS Command Line Interfaceリファレンス」の「the section called “定義済みのパラメータを使用したサービスの作成”」および「create-service」を参照してください。
- デプロイステータスが長時間 `IN_PROGRESS` のままになっていると気付くことがあるかもしれません。これは、アクティブなデプロイを削除し終えるまで Amazon ECS のステータスは変化せず、また、ベイク時間が過ぎるまでは、このステータスが変更されることがないためです。アラームの設定によっては、デプロイにアラームを使用しない場合よりも数分長くかかるように思える場合もあります (新しいプライマリタスクセットがスケールアップされ、古いデプロイがスケールダウンされる場合であってもです)。CloudFormation タイムアウトを使用する場合は、タイムアウトを増やすことを検討してください。詳細については、「AWS CloudFormation ユーザーガイド」の「テンプレートでの待機条件の作成」を参照してください。
- Amazon ECS は `DescribeAlarms` を呼び出してアラームをポーリングします。`DescribeAlarms` の呼び出しはアカウントに関連付けられた CloudWatch の Service Quotas にカウントされます。`DescribeAlarms` を呼び出す AWS サービスが他にもある場合は、アラームをポーリングする際に Amazon ECS に影響が及ぶ可能性があります。例えば、別のサービスがクォータに達するだけの `DescribeAlarms` 呼び出しを行うと、そのサービスがスロットリングされると共に Amazon ECS もスロットリングされ、アラームをポーリングできなくなります。スロットリング期間中にアラームが生成されると、Amazon ECS はアラームを見逃してロールバックが行われない可能性があります。デプロイに他の影響はありません。CloudWatch の Service Quotas の詳細については、「CloudWatch ユーザーガイド」の「CloudWatch Service Quotas」を参照してください。

- デプロイの開始時にアラームの状態が ALARM である場合、Amazon ECS はそのデプロイ中はアラームの監視を行いません (Amazon ECS はアラーム設定を無視します)。この動作により、初期デプロイの失敗を修正するために、新たなデプロイを開始する場合に対応しています。

推奨アラーム

次のアラームメトリクスを使用することをお勧めします。

- Application Load Balancer を使用する場合は、HTTPCode_ELB_5XX_Count および HTTPCode_ELB_4XX_Count の Application Load Balancer メトリクスを使用します。これらのメトリクスは HTTP スパイクをチェックします。Application Load Balancer メトリクスの詳細については、「Application Load Balancer のユーザーガイド」の「Application Load Balancer の CloudWatch メトリクス」を参照してください。
- 既存のアプリケーションがある場合は、CPUUtilization および MemoryUtilization のメトリクスを使用します。これらのメトリクスは、クラスターまたはサービスが使用する CPU とメモリの割合をチェックします。詳細については、「the section called “考慮事項”」を参照してください。
- タスクで Amazon Simple Queue Service キューを使用する場合は、ApproximateNumberOfMessagesNotVisible Amazon SQS メトリクスを使用します。このメトリクスは、遅延の発生によりすぐに読み取ることのできない、キュー内のメッセージ数をチェックします。Amazon SQS メトリクスの詳細については、「Amazon Simple Queue Service デベロッパーガイド」の「Amazon SQS で使用できる CloudWatch メトリクス」を参照してください。

ベイク時間

サービスのデプロイにロールバックオプションを使用すると、Amazon ECS はターゲットサービスリビジョンがデプロイされてから CloudWatch アラームを送信するまでの間さらに待機します。これはベイク時間と呼ばれます。この時間は、次のイベントの後に開始されます。

- ターゲットサービスリビジョンのすべてのタスクが実行中で、正常な状態である
- ソースサービスリビジョンは 0% にスケールダウンされている

デフォルトのベイク時間は 5 分未満です。サービスデプロイは、ベイク時間が経過すると完了としてマークされます。

ローリングデプロイのベイク時間を設定できます。CloudWatch アラームを使用して障害を検出し、ベイク時間を変更してから Amazon ECS のデフォルトを設定する場合は、ベイク時間を手動で設定する必要があります。

Amazon ECS サービスデプロイのライフサイクルフック

デプロイが開始されると、ライフサイクルステージが進行されます。各ステージは、IN_PROGRESS や SUCCEEDED などの状態になります。ライフサイクルフックを特定の段階で設定して、カスタムロジックを実行したり、デプロイが進む前に決定ポイントを導入したりできます。Amazon ECS は、次の 2 種類のライフサイクルフックに対応しています。

Lambda フック

Amazon ECS は、デプロイの特定の段階で Lambda 関数を呼び出します。関数にはカスタムロジックが含まれており、続行する方法を Amazon ECS に伝えるには SUCCEEDED、FAILED、または IN_PROGRESS の hookStatus を含む JSON オブジェクトを返す必要があります。デプロイが進む前に、これらのフックを使用して、検証テストの実行、ガバナンスポリシーの適用、カスタム承認ステップの実装を行うことができます。詳細については、「Amazon ECS サービスデプロイの Lambda フック」を参照してください。

フックを一時停止する

Amazon ECS は、設定されたライフサイクルステージでデプロイを一時停止し、ContinueServiceDeployment API を呼び出して続行またはロールバックを待ちます。一時停止中は、手動承認、既存ツールとの統合テスト、運用準備チェック、CI/CD パイプラインステップなど、独自のワークフローを外部で実行できます。詳細については、「Amazon ECS サービスデプロイの一時停止フック」を参照してください。

Lambda フックと一時停止フックの両方を同じライフサイクルステージで設定できます。デプロイが次のステージに進む前に、両方のフックを完了する必要があります。

ライフサイクルフックの詳細

デプロイ中にフックがアクティブになると、DescribeServiceDeployments を呼び出してそのステータスを表示できます。レスポンスには、アクティブなフックごとに次のフィールドを持つ lifecycleHookDetails 配列が含まれます。

フィールド	説明
hookId	このフック実行の一意の識別子。この値は、一時停止フックの ContinueServiceDeployment を呼び出すときに使用します。
targetType	フックのタイプ: AWS_LAMBDA または PAUSE。
targetArn	フックターゲットの ARN。Lambda フックの場合、これは Lambda 関数 ARN です。一時停止フックの場合、このフィールドは設定されません。
status	フックの現在のステータス: AWAITING_ACTION 、IN_PROGRESS 、SUCCEEDED 、FAILED、または TIMED_OUT 。
expiresAt	フックの有効期限が切れる日時。(例: 2026-05-06T12:06:49-07:00)
timeoutAction	フックがタイムアウトしたときに Amazon ECS が実行するアクション: ROLLBACK または CONTINUE。

ライフサイクルステージのカテゴリ

ライフサイクルステージは 2 つのカテゴリに分類されます。

1. 1 回呼び出しステージ – Amazon ECS は、サービスのデプロイ中にこれらのステージを 1 回だけ呼び出します。
   - RECONCILE_SERVICE
   - PRE_SCALE_UP
   - POST_SCALE_UP
   - TEST_TRAFFIC_SHIFT
   - POST_TEST_TRAFFIC_SHIFT
   - POST_PRODUCTION_TRAFFIC_SHIFT
2. 反復呼び出しステージ – Amazon ECS は、サービスのデプロイ中にこれらのステージを複数回呼び出すことができます。線形デプロイとカナリアデプロイの場合、これらのステージはトラフィックシフトの各ステップで呼び出されます。
   - PRE_PRODUCTION_TRAFFIC_SHIFT
   - PRODUCTION_TRAFFIC_SHIFT

> **Note**
>
> これらのステージはロールバック中に呼び出されるため、一時停止フックを TEST_TRAFFIC_SHIFT または PRODUCTION_TRAFFIC_SHIFT に設定することはできません。ロールバック中に一時停止するには、ロールバックを完了するための追加の ContinueServiceDeployment 呼び出しが必要になります。

フックタイプ別のサポートされているステージ

ライフサイクルステージ	Lambda フック	フックを一時停止する
RECONCILE_SERVICE	可能	はい
PRE_SCALE_UP	はい	はい
POST_SCALE_UP	はい	はい
TEST_TRAFFIC_SHIFT	はい	いいえ
POST_TEST_TRAFFIC_SHIFT	はい	はい
PRE_PRODUCTION_TRAFFIC_SHIFT	はい	はい
PRODUCTION_TRAFFIC_SHIFT	はい	いいえ
POST_PRODUCTION_TRAFFIC_SHIFT	はい	はい

Amazon ECS サービスデプロイの Lambda フック

Lambda フックは、Amazon ECS がデプロイの特定の段階で呼び出す Lambda 関数です。デプロイが進む前に、フックを使用して、検証テストの実行、ガバナンスポリシーの適用、カスタムロジックの実装を行うことができます。

Amazon ECS がフックを呼び出すと、関数が hookStatus フィールドを含む JSON オブジェクトを返す必要があります。必要に応じて、再試行のタイミングを制御するための callBackDelay と、呼び出し間でデータを渡すための hookDetails を含めることができます。関数が有効な

hookStatus を返さない場合、または失敗した場合、Amazon ECS はデプロイをロールバックします。

hookStatus 値

有効な hookStatus 値を次に示します。

- SUCCEEDED – デプロイは次のライフサイクルステージへと進みます。
- FAILED – Amazon ECS は最後に成功したサービスリビジョンにデプロイをロールバックします。
- IN_PROGRESS – Amazon ECS は遅延後に関数を再度呼び出します。遅延のデフォルト値は 30 秒です。この値は、hookStatus とともに callBackDelay を返すことでカスタマイズできます。

次の例では、カスタムコールバックの遅延を使用して hookStatus を返す方法が示されます。この例では、Amazon ECS によってこのフックはデフォルトの 30 秒ではなく 60 秒後に再試行されます。

```
{
    "hookStatus": "IN_PROGRESS",
    "callBackDelay": 60
}
```

hookDetails を使用した状態の受け渡し

hookDetails フィールドは、ライフサイクルフック関数にデータを渡すために使用できるディクショナリです。hookDetails の入力方法は次の 2 つです。

- サービスの作成時または更新時 – サービス定義内のライフサイクルフック設定で hookDetails を定義します。Amazon ECS は、呼び出しのたびにこのデータを関数に渡します。これを使用して、サービス固有の設定を渡すことで、フックを複数のサービス間で再利用できるようにします。
- IN_PROGRESS レスポンスによるランタイム時 – フックステータス IN_PROGRESS とともに hookDetails を返します。Amazon ECS は、次の呼び出し時にこのデータを関数に返します。これを使用して、外部ストレージのない呼び出し間の状態を維持します。

次の例は、S3 バケット名を関数に渡すサービス定義のライフサイクルフック設定を示しています。

```
{
    "hookTargetArn": "arn:aws:lambda:us-west-2:123456789012:function:my-approval-hook",
    "roleArn": "arn:aws:iam::123456789012:role/ecs-lambda-invoke-role",
    "lifecycleStages": [
        "POST_TEST_TRAFFIC_SHIFT"
    ],
    "hookDetails": {
        "S3_BUCKET_NAME": "my-approval-bucket"
    }
}
```

関数が IN_PROGRESS を返すと、レスポンスに hookDetails を含めることもできます。Amazon ECS はこのデータをマージし、次の呼び出し時に返します。一般的なユースケースには、呼び出し間で外部リソースのメトリクスカウンターまたは ARNs を渡すことが含まれます。

```
{
    "hookStatus": "IN_PROGRESS",
    "callBackDelay": 30,
    "hookDetails": {
        "approvalChecked": true,
        "S3_BUCKET_NAME": "my-approval-bucket"
    }
}
```

Amazon ECS は次の呼び出しで executionDetails とともにイベントペイロードに hookDetails を含めます。

```
{
  "executionId": "e8d5a28f-eb01-4f3c-9454-a30ba6dc54bc",
  "lifecycleStage": "POST_TEST_TRAFFIC_SHIFT",
  "resourceArn": "arn:aws:ecs:us-west-2:123456789012:service-deployment/my-cluster/my-
service/EZe5RNVLH6PPzHXINuP28",
  "executionDetails": {
    "serviceArn": "arn:aws:ecs:us-west-2:123456789012:service/my-cluster/my-service",
    "targetServiceRevisionArn": "arn:aws:ecs:us-west-2:123456789012:service-revision/
my-cluster/my-service/9313423515462893900",
    "testTrafficWeights": {},
    "productionTrafficWeights": {}
```

```
  },
  "hookDetails": {
    "approvalChecked": true,
    "S3_BUCKET_NAME": "my-approval-bucket"
  }
}
```

関数は event["hookDetails"] を読み取り、以前の呼び出しの設定と状態にアクセスできます。

> **Note**
>
> ランタイム時に hookDetails に追加されたデータは、1 つのデプロイ内の同じフックの呼び出し間でのみ保持されます。データは、同じデプロイ内の異なるフック間や、異なるデプロイ内の同じフック間で引き継がれることはありません。

タイムアウト設定

timeoutConfiguration を使用して Lambda フックのタイムアウトを設定できます。Lambda 関数がタイムアウトの有効期限が切れる前に SUCCEEDED または FAILED を返さない場合、Amazon ECS は設定されたタイムアウトアクションを実行します。

フィールド	説明
timeoutConfiguration.timeoutInMinutes	待機時間。有効な範囲: 1 ~ 1,440 分 (24 時間)。デフォルト: 1,440 分。
timeoutConfiguration.action	タイムアウトが期限切れになった場合に実行するアクション。有効な値: ROLLBACK、CONTINUE。デフォルト: ROLLBACK。

サポートされているライフサイクルステージ

Lambda フックは、次のライフサイクルステージで設定できます。

- RECONCILE_SERVICE
- PRE_SCALE_UP
- POST_SCALE_UP

- TEST_TRAFFIC_SHIFT
- POST_TEST_TRAFFIC_SHIFT
- PRE_PRODUCTION_TRAFFIC_SHIFT
- PRODUCTION_TRAFFIC_SHIFT
- POST_PRODUCTION_TRAFFIC_SHIFT

線形デプロイとカナリアデプロイの場合、PRODUCTION_TRAFFIC_SHIFT または PRE_PRODUCTION_TRAFFIC_SHIFT に設定された Lambda フックは、トラフィックシフトの各ステップで呼び出されます。

ライフサイクルステージのカテゴリの詳細については、「Amazon ECS サービスデプロイのライフサイクルフック」を参照してください。

ライフサイクルペイロード

一般的なペイロード構造

Amazon ECS がライフサイクルフック Lambda 関数を呼び出すと、イベントペイロードに次の最上位フィールドが含まれます。

- executionId – このフック実行の一意の識別子。
- lifecycleStage – 現在のライフサイクルステージ (PRODUCTION_TRAFFIC_SHIFT など)。
- resourceArn – デプロイに関連付けられたリソースの ARN。
- executionDetails – 次のリストで説明されているデプロイ固有の情報を含むオブジェクト。

executionDetails オブジェクトには、以下のフィールドが含まれています。

- serviceArn – サービスの Amazon リソースネーム (ARN)。
- targetServiceRevisionArn – デプロイされるターゲットサービスリビジョンの ARN。
- testTrafficWeights – サービスリビジョン ARN が対応するテストトラフィックの重みの割合を示すマップ。
- productionTrafficWeights – サービスリビジョン ARN が対応する本番トラフィックの重みの割合を示すマップ。

次の例は、Lambda 関数が受け取る完全なイベント構造を示しています。

```
{
  "executionId": "f4fcae0f-9bec-41c6-ba87-0eaa0cef8af5",
  "lifecycleStage": "PRODUCTION_TRAFFIC_SHIFT",
  "resourceArn": "arn:aws:ecs:us-west-2:123456789012:service-deployment/my-cluster/my-
service/EZe5RNVLH6PPzHXINuP28",
  "executionDetails": {
    "serviceArn": "arn:aws:ecs:us-west-2:123456789012:service/my-cluster/my-service",
    "targetServiceRevisionArn": "arn:aws:ecs:us-west-2:123456789012:service-revision/
my-cluster/my-service/9313423515462893900",
    "testTrafficWeights": {},
    "productionTrafficWeights": {
      "arn:aws:ecs:us-west-2:123456789012:service-revision/my-cluster/my-
service/9313423515462893900": 100,
      "arn:aws:ecs:us-west-2:123456789012:service-revision/my-cluster/my-
service/19204984629365805O4": 0
    }
  }
}
```

ライフサイクルステージペイロード

後のセクションで、各ライフサイクルステージのペイロードの例を示しています。これらの例では、グリーンサービスリビジョン (9313423515462893900) はデプロイされている新しいリビジョンであり、ブルーサービスリビジョン (19204984629365805O4) は既存の本番用リビジョンです。

PRE_SCALE_UP

このステージは、Amazon ECS がグリーンサービスリビジョンタスクを起動する前に発生します。グリーンサービスリビジョンは開始されておらず、トラフィックはルーティングされません。

```
{
  "executionId": "e8d5a28f-eb01-4f3c-9454-a30ba6dc54bc",
  "lifecycleStage": "PRE_SCALE_UP",
  "resourceArn": "arn:aws:ecs:us-west-2:123456789012:service-deployment/my-cluster/my-
service/EZe5RNVLH6PPzHXINuP28",
  "executionDetails": {
    "serviceArn": "arn:aws:ecs:us-west-2:123456789012:service/my-cluster/my-service",
    "targetServiceRevisionArn": "arn:aws:ecs:us-west-2:123456789012:service-revision/
my-cluster/my-service/9313423515462893900",
```

```
    "testTrafficWeights": {},
    "productionTrafficWeights": {}
  }
}
```

POST_SCALE_UP

このステージは、Amazon ECS がグリーンサービスリビジョンタスクを起動し、それらが正常になったら発生します。グリーンタスクは実行中ですが、まだトラフィックを受信していません。

```
{
  "executionId": "8b095b05-7bb0-4c56-a223-a3f61f4f9295",
  "lifecycleStage": "POST_SCALE_UP",
  "resourceArn": "arn:aws:ecs:us-west-2:123456789012:service-deployment/my-cluster/my-service/EZe5RNVLH6PPzHXINuP28",
  "executionDetails": {
    "serviceArn": "arn:aws:ecs:us-west-2:123456789012:service/my-cluster/my-service",
    "targetServiceRevisionArn": "arn:aws:ecs:us-west-2:123456789012:service-revision/my-cluster/my-service/9313423515462893900",
    "testTrafficWeights": {},
    "productionTrafficWeights": {}
  }
}
```

TEST_TRAFFIC_SHIFT

このステージは、Amazon ECS がテストトラフィックをグリーンサービスリビジョンにシフトさせるときに発生します。testTrafficWeights は、グリーンリビジョンがテストトラフィックを 100% 受信する一方で、ブルーリビジョンは受信しない (0%) ことを示しています。本番トラフィックは引き続きブルーリビジョンに流れます。

```
{
  "executionId": "779085de-ab47-42bc-84ad-41f9914a8643",
  "lifecycleStage": "TEST_TRAFFIC_SHIFT",
  "resourceArn": "arn:aws:ecs:us-west-2:123456789012:service-deployment/my-cluster/my-service/EZe5RNVLH6PPzHXINuP28",
  "executionDetails": {
    "serviceArn": "arn:aws:ecs:us-west-2:123456789012:service/my-cluster/my-service",
```

```
    "targetServiceRevisionArn": "arn:aws:ecs:us-west-2:123456789012:service-revision/
my-cluster/my-service/9313423515462893900",
    "testTrafficWeights": {
      "arn:aws:ecs:us-west-2:123456789012:service-revision/my-cluster/my-
service/9313423515462893900": 100,
      "arn:aws:ecs:us-west-2:123456789012:service-revision/my-cluster/my-
service/1920498462936580504": 0
    },
    "productionTrafficWeights": {}
  }
}
```

POST_TEST_TRAFFIC_SHIFT

このステージは、Amazon ECS がテストトラフィックシフトを完了した後に発生します。グリーンサービスリビジョンにより、テストトラフィックの 100% が処理されます。

```
{
  "executionId": "3a0345ba-b029-404b-890d-7da2a4b266aa",
  "lifecycleStage": "POST_TEST_TRAFFIC_SHIFT",
  "resourceArn": "arn:aws:ecs:us-west-2:123456789012:service-deployment/my-cluster/my-
service/EZe5RNVLH6PPzHXINuP28",
  "executionDetails": {
    "serviceArn": "arn:aws:ecs:us-west-2:123456789012:service/my-cluster/my-service",
    "targetServiceRevisionArn": "arn:aws:ecs:us-west-2:123456789012:service-revision/
my-cluster/my-service/9313423515462893900",
    "testTrafficWeights": {},
    "productionTrafficWeights": {}
  }
}
```

PRE_PRODUCTION_TRAFFIC_SHIFT

このステージは、Amazon ECS が本番トラフィックをグリーンサービスリビジョンにシフトする前に発生します。線形デプロイとカナリアデプロイの場合、このステージはトラフィックシフトの各ステップの前に呼び出されます。

```
{
```

```
  "executionId": "a2b3c4d5-e6f7-8901-abcd-ef1234567890",
  "lifecycleStage": "PRE_PRODUCTION_TRAFFIC_SHIFT",
  "resourceArn": "arn:aws:ecs:us-west-2:123456789012:service-deployment/my-cluster/my-
service/EZe5RNVLH6PPzHXINuP28",
  "executionDetails": {
    "serviceArn": "arn:aws:ecs:us-west-2:123456789012:service/my-cluster/my-service",
    "targetServiceRevisionArn": "arn:aws:ecs:us-west-2:123456789012:service-revision/
my-cluster/my-service/9313423515462893900",
    "testTrafficWeights": {},
    "productionTrafficWeights": {
      "arn:aws:ecs:us-west-2:123456789012:service-revision/my-cluster/my-
service/9313423515462893900": 0,
      "arn:aws:ecs:us-west-2:123456789012:service-revision/my-cluster/my-
service/1920498462936580504": 100
    }
  }
}
```

PRODUCTION_TRAFFIC_SHIFT

このステージは、Amazon ECS が本番トラフィックをグリーンサービスリビジョンにシフトさせるときに発生します。productionTrafficWeights は、グリーンリビジョンが本番トラフィックを 100% 受信する一方で、ブルーリビジョンは受信しない (0%) ことを示しています。

```
{
  "executionId": "f4fcae0f-9bec-41c6-ba87-0eaa0cef8af5",
  "lifecycleStage": "PRODUCTION_TRAFFIC_SHIFT",
  "resourceArn": "arn:aws:ecs:us-west-2:123456789012:service-deployment/my-cluster/my-
service/EZe5RNVLH6PPzHXINuP28",
  "executionDetails": {
    "serviceArn": "arn:aws:ecs:us-west-2:123456789012:service/my-cluster/my-service",
    "targetServiceRevisionArn": "arn:aws:ecs:us-west-2:123456789012:service-revision/
my-cluster/my-service/9313423515462893900",
    "testTrafficWeights": {},
    "productionTrafficWeights": {
      "arn:aws:ecs:us-west-2:123456789012:service-revision/my-cluster/my-
service/9313423515462893900": 100,
      "arn:aws:ecs:us-west-2:123456789012:service-revision/my-cluster/my-
service/1920498462936580504": 0
    }
  }
```

```
}
```

POST_PRODUCTION_TRAFFIC_SHIFT

このステージは、Amazon ECS が本番トラフィックシフトを完了した後に発生します。グリーンサービスリビジョンは、すべての本番トラフィックを処理します。

```
{
  "executionId": "5f40ed04-7e54-437d-b95d-98bc872fec49",
  "lifecycleStage": "POST_PRODUCTION_TRAFFIC_SHIFT",
  "resourceArn": "arn:aws:ecs:us-west-2:123456789012:service-deployment/my-cluster/my-
service/EZe5RNVLH6PPzHXINuP28",
  "executionDetails": {
    "serviceArn": "arn:aws:ecs:us-west-2:123456789012:service/my-cluster/my-service",
    "targetServiceRevisionArn": "arn:aws:ecs:us-west-2:123456789012:service-revision/
my-cluster/my-service/9313423515462893900",
    "testTrafficWeights": {},
    "productionTrafficWeights": {}
  }
}
```

ロールバック中のライフサイクルフック

専用の ROLLBACK ライフサイクルステージはありません。ロールバックが発生すると、Amazon ECS は PRODUCTION_TRAFFIC_SHIFT ステージと TEST_TRAFFIC_SHIFT ステージ (いずれも反復呼び出しステージ) に登録されたフックを再び呼び出しします。ロールバック中、ペイロードの productionTrafficWeights は、ブルーリビジョンに戻るトラフィックを表示します。

targetServiceRevisionArn は、本来のデプロイのターゲットであるため、トラフィックがシフトしてもグリーンリビジョン ARN のままです。

次の例は、ロールバック中の PRODUCTION_TRAFFIC_SHIFT ペイロードを示しています。グリーンリビジョン (1920498462936580504) が本番トラフィックを 100% 受信する一方で、ブルーリビジョン (9313423515462893900) は受信しない (0%) 点に注意してください。

```
{
  "executionId": "70073435-cb99-457f-b900-6ee1dcad05ec",
  "lifecycleStage": "PRODUCTION_TRAFFIC_SHIFT",
```

```
  "resourceArn": "arn:aws:ecs:us-west-2:123456789012:service-deployment/my-cluster/my-
service/EZe5RNVLH6PPzHXINuP28",
  "executionDetails": {
    "serviceArn": "arn:aws:ecs:us-west-2:123456789012:service/my-cluster/my-service",
    "targetServiceRevisionArn": "arn:aws:ecs:us-west-2:123456789012:service-revision/
my-cluster/my-service/9313423515462893900",
    "testTrafficWeights": {},
    "productionTrafficWeights": {
      "arn:aws:ecs:us-west-2:123456789012:service-revision/my-cluster/my-
service/9313423515462893900": 0,
      "arn:aws:ecs:us-west-2:123456789012:service-revision/my-cluster/my-
service/1920498462936580504": 100
    }
  }
}
```

ロールバック中にフックが呼び出されているかどうかを見極めるには、productionTrafficWeights を確認します。targetServiceRevisionArn (グリーンリビジョン) の重みが 0% で、もう一方のリビジョンの重みが 100% の場合、デプロイはロールバックされます。

Amazon ECS サービスデプロイの一時停止フック

一時停止フックは、特定のライフサイクル段階で Amazon ECS サービスのデプロイを一時停止し、デプロイが進む前に ContinueServiceDeployment を呼び出すのを待ちます。デプロイを続行する前に手動承認または外部検証が必要な場合は、一時停止フックを使用します。

一時停止フックの仕組み

一時停止フックが設定されたライフサイクルステージにデプロイが達すると、次が発生します。

1. Amazon ECS は、一時停止フックに一意の hookId を生成します。
2. Amazon ECS は、detail-type ECS Hook State Change とイベント名 HOOK_AWAITING_ACTION を使用して EventBridge イベントを出力します。
3. デプロイは、hookId および CONTINUE または ROLLBACK のアクションを使用して ContinueServiceDeployment を呼び出すか、設定されたタイムアウトに達するまで、一時停止されたままになります。

DescribeServiceDeployments を呼び出して hookId を取得できます。レスポンスには、フックステータスの lifecycleHookDetails 配列が含まれます。

```
{
    "serviceDeployments": [
        {
            "lifecycleHookDetails": [
                {
                    "hookId": "ecs-pause-e7tK9G_WRJqNF_EOMjztDXfKenlJuEUVjsNStf4WLKw",
                    "targetType": "PAUSE",
                    "status": "AWAITING_ACTION",
                    "expiresAt": "2024-01-15T12:00:00Z",
                    "timeoutAction": "ROLLBACK"
                }
            ]
        }
    ]
}
```

一時停止フックの設定

次の例は、テストトラフィックシフトの完了後にデプロイを一時停止するサービス定義の一時停止フック設定を示しています。

```
{
    "deploymentConfiguration": {
        "strategy": "BLUE_GREEN",
        "lifecycleHooks": [
            {
                "targetType": "PAUSE",
                "lifecycleStages": [
                    "POST_TEST_TRAFFIC_SHIFT"
                ],
                "timeoutConfiguration": {
                    "timeoutInMinutes": 10,
                    "action": "ROLLBACK"
                }
            }
        ]
    }
}
```

設定フィールド

フィールド	説明	有効値	必須
targetType	フックのタイプ。一時停止フックの場合は PAUSE である必要があります。	PAUSE	はい
lifecycleStages	デプロイが一時停止するライフサイクルステージ。	RECONCILE_SERVICE , PRE_SCALE_UP , POST_SCALE_UP , POST_TEST_TRAFFIC_SHIFT , PRE_PRODUCTION_TRAFFIC_SHIFT , POST_PRODUCTION_TRAFFIC_SHIFT	はい
timeoutConfiguration.timeoutInMinutes	タイムアウトアクションを実行するまでの待機時間。デフォルト: 1,440 分 (24 時間)。	1 ~ 20,160 (14 日間)	いいえ
timeoutConfiguration.action	タイムアウトが期限切れになった場合に実行するアクション。	ROLLBACK"" (デフォルト)CONTINUE	いいえ

制約

- サービスごとに最大 10 個の一時停止フックと 10 個の Lambda フックを設定できます。

- 一時停止フックは hookTargetArn または roleArn を使用しません。これらのフィールドは Lambda フック専用です。
- 一時停止フックは、TEST_TRAFFIC_SHIFT または PRODUCTION_TRAFFIC_SHIFT ステージでは設定できません。

一時停止したデプロイの続行

デプロイが一時停止されたとき (一時停止フックのステータスは AWAITING_ACTION)、hookId と実行するアクションを使用して ContinueServiceDeployment を呼び出します。

デプロイを続行するには:

```
aws ecs continue-service-deployment \
    --hook-id ecs-pause-e7tK9G_WRJqNF_EOMjztDXfKenlJuEUVjsNStf4WLKw \
    --action CONTINUE
```

デプロイをロールバックするには:

```
aws ecs continue-service-deployment \
    --hook-id ecs-pause-e7tK9G_WRJqNF_EOMjztDXfKenlJuEUVjsNStf4WLKw \
    --action ROLLBACK
```

詳細については、「サービスデプロイの継続」を参照してください。

同じステージの複数のフック

同じライフサイクルステージで複数のフックを設定すると、そのステージのすべてのフックが続行されるまで、デプロイは一時停止したままになります。いずれかのフックがロールバックをトリガーする場合、そのステージのその他のフックのステータスに関係なく、デプロイ全体がロールバックされます。

この動作は、Lambda フックと一時停止フックの両方が同じライフサイクルステージで設定されている場合に適用されます。デプロイは、そのステージのすべてのフックが正常に完了したか、続行された場合にのみ続行します。

同じライフサイクルステージで設定されたすべてのフックは並行して実行されます。

タイムアウト動作

タイムアウトの有効期限が切れる前に ContinueServiceDeployment を呼び出さない場合、Amazon ECS は設定されたタイムアウトアクションを実行します。

- ROLLBACK (デフォルト) - Amazon ECS はデプロイを以前のサービスリビジョンにロールバックします。
- CONTINUE – Amaon ECS はデプロイは次のライフサイクルステージへと進めます。

DescribeServiceDeployments レスポンスの lifecycleHookDetails 配列の expiresAt フィールドを確認して、タイムアウトの有効期限をチェックします。

線形デプロイとカナリアデプロイでフックを一時停止する

線形デプロイとカナリアデプロイの場合、PRE_PRODUCTION_TRAFFIC_SHIFT で設定された一時停止フックは、トラフィックシフトの各ステップで呼び出されます。呼び出しごとに一意の hookId が生成され、それぞれに対して ContinueServiceDeployment を呼び出す必要があります。

例えば、5 つのトラフィックシフトステップと PRE_PRODUCTION_TRAFFIC_SHIFT の一時停止フックを持つ線形デプロイは、5 回 (各ステップの前に 1 回) 一時停止します。一時停止するたびに、対応する hookId を使用して ContinueServiceDeployment を個別に呼び出す必要があります。

EventBridge イベント

Amazon ECS は、ECS Hook State Change 一時停止フックがライフサイクルを通じて進行するにつれて、Detail-type で EventBridge イベントを出力します。次の表は、イベント名について説明しています。

イベント名	説明
HOOK_AWAITING_ACTION	フックは ContinueServiceDeployment を呼び出すのを待っています。
HOOK_SUCCEEDED	フックが正常に完了しました (CONTINUE で ContinueServiceDeployment を呼び出しました)。
HOOK_FAILED	フックが失敗しました (ROLLBACK で ContinueServiceDeployment を呼び出しました)。

イベント名	説明
HOOK_TIMED_OUT	ContinueServiceDeployment が呼び出される前にフックがタイムアウトしました。

次の例は、一時停止フックがアクションを待機しているときに出力される EventBridge イベントを示しています。

```
{
    "version": "0",
    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "detail-type": "ECS Hook State Change",
    "source": "aws.ecs",
    "account": "123456789012",
    "time": "2024-01-15T10:00:00Z",
    "region": "us-west-2",
    "detail": {
        "eventType": "INFO",
        "eventName": "HOOK_AWAITING_ACTION",
        "hookId": "ecs-pause-e7tK9G_WRJqNF_EOMjztDXfKenlJuEUVjsNStf4WLKw",
        "hookType": "PAUSE",
        "expiresAt": "2024-01-15T12:00:00Z",
        "lifecycleStage": "POST_TEST_TRAFFIC_SHIFT",
        "clusterArn": "arn:aws:ecs:us-west-2:123456789012:cluster/my-cluster",
        "serviceArn": "arn:aws:ecs:us-west-2:123456789012:service/my-cluster/my-
service",
        "serviceDeploymentArn": "arn:aws:ecs:us-west-2:123456789012:service-deployment/
my-cluster/my-service/EZe5RNVLH6PPzHXINuP28",
        "updatedAt": "2024-01-15T10:00:00Z"
    }
}
```

サービスデプロイの継続

Amazon ECS サービスのデプロイに一時停止ライフサイクルフックが設定されている場合、デプロイは設定されたライフサイクルステージで一時停止し、明示的なアクションを待ちます。ContinueServiceDeployment API を使用して、デプロイを次のライフサイクルステージに続行するか、前のサービスリビジョンにロールバックします。

次のアクションタイプを使用できます。

- CONTINUE – このオプションは、デプロイを次のライフサイクルステージへ進めます。
- ROLLBACK - このオプションは、サービスデプロイを 1 つ前のサービスリビジョンにロールバックします。

一時停止フックが AWAITING_ACTION ステータスのデプロイを続行できます。デプロイは IN_PROGRESS ステータスである必要があります。サービスデプロイ状態の詳細については、「Amazon ECS サービスデプロイを使用してサービス履歴を表示する」を参照してください。

前提条件

- 一時停止フックは AWAITING_ACTION ステータスである必要があります。
- DescribeServiceDeployments レスポンスの hookId が必要です。

手順

ECS Console

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
2. [クラスター] ページで、クラスターを選択します。
3. [サービス] セクションで、サービスを選択します。
4. [デプロイ] タブを選択します。
5. 一時停止フックのステータスが [アクションを待機中] と表示されていることを確認します。
6. [アクションを実行] を選択し、[続行] または [ロールバック] を選択します。

CLI

1. describe-service-deployments を呼び出して hookId を取得します。

```
aws ecs describe-service-deployments \
    --service-deployment-arn arn:aws:ecs:us-west-2:123456789012:service-
deployment/my-cluster/my-service/EZe5RNVLH6PPzHXINuP28
```

2. hookId とアクションを使用して continue-service-deployment を呼び出します。

デプロイを続行するには:

```
aws ecs continue-service-deployment \
    --hook-id ecs-pause-e7tK9G_WRJqNF_EOMjztDXfKenlJuEUVjsNStf4WLKw \
    --action CONTINUE
```

デプロイをロールバックするには:

```
aws ecs continue-service-deployment \
    --hook-id ecs-pause-e7tK9G_WRJqNF_EOMjztDXfKenlJuEUVjsNStf4WLKw \
    --action ROLLBACK
```

レスポンスには、サービスデプロイの ARN が含まれます。

```
{
    "serviceDeploymentArn": "arn:aws:ecs:us-west-2:123456789012:service-
deployment/my-cluster/my-service/EZe5RNVLH6PPzHXINuP28"
}
```

次のステップ

- デプロイの進行状況は、DescribeServiceDeployments または Amazon ECS コンソールを使用してモニタリングできます。
- 後のライフサイクルステージで追加の一時停止フックが設定されている場合、デプロイはそれらのステージで再び一時停止します。

Amazon ECS サービスデプロイの停止

デプロイが失敗してサーキットブレーカーまたは CloudWatch アラームによって検出されなかった場合に、デプロイを手動で停止することができます。次の停止タイプを使用できます。

- ロールバック – このオプションは、サービスデプロイを一つ前のサービスリビジョンにロールバックします。

  このオプションは、サービスデプロイにロールバックオプションを設定していなかった場合でも使用できます。

次のいずれかの状態のデプロイを停止できます。サービスデプロイ状態の詳細については、「Amazon ECS サービスデプロイを使用してサービス履歴を表示する」を参照してください。

- PENDING (保留中) – サービスデプロイが ROLLBACK_REQUESTED 状態に移行した後、ロールバックオペレーションが開始されます。
- IN_PROGRESS (進行中) – サービスデプロイが ROLLBACK_REQUESTED 状態に移行した後、ロールバックオペレーションが開始します。
- STOP_REQUESTED (停止リクエスト済み) – サービスデプロイの停止処理が継続します。
- ROLLBACK_REQUESTED (ロールバックのリクエスト済み) – サービスデプロイはロールバックオペレーションを続行します。
- ROLLBACK_IN_PROGRESS (ロールバックの進行中) – サービスデプロイはロールバックオペレーションを続行します。

手順

開始する前に、サービスデプロイを表示するために必要なアクセス許可を設定します。詳細については、「Amazon ECS サービスデプロイを表示するために必要なアクセス許可」を参照してください。

Amazon ECS Console

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. [クラスター] ページで、クラスターを選択します。
3. [クラスターの詳細] ページの [サービス] セクションで、サービスを選択します。

   [サービスの詳細] ページが表示されます。
4. [サービスの詳細] ページで、[デプロイ] を選択します。

   デプロイページが表示されます。
5. [進行中のデプロイ] で、[ロールバック] を選択します。次に、確認ウィンドウで、[ロールバック] を選択します。

AWS CLI

1. list-service-deployments を実行して、サービスデプロイ ARN を取得します。

   user-input を独自の値に置き換えます。

   ```
   aws ecs list-service-deployments --cluster cluster-name --service service-name
   ```

   停止するデプロイの serviceDeploymentArn に注意してください。

   ```
   {
       "serviceDeployments": [
           {
               "serviceDeploymentArn": "arn:aws:ecs:us-west-2:123456789012:service-
   deployment/cluster-name/service-name/NCWGC2ZR-taawPAYrIaU5",
               "serviceArn": "arn:aws:ecs:us-west-2:123456789012:service/cluster-
   name/service-name",
               "clusterArn": "arn:aws:ecs:us-west-2:123456789012:cluster/cluster-
   name",
               "targetServiceRevisionArn": "arn:aws:ecs:us-
   west-2:123456789012:service-revision/cluster-name/service-
   name/4980306466373577095",
               "status": "SUCCESSFUL"
           }
       ]
   }
   ```

2. stop-service-deployments を実行します。list-service-deployments から返された serviceDeploymentArn を使用します。

   user-input を独自の値に置き換えます。

   ```
   aws ecs stop-service-deployment --service-deployment-arn
    arn:aws:ecs:region:123456789012:service-deployment/cluster-name/service-
   name/NCWGC2ZR-taawPAYrIaU5 --stop-type ROLLBACK
   ```

次のステップ

サービスにどのような変更を加える必要があるかを確認し、サービスを更新します。詳細については、「Amazon ECS サービスを更新する」を参照してください。

タスクを置き換えて Amazon ECS サービスをデプロイする

ローリング更新 (ECS) デプロイタイプを使用するサービスを作成すると、Amazon ECS サービススケジューラは現在実行中のタスクを新しいタスクに置き換えます。ローリング更新中にサービスに対して Amazon ECS が追加または削除するタスク数は、サービスデプロイ設定により制御されます。

Amazon ECS は、次のパラメータを使用してタスク数を決定します。

- `minimumHealthyPercent` は、ローリングデプロイメント中またはコンテナインスタンスがドレインしているときに、サービスに対して実行され正常な状態であることが必要なタスク数の下限を、サービスに必要なタスク数の割合 (%) として表します。この値は切り上げられます。例えば、最小正常率が 50 で、必要なタスク数が 4 の場合、スケジューラは 2 つの新しいタスクを開始する前に既存のタスクを 2 つ停止できます。同様に、最小正常率が 75% で、必要なタスク数が 2 の場合、結果の値も 2 であるため、スケジューラはタスクを停止できません。
- `maximumPercent` は、ローリングデプロイメント中またはコンテナインスタンスがドレインしているときに、サービスに対して実行する必要があるタスク数の上限を、サービスに必要なタスク数の割合 (%) として表します。この値は切り捨てられます。例えば、最大パーセンテージが 200 で、目的のタスク数が 4 の場合、スケジューラは既存のタスクを 4 つ停止する前に 4 つの新しいタスクを開始できます。同様に、最大比率が 125 で、必要なタスク数が 3 の場合、結果の値も 3 であるため、スケジューラはタスクを開始できません。

ローリングデプロイ中にタスクが異常な状態になると、Amazon ECS はそれらを置き換えてサービスの `minimumHealthyPercent` を維持し、可用性を保護します。異常な状態のタスクは、それらが属するのと同じサービスリビジョンを使用して置き換えられます。これにより、ソースリビジョンの異常な状態のタスクの置換は、ターゲットリビジョンのタスクの失敗とは無関係になります。`maximumPercent` 設定で許可されている場合、スケジューラは異常な状態のタスクを停止する前に代替タスクを起動します。`maximumPercent` パラメータによって、代替タスクを先に開始できないようにスケジューラが制限されている場合、スケジューラは異常なタスクを 1 つずつ停止して容量を解放してから代替タスクを開始します。

> **Important**
>
> 最小正常率または最大正常率を設定するときは、デプロイが開始されたときにスケジューラが 1 つ以上のタスクを停止または開始できることを確認する必要があります。無効なデプロイ設定が原因でサービスのデプロイが停止している場合、サービスイベントメッセージが送信されます。詳細については、「サービス (*service-name*) は、サービスデプロイメント構成のため、デプロイメント中にタスクを停止または開始できませんでし

た。minimumHealthyPercent または maximumPercent の値を更新してから、もう一度試してください。」を参照してください。

ローリングデプロイには 2 つの方法があり、サービスデプロイが失敗したタイミングをすばやく特定できます。

- the section called “ デプロイサーキットブレーカーが障害を検出する方法”
- the section called “CloudWatch アラームがデプロイ障害を検出する方法”

これらの方法は個別に使用することも、一緒に使用することもできます。両方の方法を使用する場合、失敗に対するいずれかの方法の失敗の基準が満たされ次第、デプロイは失敗に設定されます。

使用するメソッドを決定する際は、以下のガイドラインを参考にします。

- サーキットブレーカー - タスクが開始できないときにデプロイを停止する場合は、このメソッドを使用します。
- CloudWatch アラーム - アプリケーションメトリクスに基づきデプロイを停止する場合は、このメソッドを使用します。

どちらの方法も、以前のサービスリビジョンへのロールバックをサポートしています。

ターゲットサービスリビジョンで定義したタスクの割合が正常である場合に、Amazon ECS がローリングデプロイを完了するように早期成功基準を設定できます。詳細については、「早期成功基準を使用した Amazon ECS ローリングデプロイの早期完了」を参照してください。

コンテナイメージの解決

デフォルトでは、Amazon ECS はタスク定義で指定されたコンテナイメージタグをコンテナイメージダイジェストに解決します。単一のタスクを実行および維持するサービスを作成すると、そのタスクはタスク内のコンテナイメージダイジェストを確立するために使用されます。複数のタスクを実行および維持するサービスを作成すると、デプロイ中にサービススケジューラによって開始された最初のタスクを使用して、タスク内のコンテナイメージダイジェストを確立します。

コンテナイメージダイジェストの確立を 3 回以上試行しても失敗した場合、デプロイはイメージダイジェストの解決なしで続行されます。デプロイサーキットブレーカーが有効になっている場合、デプロイも失敗し、ロールバックされます。

コンテナイメージダイジェストが確立されると、Amazon ECS はこのダイジェストを、他の対象タスクの開始や、その後のサービスの更新に使用します。この結果、サービス内のすべてのタスクで常に同じコンテナイメージが実行され、ソフトウェアのバージョン整合性が得られます。

この動作は、コンテナ定義の `versionConsistency` パラメータを使用して、タスク内のコンテナごとに設定できます。詳細については、「`versionConsistency`」を参照してください。

Note

- `1.31.0` より前のバージョンの Amazon ECS エージェントは、イメージダイジェストの解決をサポートしていません。エージェントバージョン `1.31.0`~`1.69.0` では、Amazon ECR リポジトリにプッシュされたイメージに対してのみイメージダイジェストの解決がサポートされています。エージェントバージョン `1.70.0` 以降では、すべてのイメージに対してイメージダイジェストの解決がサポートされています。
- イメージダイジェスト解決に対応した Fargate Linux プラットフォームの最低バージョンは `1.3.0` です。イメージダイジェスト解決に対応した Fargate Windows プラットフォームの最低バージョンは `1.0.0` です。
- Amazon ECS は、Amazon GuardDuty セキュリティエージェントや Service Connect プロキシなど、Amazon ECS によって管理されるサイドカーコンテナのダイジェストをキャプチャしません。
- 複数のタスクがあるサービスでコンテナイメージの解決に関連する潜在的なレイテンシーを減らすには、EC2 コンテナインスタンスで Amazon ECS エージェントバージョン `1.83.0` 以降を実行してください。潜在的なレイテンシーを回避するには、タスク定義でコンテナイメージダイジェストを指定します。
- タスク数が 0 のサービスを作成した場合、タスク数が 0 よりも大きいサービスの別のデプロイをトリガーするまで Amazon ECS はコンテナダイジェストを確立できません。
- 更新されたイメージダイジェストを確立するために、新しいデプロイを強制できます。更新されたダイジェストは新しいタスクの開始に使用され、既に実行中のタスクには影響しません。新しいデプロイの強制の詳細については、「Amazon ECS API リファレンス」の「forceNewDeployment」を参照してください。
- EC2 キャパシティプロバイダーを使用すると、最初のデプロイ中にタスクを開始するキャパシティーが不十分な場合、ソフトウェアバージョンの整合性が失敗する可能性があります。容量が制限されていてもバージョンの整合性を維持するには、デフォルトの動作に依存するのではなく、タスク定義のコンテナ設定で `versionConsistency: "enabled"`

を明示的に設定してください。これは、デプロイに進む前に、キャパシティが利用可能になるまで Amazon ECS が待機する原因になります。

Amazon ECS サービスパラメータのベストプラクティス

アプリケーションのダウンタイムを発生させないためのデプロイプロセスは次のとおりです。

1. 既存のコンテナを実行したまま、新しいアプリケーションコンテナを起動します。
2. 新しいコンテナが正常であることを確認します。
3. 古いコンテナを停止します。

デプロイ設定とクラスター内の予約されていない空きスペースの量によっては、すべての古いタスクを新しいタスクに置き換えるまでに、これを複数回行う必要がある場合があります。

数を変更するのに使用できるサービス設定オプションは 2 つあります。

- `minimumHealthyPercent`: 100% (デフォルト)

  デプロイ中にサービスが RUNNING 状態を維持する必要があるタスクの数の下限を入力します。これは、最も近い整数に切り上げられた `desiredCount` のパーセンテージです。このパラメータにより、追加のクラスターキャパシティーを使用せずにデプロイできます。
- `maximumPercent`: 200% (デフォルト)

  デプロイ中に RUNNING または PENDING 状態で許可されるサービスのタスク数の上限。これは、最も近い整数に切り捨てられた `desiredCount` のパーセンテージ (%) です。

例: デフォルト設定オプション

タスクが 6 つある次のサービスが、合計 8 つのタスクを処理できるクラスターにデプロイされているとします。デフォルトのサービス設定オプションでは、デプロイは 6 つの対象タスクの 100% を下回ることはありません。

デプロイの手順は次のとおりです。

1. 目標は 6 つのタスクを置き換えることです。
2. デフォルト設定では実行中のタスクが 6 つ必要なため、スケジューラは新しいタスクを 2 つ開始します。

現在、既存のタスクが 6 つあり、新しいタスクが 2 つあります。

3. スケジューラは既存のタスクのうちの 2 つを停止します。

   現在、既存のタスクが 4 つあり、新しいタスクが 2 つあります。

4. スケジューラはさらに新しいタスクを 2 つ開始します。

   現在、既存のタスクが 4 つあり、新しいタスクが 4 つあります。

5. スケジューラは既存のタスクのうちの 2 つをシャットダウンします。

   現在、既存のタスクが 2 つあり、新しいタスクが 4 つあります。

6. スケジューラはさらに新しいタスクを 2 つ開始します。

   現在、既存のタスクが 2 つあり、新しいタスクが 6 つあります。

7. スケジューラは最後の 2 つの既存のタスクをシャットダウンします。

   現在、新しいタスクが 6 つあります。

前述の例では、オプションにデフォルト値を使用すると、新しいタスクが開始されるたびに 2.5 分待たされます。さらに、ロードバランサーは古いタスクが停止するまで 5 分間待たなければならない場合があります。

例: **minimumHealthyPercent** の変更

minimumHealthyPercent 値を 50% に設定すると、デプロイをスピードアップできます。

タスクが 6 つある次のサービスが、合計 8 つのタスクを処理できるクラスターにデプロイされているとします。デプロイの手順は次のとおりです。

1. 目標は 6 つのタスクを置き換えることです。
2. スケジューラは既存のタスクのうちの 3 つを停止します。

   minimumHealthyPercent 値を満たす既存のタスクがまだ 3 つ実行されています。

3. スケジューラは新しいタスクを 5 つ開始します。

   既存のタスクが 3 つあり、新しいタスクが 5 つあります。

4. スケジューラは残りの 3 つの既存のタスクを停止します。

   新しいタスクが 5 つあります。

5. スケジューラは最後の新しいタスクを開始します。

   新しいタスクが 6 つあります。

例: クラスターの空き容量を変更する

追加のタスクを実行できるように、空き容量を増やすこともできます。

タスクが 6 つある次のサービスが、合計 10 個のタスクを処理できるクラスターにデプロイされているとします。デプロイの手順は次のとおりです。

1. 目標は既存のタスクを置き換えることです。
2. スケジューラは既存のタスクのうちの 3 つを停止します。

   既存のタスクが 3 つあります。
3. スケジューラは新しいタスクを 6 つ開始します。

   既存のタスクと 6 つの新しいタスクがあります。
4. スケジューラは既存のタスク 3 つを停止します。

   新しいタスクが 6 つあります。

推奨事項

タスクがしばらくアイドル状態で、使用率が高くない場合は、サービス設定オプションで次の値を使用してください。

- `minimumHealthyPercent`: 50%
- `maximumPercent`: 200%

Amazon ECS のローリング更新デプロイの作成

タスク定義の指定された数のインスタンスをクラスター内で同時に実行して維持するサービスを作成します。タスクの 1 つが失敗または停止した場合、Amazon ECS サービススケジューラはタスク定義の別のインスタンスを起動してそれを置き換えます。これは、サービスで必要な数のタスクを維持するのに役立ちます。

サービスを作成する前に、次の設定パラメータを決定します。

- タスクを分散するコンピューティングオプションが 2 つあります。
  - [capacity provider strategy] (キャパシティープロバイダー戦略) により、Amazon ECS がタスクを 1 つまたは複数のキャパシティプロバイダーに分散させます。

    Amazon ECS マネージドインスタンスでワークロードを実行する場合は、キャパシティプロバイダーの戦略オプションを使用する必要があります。
  - [起動タイプ] により、Amazon ECS は Fargate またはクラスターに登録された EC2 インスタンスのいずれかでタスクを直接起動します。

    Amazon ECS マネージドインスタンスでワークロードを実行する場合は、キャパシティプロバイダーの戦略オプションを使用する必要があります。
- `awsvpc` ネットワークモードを使用するタスク定義、またはロードバランサーを使用するように設定されたサービスには、ネットワーク設定が必要です。デフォルトでは、コンソールは、デフォルトAmazon VPC 内のすべてのサブネットおよびデフォルトセキュリティグループとともにデフォルトのAmazon VPC を選択します。
- 配置戦略、デフォルトのタスク配置戦略は、アベイラビリティーゾーン間でタスクを均等に分散します。

  サービスの高可用性を確保するために、アベイラビリティーゾーンの再調整を使用することをお勧めします。詳細については、「アベイラビリティーゾーン間での Amazon ECS サービスの調整」を参照してください。
- サービスデプロイに [Launch Type] (起動タイプ) を使用するバージョン、デフォルトではクラスター VPC のサブネットでサービスが開始されます。
- [capacity provider strategy] (キャパシティープロバイダー戦略) では、コンソールはデフォルトでコンピューティングオプションを選択します。次に、コンソールがデフォルトを選択するときに使用する順序について説明します。
  - クラスターにデフォルトのキャパシティープロバイダー戦略が定義されている場合は、その戦略が選択されます。
  - クラスターにデフォルトのキャパシティープロバイダー戦略が定義されていないが、Fargate キャパシティープロバイダーをクラスターに追加している場合は、キャパシティープロバイダーを使用するカスタム `FARGATE` キャパシティープロバイダー戦略が選択されます。
  - クラスターにデフォルトのキャパシティープロバイダー戦略が定義されていないが、クラスターに 1 つ以上の Auto Scaling グループキャパシティープロバイダーが追加されている場合は、[Use custom (アドバンスト)] オプションが選択され、戦略を手動で定義する必要があります。

  - クラスターにデフォルトのキャパシティープロバイダー戦略が定義されておらず、クラスターにキャパシティープロバイダーが追加されていない場合は、Fargate起動タイプが選択されます。
- デプロイの失敗が検出された場合のデフォルトのオプションでは、[Amazon ECS デプロイサーキットブレーカー] オプションと [失敗時のロールバック] オプションを使用します。

  詳細については、「Amazon ECS デプロイサーキットブレーカーが障害を検出する方法」を参照してください。
- Amazon ECS でサービス内の必要なタスク数を自動的に増減するかどうかを決定します。詳細については、「Amazon ECS サービスを自動的にスケールする」を参照してください。
- Amazon ECS 内で実行される他のアプリケーションにアプリケーションを接続する必要がある場合、アーキテクチャに適したオプションを決定してください。詳細については、「Amazon ECS サービスを相互接続する」を参照してください。
- Amazon ECS サーキットブレーカーを使用するサービスを作成すると、Amazon ECS はサービスデプロイとサービスリビジョンを作成します。これらのリソースを使用すると、サービス履歴に関する詳細情報を表示できます。詳細については、「Amazon ECS サービスデプロイを使用してサービス履歴を表示する」を参照してください。

  AWS CLI を使用してサービスを作成する方法については、「AWS Command Line Interfaceリファレンス」の「create-service」を参照してください。

  AWS CloudFormation を使用してサービスを作成する方法については、「AWS CloudFormation ユーザーガイド」の「AWS::ECS::Service」を参照してください。

デフォルトのオプションを使用してサービスを作成する

コンソールを使用すると、サービスをすばやく作成してデプロイできます。サービスには次の設定があります。

- クラスターに関連する VPC およびサブネットにデプロイします
- タスクを 1 つ展開します
- ローリングデプロイを使用します
- デフォルトのキャパシティプロバイダーでキャパシティプロバイダー戦略を使用します
- デプロイサーキットブレーカーを使用して障害を検出し、障害時にデプロイを自動的にロールバックするオプションを設定します。

デフォルトパラメータを使用してサービスをデプロイするには、次の手順に従います。

サービスを作成するには (Amazon ECS コンソール)

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションページで、[クラスター] を選択します。
3. [クラスター] ページで、サービスを作成するクラスターを選択します。
4. [Services] (サービス) タブから、[Create] (作成) を選択します。

   [サービスの作成] ページが表示されます。
5. [サービスの詳細] で、次の操作を行います。

   a. [タスク定義] に、使用するタスク定義ファミリーとリビジョンを入力します。

   b. [Service name] (サービス名) でサービスの名前を入力します。
6. ECS Exec を使用してサービスをデバッグするには、[設定のトラブルシューティング] で [ECS Exec をオンにする] を選択します。
7. [デプロイ設定] で、次の操作を行います。

   - [Desired tasks] (必要なタスク) で、サービス内で起動および維持するタスクの数を入力します。
8. （オプション）サービスとタスクを識別しやすくするには、[Tags] (タグ) セクションを展開し、タグを設定します。

   Amazon ECS で、新しく起動されたすべてのタスクに、クラスター名とタスク定義のタグで自動でタグ付けするには、[Turn on Amazon ECS managed tags] (Amazon ECS で管理されたタグを有効にする) を選択し、[Task definition] (タスク定義) を選択します。

   Amazon ECS で、新しく起動されたすべてのタスクに、クラスター名とサービスタグを自動でタグ付けするには、[Turn on Amazon ECS managed tags] (Amazon ECS で管理されたタグを有効にする) を選択し、[Service] (サービス) を選択します。

   タグを追加または削除します。

   - [タグを追加] [Add tag] (タグを追加) を選択し、以下を実行します。
     - [キー] にはキー名を入力します。
     - [値] にキー値を入力します。
   - [タグの削除] タグの横にある [タグの削除] を選択します。

定義済みのパラメータを使用したサービスの作成

定義済みのパラメータを使用してサービスを作成するには、次の手順に従います。

サービスを作成するには (Amazon ECS コンソール)

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. サービスを起動するリソースを決定します。

サービスの起動元	Steps
クラスター	a. [Clusters] ページで、サービスを作成するクラスターを選択します。 b. [Services] (サービス) タブから、[Create] (作成) を選択します。
タスク定義	a. [タスク定義] ページで、タスク定義の横にあるオプションボタンを選択します。 b. [デプロイ] メニューで [サービスを作成] を選択します。

   [サービスの作成] ページが表示されます。
3. [サービスの詳細] で、次の操作を行います。
   a. [タスク定義]に、使用するタスク定義を入力します。次に、[リビジョン] で、使用するリビジョンを選択します。
   b. [Service name] (サービス名) でサービスの名前を入力します。
4. [既存のクラスター] で、クラスターを選択します。

   [クラスターの作成] を選択して、新しいクラスターでタスクを実行します。

5. クラスターのインフラストラクチャ全体にタスクを分散する方法を選択します。[コンピュート設定] で、オプションを選択します。

コンピューティングオプション	Steps
キャパシティープロバイダー戦略	a. [コンピューティングオプション] で、[キャパシティープロバイダー戦略] を選択します。 b. 戦略を選択: • デフォルトのキャパシティープロバイダー戦略を使用するには、[Use cluster default] (デフォルトのクラスターを使用) を選択します。 • クラスターにデフォルトのキャパシティープロバイダー戦略がない場合、またはカスタム戦略を使用する場合は、[カスタムを使用]、[キャパシティープロバイダー戦略を追加] を選択し、[ベース]、[キャパシティプロバイダー]、[重み] を指定して、カスタムキャパシティプロバイダー戦略を定義します。 **Note** 戦略でキャパシティープロバイダーを使

コンピューティングオプション	Steps
	用するには、キャパシティープロバイダーをクラスターに関連付ける必要があります。
起動タイプ	a. [Compute options] (コンピューティングオプション) セクションで、[Launch type] (起動タイプ) を選択します。 b. [Launch type] (起動タイプ) で、起動タイプを選択します。 c. (オプション) Fargate を使用する場合は、[プラットフォームバージョン] で、使用するプラットフォームのバージョンを指定します。プラットフォームバージョンが指定されない場合、LATEST プラットフォームバージョンが使用されます。

6. ECS Exec を使用してサービスをデバッグするには、[設定のトラブルシューティング] で [ECS Exec をオンにする] を選択します。
7. [デプロイ設定] で、次の操作を行います。

   a. [Service type] (サービスタイプ) で、サービススケジューリング戦略を選択します。

      - タスク配置の制約をすべて満たすアクティブなコンテナインスタンスに、スケジューラが正確に 1 つのタスクをデプロイするには、[Daemon] (デーモン) を選択します。

- スケジューラがクラスターに必要数のタスクを配置して維持するためには、[Replica] (レプリカ) を選択します。

b. [Replica] (レプリカ) を使用した場合、[Desired tasks] (必要なタスク) に、サービス内で起動および維持するタスクの数を入力します。

c. [レプリカ] を選択した場合、Amazon ECS がアベイラビリティーゾーン間のタスクの分散をモニタリングし、不均衡が発生したときにタスクを再分散させるには、[アベイラビリティーゾーンのサービス再調整] で、[アベイラビリティーゾーンのサービス再調整] を選択します。

d. [ヘルスチェックの猶予期間]に、サービススケジューラがタスクの初回起動後に異常な Elastic Load Balancing、VPC Lattice、およびコンテナヘルスチェックを無視する時間 (秒単位) を入力します。ヘルスチェックの猶予期間値を指定しない場合、デフォルト値 0 が使用されます。

e. サービスのデプロイタイプを決定してください。[デプロイオプション] を展開し、次のパラメータを指定します。

デプロイタイプ	Steps	
[ローリング更新]	a. [Min running tasks] (実行中のタスクの最小化) の場合、デプロイ時に RUNNING の状態に保つ必要のあるサービス内のタスクの下限数をタスクの必要数のパーセント値 (最も近い整数に切り上げ) で入力します。詳細については、[Deployment configuration] (デプロイ設定) を参照してください。 b. [Max running tasks] (実行中のタスクの最大化)) には、デプロイ時に RUNNING または PENDING 状態にできるサービスのタスクの上限数を必要数のタスクのパーセント値 (最も近い整数に切り下げ) で入力します。	

f. [デプロイ設定] で、ターゲットサービスリビジョンが希望数に達する前にデプロイを完了するには、早期成功基準を設定します。Amazon ECS がソースサービスリビジョンのタスクをクリーンアップするタイミングを選択することもできます。

   i. [早期成功基準] をオンにします。

   ii. [正常率] には、Amazon ECS がデプロイを完了する前に、ターゲットサービスリビジョンで実行され、正常である必要があるタスクの割合を入力します。

   iii. [ソースサービスリビジョンのクリーンアップ] で、[ブロッキング] または [延期] を選択します。

g. Amazon ECS がデプロイの障害を検出して処理する方法を設定するには、[Deployment failure detection] (デプロイ障害検出) を展開し、オプションを選択します。

i. タスクを開始できない場合にデプロイを停止するには、[Use the Amazon ECS deployment circuit breaker] (Amazon ECS デプロイサーキットブレーカーを使用する) を選択します。

デプロイサーキットブレーカーによってデプロイが失敗状態に設定されたときに、ソフトウェアがデプロイを最後に完了したデプロイ状態に自動的にロールバックするようにするには、[失敗時のロールバック] を選択します。

ii. アプリケーションメトリクスに基づいてデプロイを停止するには、[CloudWatch アラームを使用する] を選択します。次に、[CloudWatch アラーム名] からアラームを選択します。新しいアラームを作成するには、CloudWatch コンソールに移動します。

CloudWatch アラームによってデプロイが失敗状態に設定されたときに、ソフトウェアがデプロイを最後に完了したデプロイ状態に自動的にロールバックするようにするには、[失敗時のロールバック] を選択します。

8. タスク定義で `awsvpc` ネットワークモードを使用している場合は、カスタムネットワーク設定を指定できます。これを行うには、[ネットワーキング] を展開して次の操作を実行します。

a. VPC の場合、使用する VPC を選択します。

b. [Subnets] (サブネット) で、タスクスケジューラがタスクを配置するときに考慮する VPC 内のサブネットを 1 つ以上選択します。

c. [Security groups] (セキュリティグループ) で、既存のセキュリティグループを選択することも、新しいセキュリティグループを作成することもできます。既存のセキュリティグループを使用するには、セキュリティグループを選択し、次のステップに進みます。新しいセキュリティグループを作成するには、[Create a new security group (新しいセキュリティグループの作成)] を選択します。セキュリティグループの名前、説明を指定してから、セキュリティグループのインバウンドルールを 1 つ以上追加する必要があります。

d. [Public IP] (パブリック IP) では、タスクの Elastic Network Interface (ENI) にパブリック IP アドレスを自動的に割り当てるかどうかを選択します。

AWS Fargate タスクをパブリックサブネットで実行するときに、そのタスクにパブリック IP アドレスを割り当てて、インターネットへのルートを持つようにすることができます。このフィールドを使用して EC2 タスクにパブリック IP を割り当てることはできません。詳細については、「Fargate の Amazon ECS タスクのネットワーキングオプション」および

「Amazon ECS タスクにネットワークインターフェイスを割り当てる」を参照してください。

9. (オプション) Service Connect を使用してサービスを相互接続するには、[Service Connect] を展開して以下を指定します。

   a. [Service Connect をオンにする] を選択します。

   b. [Service Connect configuration] (Service Connect 設定) で、クライアントモードを指定します。

      - サービスが名前空間内の他のサービスへの接続のみを必要とするネットワーククライアントアプリケーションを実行している場合は、[クライアント側のみ] を選択します。
      - サービスがネットワークまたは Web サービスアプリケーションを実行していて、このサービスにエンドポイントを提供し、名前空間内の他のサービスに接続する必要がある場合は、[Client and server] (クライアントとサーバー) を選択します。

   c. デフォルトのクラスター名前空間ではない名前空間を使用するには、[Namespace] (名前空間) でサービス名前空間を選択します。この場合、AWS アカウントの同じ AWS リージョンで個別に作成された名前空間、またはAWS Resource Access Manager (AWS RAM) を使用してアカウントと共有されている同じリージョンの名前空間のいずれかを選択できます。共有 AWS Cloud Map 名前空間の詳細については、「AWS Cloud Map デベロッパーガイド」の「クロスアカウント AWS Cloud Map 名前空間の共有」を参照してください。

   d. (オプション) ログ設定を指定します。[ログコレクションを使用] を選択します。デフォルトのオプションでは、コンテナログを CloudWatch Logs に送信します。その他のログドライバオプションは、AWS FireLens を使用して構成されます。詳細については、「Amazon ECS ログを AWS サービスまたは AWS Partner に送信する」を参照してください。

      以下では、各コンテナログの送信先について詳しく説明します。

      - Amazon CloudWatch — コンテナログを CloudWatch Logs に送信するようにタスクを設定します。デフォルトのログドライバーオプションが提供され、ユーザーに代わり CloudWatch ロググループを作成します。別のロググループ名を指定するには、ドライバーオプションの値を変更します。
      - [Amazon Data Firehose] — Firehose にコンテナログを送信するようタスクを設定します。Firehose 配信ストリームにログを送信するデフォルトのログドライバーオプションが提供されています。別の配信ストリーム名を指定するには、ドライバーオプションの値を変更します。

- Amazon Kinesis Data Streams — Kinesis Data Streams にコンテナログを送信するようタスクを設定します。Kinesis Data Streams のストリームにログを送信するデフォルトのログドライバーオプションが提供されています。別のストリーム名を指定するには、ドライバーオプションの値を変更します。
- Amazon OpenSearch Service — コンテナログを OpenSearch Service ドメインに送信するようタスクを設定します。ログドライバーオプションを提供する必要があります。
- Amazon S3 — Amazon S3 バケットにコンテナログを送信するようタスクを設定します。デフォルトのログドライバーオプションはデフォルトで設定されていますが、有効な Amazon S3 バケット名を指定する必要があります。

e. (オプション) アクセスログを有効にするには、次の手順に従います。

i. [アクセスログ設定] を展開します。[形式] で、[JSON] または [TEXT] を選択します。

ii. アクセスログにクエリパラメータが含まれるようにするには、[クエリパラメータを含める] を選択します。

10. (オプション) サービス検出を使用してサービスを相互接続するには、[サービス検出] を展開して次の操作を行います。

a. [サービス検出を使用] を選択します。

b. 新しい名前空間を使用するには、[名前空間の設定] で [新しい名前空間の作成] を選択し、名前空間の名前と説明を入力します。既存の名前空間を使用するには、[既存の名前空間を選択] を選択し、使用する名前空間を選択します。

c. サービスの名前や説明などのサービス検出サービス情報を入力します。

d. Amazon ECS にコンテナレベルのヘルスチェックを定期的に実行させるには、[Amazon ECS タスク状態の伝達の有効化] を選択します。

e. [DNS record type (DNS レコード型)] で、サービスに作成する DNS レコード型を選択します。Amazon ECS サービス検出は、タスク定義で指定されたネットワークモードに応じて、[A] レコードおよび [SRV] レコードのみをサポートします。これらのレコードタイプの詳細については、Amazon Route 53 デベロッパーガイドのサポートされているDNSレコードタイプを参照してください。

- サービスタスクで指定されたタスク定義が `bridge` または `host` ネットワークモードを使用する場合、SRV タイプのレコードのみがサポートされます。レコードに関連付けるコンテナ名とポートの組み合わせを選択します。
- サービスタスクで指定されたタスク定義が `awsvpc` ネットワークモードを使用する場合、A または SRV レコードタイプのいずれかを選択します。[A] を選択した場合は、次

の手順をスキップします。[SRV] を選択する場合は、サービスを検出できるポートまたはレコードに関連付けるコンテナ名とポートの組み合わせを指定します。

TTL には、レコードセットが DNS リゾルバーおよびウェブブラウザによってキャッシュされる時間を秒単位で入力します。

11. (オプション) VPC Lattice を使用してサービスを相互接続するには、[VPC Lattice] を展開して次の操作を行います。

    a. [VPC Lattice をオンにする] を選択します。

    b. インフラストラクチャロール については、インフラストラクチャロールを選択します。

    ロールを作成していない場合は、[インフラストラクチャロールの作成] を選択します。

    c. [ターゲットグループ] で、1 つまたは複数のターゲットグループを選択します。少なくとも 1 つのターゲットグループを選択する必要があります。最大 5 つのターゲットグループを選択できます。追加のターゲットグループを追加するには、[ターゲットグループの追加] を選択します。選択した各ターゲットグループの [ポート名]、[プロトコル]、および [ポート] を選択します。

    ターゲットグループを削除するには、[削除] を選択します。

    > **Note**
    >
    > - 既存のターゲットグループを追加する場合は、AWS CLI を使用する必要があります。AWS CLI を使用してターゲットグループを追加する方法については、「AWS Command Line Interface リファレンス」の「register-targets」を参照してください。
    > - VPC Lattice サービスは複数のターゲットグループを持つことができますが、各ターゲットグループは 1 つのサービスにのみ追加できます。

    d. VPC Lattice 設定を完了するには、リスナーのデフォルトアクションに新しいターゲットグループを含めるか、VPC Lattice コンソール内の既存の VPC Lattice サービスのルールに新しいターゲットグループを含めます。詳細については、「Listener rules for your VPC Lattice service」を参照してください。

12. (オプション) サービスのロードバランサーを設定するには、[Load Balancing] (負荷分散) セクションを展開します。

ロードバランサーを選択します。

このロードバランサーを使用するには	この操作を行います	
Application Load Balancer	a. [Load balancer type] で、[Application Load Balancer] を選択します。 b. 選択新しいロードバランサーを作成するをクリックして新しいApplication Load Balancer を作成するか、既存のロードバランサーを使用する既存の Application Load Balancer を選択します。 c. [Load balancer name] (ロードバランサー名) では、一意な名前を入力します。 d. [Choose container to load balance] (負荷分散するコンテナを選択) で、サービスをホストするコンテナを選択します。 e. [Listener] (リスナー) で、Application Load Balancer が接続リクエストをリッスンするポートとプロトコルを入力します。デフォルトでは、ロードバランサーはポート 80 と HTTP を使用するように設定されます。 f. [Target group name] (ターゲットグループ名) で	

このロードバランサーを使用するには	この操作を行います	
	は、Application Load Balancer がリクエストをルーティングするターゲットグループの名前とプロトコルを入力します。デフォルトでは、ターゲットグループは、タスク定義で定義されている最初のコンテナにリクエストをルーティングします。 g. [登録の遅延] には、ロードバランサーがターゲットの状態を UNUSED に変更するまでの秒数を入力します。デフォルトは 300 秒です。 h. [Health check path] (ヘルスチェックパス) では、Application Load Balancer とコンテナ間の接続の正常性を検証するために、Application Load Balancer が定期的にリクエストを送信するコンテナ内に存在するパスを入力します。デフォルトはルートディレクトリ (/)。	

このロードバランサーを使用するには	この操作を行います
Network Load Balancer	a. [Load balancer type] (ロードバランサータイプ) では、[Network Load Balancer] を選択します。 b. [Load Balancer] (ロードバランサー) では、既存の Network Load Balancer を選択します。 c. [Choose container to load balance] (負荷分散するコンテナを選択) で、サービスをホストするコンテナを選択します。 d. [Target group name] (ターゲットグループ名) では、Network Load Balancer がリクエストをルーティングするターゲットグループの名前とプロトコルを入力します。デフォルトでは、ターゲットグループは、タスク定義で定義されている最初のコンテナにリクエストをルーティングします。 e. [登録の遅延] には、ロードバランサーがターゲットの状態を UNUSED に変更するまでの秒数を入力します。デフォルトは 300 秒です。

このロードバランサーを使用するには	この操作を行います	
	f. [Health check path] (ヘルスチェックパス) では、Application Load Balancer とコンテナ間の接続の正常性を検証するために、Network Load Balancer が定期的にリクエストを送信するコンテナ内に存在するパスを入力します。デフォルトはルートディレクトリ (/)。	

13. (オプション) サービスの自動スケーリングを設定するには、[サービスの自動スケーリング] を展開し、次のパラメータを指定します。トラフィックフローからの過去のロードデータを調べる予測自動スケーリングを使用するには、サービスを作成した後にそれを設定します。詳細については、「履歴パターンを使用して予測スケーリングで Amazon ECS サービスをスケールする」を参照してください。

    a. サービスの自動スケーリングを使用するには、[Service auto scaling] (サービスの自動スケーリング) を選択します。

    b. [タスクの最小数] に、サービスの自動スケーリングで使用するタスクの下限数を入力します。必要な数がこの数を下回ることはありません。

    c. [タスクの最大数] に、サービスの自動スケーリングで使用するタスクの上限数を入力します。必要な数がこの数を超えることはありません。

    d. ポリシータイプを選択します。[スケーリングポリシータイプ] で、次のいずれかのオプションを選択します。

使用するポリシータイプ	この操作を行います	
ターゲット追跡	a. [スケーリングポリシータイプ] で [ターゲットの追跡] を選択します。	

使用するポリシータイプ	この操作を行います	
	b. [Policy Name] (ポリシー名) にこのポリシーの名前を入力します。 c. [ECS サービスメトリクス] で次のいずれかのメトリクスを選択します。 • [ECSServiceAverageCPUUtilization] — サービスの平均 CPU 使用率。 • [ECSServiceAverageMemoryUtilization] — サービスのメモリ平均使用率。 • [ALBRequestCountPerTarget] - Application Load Balancer ターゲットグループ内のターゲットごとに完了したリクエストの数。 d. [Target value] (ターゲット値) には、選択したメトリックに対してサービスが保持する値を入力します。 e. [スケールアウトのクールダウン期間] には、スケールインアクティビティが完了してから別のスケールアウトアクティビティが開始されるまでの時間 (秒) を入力します。	

使用するポリシータイプ	この操作を行います	
	f. [スケールインのクールダウン期間] には、スケールインアクティビティが完了してから別のスケールインアクティビティが開始されるまでの時間 (秒) を入力します。 g. ポリシーがスケールインアクティビティを実行しないようにするには、[Turn off scale-in] (スケールインをオフにする) を選択します。 h. • (オプション) トラフィックの増加に合わせてスケーリングポリシーをスケールアウトさせたいが、トラフィックが減少してもスケールインする必要がない場合は、[スケールインをオフにする] を選択します。	

使用するポリシータイプ	この操作を行います	
ステップスケーリング	a. [スケーリングポリシータイプ] で [ステップスケーリング] を選択します。 b. [ポリシー名] で、このポリシー名を入力します。 c. [アラーム名] に、アラームの一意の名前を入力します。 d. [Amazon ECS サービスメトリクス] で、アラームに使用するサービスメトリクスを選択します。 e. [統計] で、アラーム統計を選択します。 f. [期間] で、アラームの期間を選択します。 g. [アラーム条件] で、選択したメトリクスを定義されたしきい値と比較する方法を選択します。 h. [メトリクスを比較するためのしきい値] と [アラームを開始する評価期間] に、アラームに使用するしきい値としきい値を評価する期間を入力します。 i. [スケーリングアクション]で、次の手順を実行します。 • [アクション] には、サービスに対してタス	

使用するポリシータイプ	この操作を行います	
	クを追加する、削除する、または特定の必要数を設定するかを選択します。 • タスクの追加または削除を選択した場合は、スケーリングアクションの開始時に追加または削除するタスクの数 (または既存のタスクの割合) を [値] に入力します。希望の数を設定することを選択した場合は、タスクの数を入力します。[タイプ] で、[値] が整数であるか、既存の必要なカウントのパーセント値であるかを選択します。 • [下限] と [上限] に、ステップスケーリング調整の下限と上限を入力します。デフォルトでは、[ポリシーを追加] の下限はアラームしきい値であり、上限は正 (+) の無限大です。デフォルトでは、[ポリシーを削除] の上限はアラームしきい値であり、下限は負 (-) の無限大です。	

使用するポリシータイプ	この操作を行います	
	• (オプション) さらにスケーリングオプションを追加します。[新しいスケーリングアクションの追加] を選択し、[アクションのスケーリング] の手順を繰り返します。 • [クールダウン期間] には、前回のスケーリングアクティビティが有効になるまで待機する時間を秒単位で入力します。追加ポリシーの場合、スケールアウトアクティビティの終了後、スケーリングポリシーによってスケールインアクティビティがブロックされ、一度にスケールアウトできるタスクの数が制限されます。削除ポリシーの場合、これはスケールインアクティビティの後、別のスケールインアクティビティが開始されるまでに経過する必要がある時間です。	

14. (オプション) デフォルト以外のタスク配置戦略を使用するには、[Task Placement] (タスクの配置) を展開し、以下のオプションから選択します。

    詳細については、「Amazon ECS がタスクをコンテナインスタンスに配置する方法」を参照してください。

- [AZ バランススプレッド] - アベイラビリティーゾーン間およびアベイラビリティーゾーン内のコンテナインスタンス間でタスクを分散します。
- [AZ Balanced BinPack] - 利用可能な最小メモリでアベイラビリティーゾーン間およびコンテナインスタンス間でタスクを分散します。
- [ビンパック] - CPU またはメモリの最小利用可能量に基づいてタスクを配置します。
- [ホストごとに 1 つのタスク] - 各コンテナインスタンスのサービスから最大 1 タスクを配置します。
- [カスタム] - 独自のタスク配置戦略を定義します。

[Custom] (カスタム) を選択した場合、タスクを配置するアルゴリズムと、タスク配置時に考慮されるルールを定義します。

- [Strategy] (方針) にとって [Type] (タイプ) そして [Field] (フィールド) で、アルゴリズムとアルゴリズムに使用するエンティティを選択します。

  最大 5 個の戦略を入力できます。
- [制約] の [タイプ] および [式] で、制約のルールと属性を選択します。

  例えば、T2 インスタンスにタスクを配置する制約を設定するには、[Expression] (表現) で、[attribute:ecs.instance-type =~ t2.*] と入力します。

  最大 10 個の制約を入力できます。

15. デプロイ時の設定と互換性のあるデータボリュームをタスクで使用する場合は、[ボリューム] を拡張してボリュームを設定できます。

    ボリューム名とボリュームタイプはタスク定義リビジョンを作成するときに設定され、サービスの作成時には変更できません。ボリューム名とタイプを更新するには、新しいタスク定義リビジョンを作成し、その新しいリビジョンを使用してサービスを作成する必要があります。

このボリュームタイプを設定するには	この操作を行います	
Amazon EBS	a. [EBS ボリュームタイプ] には、タスクにアタッチする EBS ボリュームのタイプを選択します。	

このボリュームタイプを設定するには	この操作を行います	
	b. [サイズ (GiB)] には、ボリュームサイズの有効な値をギビバイト (GiB) 単位で入力します。最小で 1 GiB、最大で 16,384 GiB のボリュームサイズを指定できます。スナップショット ID を指定しない限り、この値は必須です。 c. [IOPS] には、ボリュームが提供する入力/出力オペレーションの数 (IOPS) の最大数を入力します。この値は `io1`、`io2`、および `gp3` ボリュームタイプでのみ設定できます。 d. [スループット (MiB/秒)] には、ボリュームが提供すべきスループットをメビバイト/秒 (MiBps または MiB/s) で入力します。この値は `gp3` ボリュームタイプでのみ設定できます。 e. [スナップショット ID] には、既存の Amazon EBS ボリュームスナップショットを選択するか、スナップショットからボリュームを作成する場合はスナップショットの ARN を入力します。新し	

このボリュームタイプを設定するには	この操作を行います	
	い空のボリュームは、スナップショット ID を選択または入力しないで作成することもできます。 f. スナップショット ID を指定すると、ボリューム初期化レート (MiB/秒) を指定できます。100 から 300 の値 (MiB/秒単位) を入力して、スナップショット ID を使用してボリュームを作成する際のスナップショットからのデータ読込み速度を指定します。 g. [ファイルシステムのタイプ] では、ボリューム上のデータストレージおよび取得に使用するファイルシステムの種類を選択します。オペレーティングシステムのデフォルトまたは特定のファイルシステムタイプのいずれかを選択できます。Linux のデフォルトは XFS です。スナップショットから作成されたボリュームでは、スナップショットの作成時にボリュームが使用していたのと同じファイルシステムタイプを指定する必要があります。ファ	

このボリュームタイプを設定するには	この操作を行います	
	イルシステムのタイプが一致しない場合、タスクは開始できません。 h. [インフラストラクチャロール] には、Amazon ECS がタスクの Amazon EBS ボリュームを管理できるようにするために必要な権限を持つ IAM ロールを選択します。`AmazonECSInfrastructureRolePolicyForVolumes` 管理ポリシーをロールにアタッチすることも、このポリシーをガイドとして使用して、特定のニーズを満たすアクセス権限を持つ独自のポリシーを作成してアタッチすることもできます。必要なアクセス権限の詳細については、「Amazon ECS インフラストラクチャ IAM ロール」を参照してください。 i. デフォルト設定で Amazon EBS 暗号化を使用する場合は、[暗号化] で [デフォルト] を選択します。アカウントにデフォルトで暗号	

このボリュームタイプを設定するには	この操作を行います	
	化が設定されている場合、ボリュームは設定で指定されている AWS Key Management Service (AWS KMS) キーで暗号化されます。[デフォルト] を選択し、Amazon EBS のデフォルト暗号化がオンになっていない場合、ボリュームは暗号化されません。 [カスタム] を選択した場合は、ボリューム暗号化に任意の AWS KMS key を指定できます。 [なし] を選択すると、デフォルトで暗号化が設定されている場合や、暗号化されたスナップショットからボリュームを作成した場合を除き、ボリュームは暗号化されません。 j. [暗号化] に [カスタム] を選択した場合は、使用する AWS KMS key を指定する必要があります。[KMS キー] には、AWS KMS key を選択するか、キーの ARN を入力します。対称型のカスタマー管理キーを使用してボリュームを暗号化	

このボリュームタイプを設定するには	この操作を行います	
	する場合は、AWS KMS key ポリシーに適切な権限が定義されていることを確認してください。詳細については、「Amazon EBS ボリュームのデータ暗号化」を参照してください。 k. (オプション) [タグ] では、タスク定義またはサービスからタグを伝達するか、独自のタグを指定することで、Amazon EBS ボリュームにタグを追加できます。 タスク定義からタグを伝達する場合は、[タグの伝播元] で [タスク定義] を選択します。サービスからタグを伝達する場合は、[タグの伝播元] で [サービス] を選択します。[伝播しない] を選択した場合、または値を選択しなかった場合、タグは伝播されません。 独自のタグを指定する場合は、[タグの追加] を選択し、追加する各タグのキーと値を指定します。 Amazon EBS ボリュームのタグ付けの詳細につい	

このボリュームタイプを設定するには	この操作を行います
	ては、「Amazon EBS ボリュームのタグ付け」を参照してください。

16. ECS Exec を使用してサービスをデバッグするには、[設定のトラブルシューティング] で [ECS Exec をオンにする] を選択します。
17. （オプション）サービスとタスクを識別しやすくするには、[Tags] (タグ) セクションを展開し、タグを設定します。

    新しく起動したすべてのタスクに対して、Amazon ECS がクラスター名とタスク定義タグで自動的にタグ付けするようにするには、[Amazon ECS マネージドタグを有効にする] を選択し、[タグの伝播元] で [タスク定義] を選択します。

    新しく起動したすべてのタスクに対して、Amazon ECS がクラスター名とサービスタグで自動的にタグ付けするようにするには、[Amazon ECS マネージドタグを有効にする] を選択し、[タグの伝播元] で [サービス] を選択します。

    タグを追加または削除します。

    - [タグを追加] [Add tag] (タグを追加) を選択し、以下を実行します。
      - [キー] にはキー名を入力します。
      - [値] にキー値を入力します。
    - [タグの削除] タグの横にある [タグの削除] を選択します。
18. [作成] を選択します。

次のステップ

以下は、サービスを作成した後の追加アクションです。

- トラフィックフローからの過去のロードデータを調べる予測自動スケーリングを設定します。詳細については、「履歴パターンを使用して予測スケーリングで Amazon ECS サービスをスケールする」を参照してください。
- Amazon ECS サーキットブレーカーのサービスのデプロイを追跡し、サービス履歴を表示します。詳細については、「Amazon ECS サービスデプロイを使用してサービス履歴を表示する」を参照してください。

早期成功基準を使用した Amazon ECS ローリングデプロイの早期完了

ローリングデプロイ戦略を使用するサービスを更新すると、Amazon ECS はソースサービスリビジョンのタスクをターゲットサービスリビジョンの新しいタスクに置き換えるデプロイを開始します。Amazon ECS は、以下のすべての条件を満たした場合にデプロイを完了します。

- ターゲットサービスリビジョンが必要数の 100% に達し、すべてのタスクが実行中かつ正常な状態である。
- デプロイサーキットブレーカーまたは CloudWatch アラームがロールバックをトリガーしていない。
- アラームベースのロールバックを使用している場合、ベイク時間が経過している。
- ソースサービスリビジョンのタスクがクリーンアップされている。

早期成功基準を使用すると、ワークロードの信頼度と運用上のニーズに基づいて、Amazon ECS がデプロイを完了するタイミングを設定できます。

早期成功基準は、以下のシナリオで使用できます。

- ターゲットサービスリビジョンの正常なタスクの割合がデプロイを完了するのに十分であると確信しており、後続のデプロイ、CI/CD パイプライン、その他の依存オペレーションを続行できるように、より早くデプロイを完了したい場合。
- サービスは、GPU アクセラレーション推論ワークロードなどの特殊な容量または制約された容量で実行されており、ハードウェアの可用性により一部のタスクの起動に時間がかかります。
- サービスには、ソースサービスリビジョンにドレインが遅いタスクがあり、存続期間の長い接続を持つタスクなど、デプロイをオープンのままにする可能性があります。
- デプロイサーキットブレーカーまたは CloudWatch アラームのロールバックを使用して、ターゲットサービスリビジョンが定義したヘルスレベルに達するまでデプロイを保護し、それ以降はロールバックを行わないようにすることができます。

デプロイ設定の次の設定を使用して条件を定義します。

正常率

`earlySuccessCriteria.healthyPercent` は、Amazon ECS がデプロイを完了する前にターゲットサービスリビジョンで実行され、正常である必要があるタスクの数を、サービスのタスクの必要数に対するパーセントとして表します。この値は切り上げられます。たとえば、正常率が 90 で必要数が 100 の場合、Amazon ECS は 90 個のタスクが正常になった後にデプロイを完了します。正

常率が 50 で必要数が 3 の場合、Amazon ECS は 2 つのタスクが正常になった後にデプロイを完了します。

タスクは、サービス用に設定したヘルスチェックに合格すると正常とみなされます。

Amazon ECS は、通常のサービススケーリングを通じて、デプロイ外のターゲットサービスリビジョンで残りのタスクを起動します。Amazon ECS がデプロイを完了すると、デプロイサーキットブレーカーと CloudWatch アラームのロールバックは適用されなくなります。

正常率は、サービスの `minimumHealthyPercent` と 100 の間である必要があります。

ソースサービスリビジョンのクリーンアップ

`earlySuccessCriteria.sourceServiceRevisionCleanup` は、Amazon ECS がソースサービスリビジョンからタスクをクリーンアップするタイミングを決定します。これには、`BLOCKING` と `DEFERRED` の 2 つの値があります。

`BLOCKING` では、デプロイは次の順序で続行されます。

1. 正常率分のタスクが、ターゲットサービスリビジョンで実行中かつ正常な状態になっています。
2. アラームベースのロールバックを使用している場合、ベイク時間が経過している。
3. Amazon ECS は、ソースサービスリビジョンのタスクをクリーンアップします。
4. Amazon ECS はデプロイを完了します。

`DEFERRED` では、デプロイは次の順序で続行されます。

1. 正常率分のタスクが、ターゲットサービスリビジョンで実行中かつ正常な状態になっています。
2. アラームベースのロールバックを使用している場合、ベイク時間が経過している。
3. Amazon ECS はデプロイを完了します。
4. Amazon ECS は、デプロイの外部で、ソースサービスリビジョンのタスクをクリーンアップします。

`DEFERRED` では、Amazon ECS はソースサービスリビジョンのタスクを最大 2 週間クリーンアップしようとします。

ソースサービスリビジョンのタスクがデプロイをオープンのままにせずに実行し続ける必要がある、存続期間の長い接続またはタスクのスケールイン保護を持つサービスには、`DEFERRED` を使用しま

す。これは、ソースサービスリビジョンのドレインがロングテールオペレーションであり、CI/CD ツールに時間制限がある場合に役立ちます。Amazon ECS は、正常率が満たされると、デプロイを完了し、ソースサービスリビジョンのタスクはデプロイの外部でドレインされます。

Amazon ECS による初期成功基準の評価方法

Amazon ECS は、次のすべての条件が満たされた後、初期成功基準を使用するデプロイを完了します。

- ターゲットサービスリビジョンで実行中の正常なタスクの数が、希望するカウントの正常率以上 (切り上げ) であること。
- デプロイサーキットブレーカーまたは CloudWatch アラームがロールバックをトリガーしていない。
- アラームベースのロールバックを使用している場合、ベイク時間が経過している。
- ソースサービスリビジョンのクリーンアップが `BLOCKING` の場合、ソースサービスリビジョンのタスクはクリーンアップされます。

Amazon ECS は、ターゲットサービスリビジョンで少なくとも 1 つのタスクを起動し、正常率を評価する前にそのタスクが正常になるまで待機します。

正常タスク数の例

必要数	正常率	必要な正常タスク数 (切り上げ)
2	50	1
3	50	2
10	80	8
100	90	90
10	100	10

早期成功基準の設定

コンソールまたは AWS CLI を使用して、新規または既存のサービスの早期成功基準を設定できます。

コンソール

サービスの作成または更新フローの [デプロイ設定] で、早期成功基準をオンにします。[正常率] の値を入力し、ソースサービスリビジョンのクリーンアップオプションとして [ブロッキング] または [延期] のいずれかを選択します。

AWS CLI

サービスを作成または更新するときに、`--deployment-configuration` パラメータで `earlySuccessCriteria` の値を設定します。

```
aws ecs update-service \
    --cluster MyCluster \
    --service MyService \
    --deployment-configuration '{
        "strategy": "ROLLING",
        "earlySuccessCriteria": {
            "enable": true,
            "healthyPercent": 90,
            "sourceServiceRevisionCleanup": "BLOCKING"
        }
    }'
```

観察される結果

- `DescribeServiceDeployments` は、デプロイステータスと設定された早期成功基準を返します。デプロイの進行中、タスク数はリアルタイムで更新されます。デプロイが完了すると、タスク数はデプロイ完了時点のスナップショットになります。これは、`BLOCKING` と `DEFERRED` のいずれにも適用されます。
- サービスのライブタスク数を表示するには、`DescribeServices` を使用します。
- `ListServiceDeployments` でフィルタリングする際にステータスが `SUCCESSFUL` の場合、早期に完了したデプロイが含まれます。
- デプロイ状態変更イベントと AWS CloudTrail レコードは、`IN_PROGRESS` から `SUCCESSFUL` への標準ライフサイクルを示しています。早期成功基準では、新しいデプロイステータスは追加されません。

考慮事項

早期成功基準を使用する前に、以下の考慮事項を確認してください。

ロールバックモニタリングはデプロイ中に適用されます

デプロイサーキットブレーカーと CloudWatch アラームのロールバックは、デプロイの障害検出メカニズムです。デプロイの進行中にロールバックできます。Amazon ECS がデプロイを完了すると、Amazon ECS が通常のサービススケーリングを通じて残りのタスクを起動している間も含め、これらのメカニズムはサービスをロールバックしなくなります。デプロイの完了後に停止することはできません。正常率を、ターゲットサービスリビジョンが正常であると確信できる割合に設定します。

DEFERRED クリーンアップ

DEFERRED を使用すると、Amazon ECS はデプロイの完了後にソースサービスリビジョンのタスクをクリーンアップします。ソースサービスリビジョンのタスクが、デプロイ後 2 週間以上、タスクのスケールイン保護または他のメカニズムによって保護されている場合、Amazon ECS はそれらをクリーンアップできません。DescribeServices を使用して、ソースサービスリビジョンのタスクをモニタリングします。

Amazon ECS ブルー/グリーンデプロイ

ブルー/グリーンデプロイは、ブルーおよびグリーンという 2 つの同一の本番環境を実行することで、ダウンタイムおよびリスクを軽減するリリース方法です。Amazon ECS ブルー/グリーンデプロイを使用すると、本番トラフィックをルーティングする前に、新しいサービスリビジョンを検証できます。このアプローチによって変更をデプロイするより安全な方法が実現され、必要に応じてすばやくロールバックできます。

利点

ブルー/グリーンデプロイの使用には、次のメリットがあります。

- 本番トラフィックでテストしてから本番に切り替えることで、リスクを軽減します。本番トラフィックをルーティングする前に、テストトラフィックを使用して新しいデプロイを検証できます。
- ダウンタイムのないデプロイ。本番環境はデプロイプロセス全体で引き続き利用でき、サービスの継続的な利用可能性を実現します。
- 問題が検出された場合、簡単にロールバックできます。グリーンデプロイで問題が発生した場合、サービス中断が継続されることなく、ブルーデプロイにすばやく戻ることができます。
- 管理されたテスト環境。完全にデプロイする前に実際のトラフィックパターンで新機能をテストするため、グリーン環境では隔離されたスペースが提供されます。

- 予測可能なデプロイプロセス。定義されたライフサイクルステージによる構造化されたアプローチにより、デプロイの整合性および信頼性が向上します。
- ライフサイクルフックによる自動検証。デプロイのさまざまな段階で自動テストを実装し、機能性を検証できます。

用語

Amazon ECS ブルー/グリーンデプロイの条件は次のとおりです。

- ベイク時間 – 本番トラフィックが移行した後に、ブルーおよびグリーンのサービスリビジョンの両方が同時に実行される期間。
- ブルーデプロイ – 置き換える予定の、現在の本番サービスリビジョン。
- グリーンデプロイ – デプロイする新しいサービスリビジョン。
- ライフサイクルステージ –「本番トラフィックの移行後」など、デプロイオペレーションの一連のイベント。
- ライフサイクルフック – 特定のライフサイクルステージの Lambda 関数または一時停止ポイント。Lambda フックは、カスタムコードを実行するために定義した Lambda 関数を呼び出します。一時停止フックはデプロイを一時停止し、`ContinueServiceDeployment` を呼び出して続行を待ちます。
- リスナー – 設定したプロトコルおよびポートを使用し、接続リクエストを確認する Elastic Load Balancing リソース。リスナーに対して定義したルールにより、Amazon ECS が登録済みターゲットにリクエストをルーティングする方法が決定します。
- ルール – リスナーに関連付けられた Elastic Load Balancing リソース。ルールはリクエストのルーティング方法を定義し、アクション、条件、優先度で構成されます。
- ターゲットグループ – リクエストを 1 つ以上の登録済みターゲットにルーティングするために使用される Elastic Load Balancing リソース (EC2 インスタンスなど)。リスナーを作成するときは、デフォルトアクションのターゲットグループを指定します。トラフィックは、リスナー規則で指定されたターゲットグループに転送されます。
- トラフィック移行 – Amazon ECS がブルーデプロイからグリーンデプロイにトラフィックを移行するために使用するプロセス。ロードバランサーまたは Service Connect を使用するサービスの場合、Amazon ECS はトラフィックシフトを自動で管理します。ヘッドレスサービスの場合、Amazon ECS はブルータスクをグリーンタスクに置き換えますが、トラフィックシフトは管理しません。

考慮事項

デプロイタイプを選択する際、次の内容を考慮してください。

- リソースの使用量: ブルー/グリーンデプロイではブルーおよびグリーンのサービスリビジョンの両方が一時的に実行されるため、デプロイ中にリソースの使用量が 2 倍になる場合があります。
- デプロイモニタリング: ブルー/グリーンデプロイではデプロイステータスの詳細な情報が提供されるため、デプロイプロセスの各ステージをモニタリングできます。
- ロールバック: ブルーリビジョンがベイク時間が終了するまで実行が継続されるため、問題が検出された場合、ブルー/グリーンデプロイでは前のバージョンへのロールバックが容易になります。
- Network Load Balancer ライフサイクルフック: ブルー/グリーンデプロイに Network Load Balancer を使用する場合は、TEST_TRAFFIC_SHIFT および PRODUCTION_TRAFFIC_SHIFT ライフサイクルステージにさらに 10 分を要します。これは、Amazon ECS がトラフィックを移行しても安全であることを確認するためです。

Amazon ECS ブルー/グリーンサービスのデプロイワークフロー

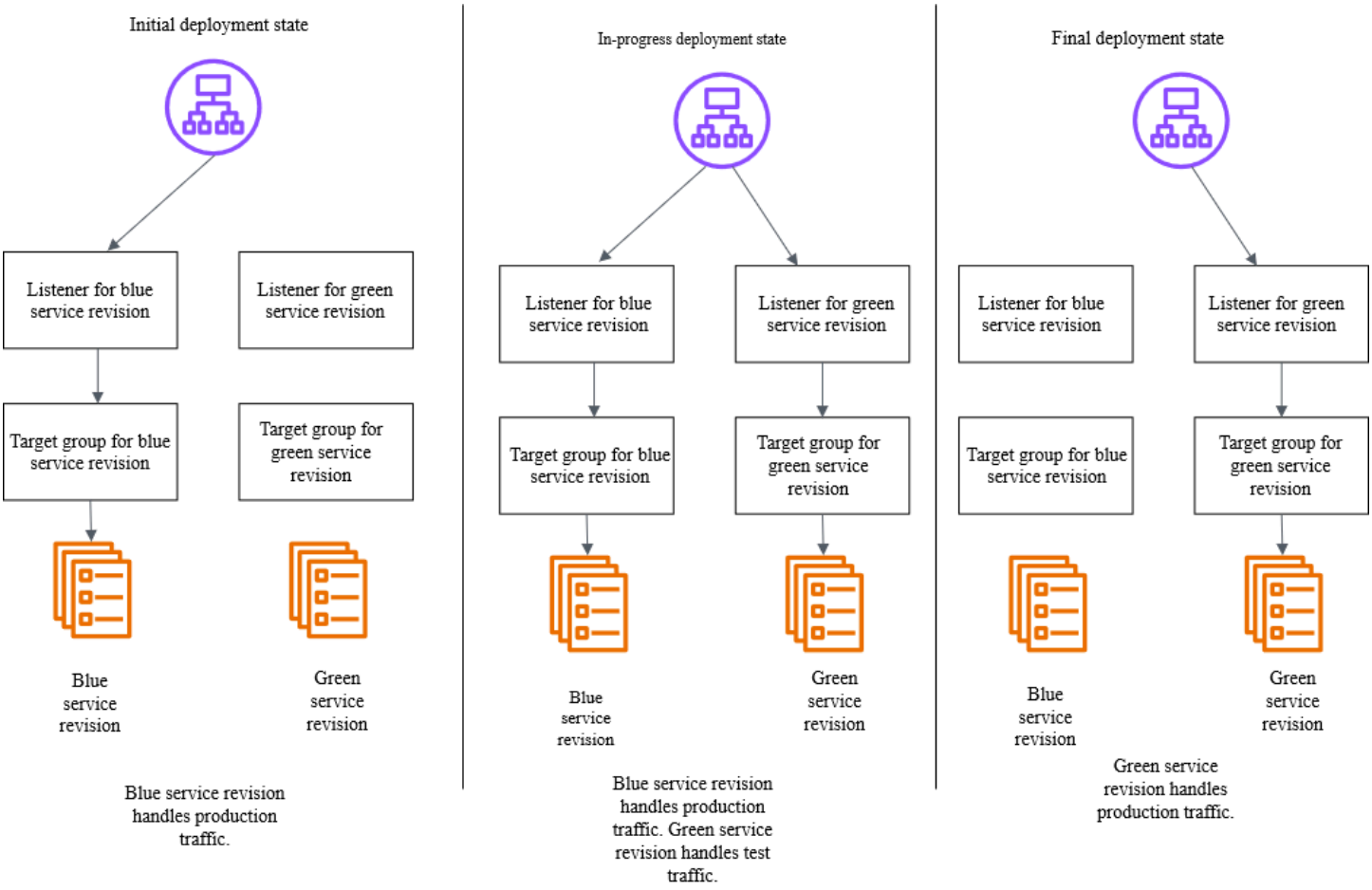
Amazon ECS ブルー/グリーンデプロイプロセスは構造化されたアプローチに従っており、安全かつ信頼性の高いアプリケーション更新を実現する 6 つの異なるフェーズで構成されます。各フェーズは、アプリケーションを現在のバージョン (ブルー) から新しいバージョン (グリーン) に検証および移行するためにそれ特有の目的を果たします。

1. 準備フェーズ: 既存のブルー環境と一緒にグリーン環境を作成します。新しいサービスリビジョンのプロビジョニングやターゲットグループの準備が含まれます。
2. デプロイフェーズ: 新しいサービスリビジョンをグリーン環境にデプロイします。Amazon ECS は更新されたサービスリビジョンを使用して新しいタスクを起動し、ブルー環境は引き続き本番トラフィックを処理します。
3. テストフェーズ: テストトラフィックルーティングを使用してグリーン環境を検証します。Application Load Balancer によってテストリクエストがグリーン環境に送信され、本番トラフィックはブルー環境の状態にとどまります。
4. トラフィック移行のフェーズ: 設定されたデプロイ戦略に基づき、本番トラフィックをブルーからグリーンに移行します。このフェーズには、モニタリングおよび検証チェックポイントが含まれます。
5. モニタリングフェーズ: ベイク期間中のアプリケーションの正常性、パフォーマンスメトリクス、アラーム状態がモニタリングされます。ロールバックオペレーションは問題が検出されると開始されます。

6. 完了フェーズ: 設定に応じて、ブルー環境を終了するか、潜在的なロールバックシナリオに合わせて維持することで、デプロイを確定します。

ワークフロー

次の図は、包括的なブルー/グリーンデプロイのワークフローであり、Amazon ECS および Application Load Balancer 間の相互作用を示しています。



拡張デプロイワークフローには、次の詳細なステップが含まれます。

1. 初期状態: ブルーサービス (現在の本番用) は本番トラフィックの 100% を処理します。Application Load Balancer には、正常なブルータスクを含むブルーターゲットグループにすべてのリクエストをルーティングするルールを含む 1 つのリスナーがあります。
2. グリーン環境プロビジョニング: Amazon ECS により、更新されたタスク定義を使用して新しいタスクが作成されます。これらのタスクは新しいグリーンターゲットグループに登録されますが、最初はトラフィックを受信しません。

3. ヘルスチェックの検証: Application Load Balancer がグリーンタスクにヘルスチェックを実行します。グリーンタスクがヘルスチェックに合格した場合にのみ、デプロイは次のフェーズに進行します。
4. テストトラフィックのルーティング: 設定された場合、Application Load Balancer のリスナールールは特定のトラフィックパターン (テストヘッダーを持つリクエストなど) が検証のためにグリーン環境にルーティングされ、本番トラフィックはブルー環境の状態にとどまります。これは本番トラフィックを処理するリスナーによって制御され、リクエスト属性に基づく異なるルールを使用します。
5. 本番トラフィック移行: デプロイ設定に基づき、トラフィックはブルーからグリーンに移行します。ECS ブルー/グリーンデプロイは、トラフィックの 100% がブルー環境からグリーン環境に移動する即時 (すべてを一度に) 移行です。Application Load Balancer により、重みに基づいてブルーおよびグリーンのターゲットグループ間のトラフィック分散を制御するリスナールールを持つ 1 つのリスナーが使用されます。
6. モニタリングと検証: トラフィック移行中の全過程において、Amazon ECS は CloudWatch メトリクス、アラーム状態、デプロイの正常性をモニタリングします。問題が検出された場合、自動ロールバックトリガーが発動します。
7. ベイク期間: 本番トラフィックが移行した後、ブルーおよびグリーンのサービスリビジョンの両方が同時に実行される期間。
8. ブルー環境の終了: トラフィックの移行および検証が正常に処理されたら、ブルー環境が終了してクラスターリソースが解放されるか、迅速にロールバックできるように維持されます。
9. 最終状態: グリーン環境が新しい本番環境になり、トラフィックの 100% を処理します。デプロイは正常処理としてマークされます。

デプロイのライフサイクルステージ

ブルー/グリーンデプロイプロセスは、個別のライフサイクルステージ (「本番トラフィックの移行後」など、デプロイオペレーションの一連のイベント) を経て進行していき、それぞれのステージに特定の役割および検証チェックポイントがあります。これらのステージを理解すると、デプロイの進行状況をモニタリングして問題を効果的にトラブルシューティングできます。

各ライフサイクルステージは最大 24 時間継続できます。値を 24 時間未満にとどめることをお勧めします。非同期プロセスがフックをトリガーするために時間が必要になるからです。システムがタイムアウトしてデプロイに失敗し、ステージが 24 時間に達した後にロールバックが開始されます。CloudFormation デプロイには追加のタイムアウト制限があります。24 時間のステージ制限は有効性が維持されますが、CloudFormation によってデプロイ全体に 36 時間の制限が実施されま

す。CloudFormation によってデプロイが失敗し、36 時間以内にプロセスが完了しない場合はロールバックが開始されます。

一時停止フックの場合、タイムアウトを最大 20,160 分 (14 日間) まで設定できます。デプロイ全体のタイムアウトは 30 日です。

ライフサイクルのステージ	説明	ライフサイクルフックにこのステージを使用しますか?
RECONCILE_SERVICE	このステージは、複数のサービスリビジョンが ACTIVE 状態で新しいサービスのデプロイを開始した場合にのみ発生します。	はい
PRE_SCALE_UP	グリーンサービスリビジョンは開始されていません。ブルーサービスリビジョンにより、本番トラフィックの 100% が処理されています。テストトラフィックがありません。	はい
SCALE_UP	グリーンサービスリビジョンが 100% までにスケールアップし、新しいタスクを起動する時間。現時点では、グリーンサービスリビジョンはトラフィックを処理していません。	いいえ
POST_SCALE_UP	グリーンサービスリビジョンが開始されました。ブルーサービスリビジョンにより、本番トラフィックの 100% が処理されています。テストトラフィックがありません。	はい
TEST_TRAFFIC_SHIFT	ブルーおよびグリーンのサービスリビジョンが実行されています。ブルーサービスリビジョンにより、本番トラフィックの 100% が処理されます。グリーンサー	はい (Lambda のみ)

ライフサイクルのステージ	説明	ライフサイクルフックにこのステージを使用しますか?
	ビスリビジョンは、テストトラフィックの 0% から 100% に移行しています。	
POST_TEST_TRAFFIC_SHIFT	テストトラフィックの移行が完了しました。グリーンサービスリビジョンにより、テストトラフィックの 100% が処理されます。	はい
PRE_PRODUCTION_TRAFFIC_SHIFT	本番トラフィックがシフトする前に発生します。ブルー/グリーンデプロイの場合、このステージは 1 回呼び出されます。	はい
PRODUCTION_TRAFFIC_SHIFT	本番トラフィックがグリーンサービスリビジョンに移行しています。グリーンサービスリビジョンは、本番トラフィックの 0% から 100% に移行しています。	はい (Lambda のみ)
POST_PRODUCTION_TRAFFIC_SHIFT	本番トラフィックの移行が完了しました。	はい
BAKE_TIME	ブルーおよびグリーンのサービスリビジョンの両方が同時に実行される期間。	いいえ
CLEAN_UP	ブルーサービスリビジョンにより、実行中のタスク 0 へと完全にスケールダウンされました。グリーンサービスリビジョンは、このステージの後に本番サービスリビジョンになります。	いいえ

各ライフサイクルステージには、次のステージに進む前に合格する必要があるビルトインの検証チェックポイントが含まれています。検証が失敗した場合、サービスの可用性および信頼性を維持するため、デプロイを自動的にロールバックできます。

Lambda 関数を使用する場合、関数は作業を完了するか、15 分以内に IN_PROGRESS を返す必要があります。`callBackDelaySeconds` を使用して、Lambda への呼び出しを遅らせることができます。詳細については、GitHub の sample-amazon-ecs-blue-green-deployment-patterns の app.py 関数を参照してください。

Amazon ECS ブルー/グリーンデプロイに必要なリソース

マネージドトラフィックのシフトの有無にかかわらず、Amazon ECS ブルー/グリーンデプロイを使用できます。サービスが Elastic Load Balancing または Service Connect を使用している場合、Amazon ECS はブルーとグリーンのサービスリビジョン間のトラフィックシフトを管理します。サービスがロードバランサーまたは Service Connect (ヘッドレスサービス) を使用していない場合でも、制御されたロールアウトにブルー/グリーンデプロイを使用できますが、Amazon ECS はトラフィックシフトの自動管理は行いません。

マネージドトラフィックシフトについて、次のいずれかを設定します。

- Elastic Load Balancing
- Service Connect

次のリストは、Amazon ECS ブルー/グリーンデプロイに設定する必要があるものに対する大まかな概要を示しています。

- サービスで Application Load Balancer、Network Load Balancer、または Service Connect を使用している場合は、マネージドトラフィックのシフトに適したリソースを設定します。
  - Application Load Balancer – 詳細については、「ブルー/グリーンデプロイ、リニアデプロイおよびカナリアデプロイ用の Application Load Balancer リソース」を参照してください。
  - Network Load Balancer – 詳細については、「Amazon ECS ブルー/グリーン、リニアおよびカナリアデプロイの Network Load Balancer リソース」を参照してください。
  - Service Connect – 詳細については、「Amazon ECS ブルー/グリーンデプロイ、リニアデプロイおよびカナリアデプロイ用の Service Connect リソース」を参照してください。

  サービスがヘッドレス (ロードバランサーも Service Connect もなし) の場合は、追加のトラフィックシフトリソースを設定する必要はありません。
- サービスのデプロイコントローラーを `ECS` に設定します。
- サービス定義でデプロイ戦略を `blue/green` として設定します。
- 必要に応じて、次のように追加のパラメータを設定します。
  - 新しいデプロイのベイク時間

- 自動ロールバックの CloudWatch アラーム
- デプロイのライフサイクルフック (指定されたデプロイステージで実行される Lambda 関数または一時停止フック)

ベストプラクティス

Amazon ECS ブルー/グリーンデプロイを正常に処理するには、次のベストプラクティスに従ってください。

- アプリケーションの正常性を正確に反映する、適切なヘルスチェックを設定します。
- グリーンデプロイの十分なテストを可能にするベイク時間を設定します。
- CloudWatch アラームを実装し、問題を自動的に検出してロールバックをトリガーします。
- ライフサイクルフックを使用し、各デプロイステージで自動テストを実行します。
- アプリケーションが同時に実行されているブルーおよびグリーンのサービスリビジョンの両方を処理できることを確認します。
- デプロイ中に両方のサービスリビジョンを処理するため、十分なクラスターキャパシティの確保を計画します。
- ロールバック手順は、本番環境での実装前にテストします。

ブルー/グリーンデプロイ、リニアデプロイおよびカナリアデプロイ用の Application Load Balancer リソース

Amazon ECS ブルー/グリーンデプロイに Application Load Balancer を使用するには、ブルーおよびグリーンのサービスリビジョン間のトラフィックルーティングを可能にする特定のリソースを設定する必要があります。

ターゲットグループ

Elastic Load Balancing を使用したブルー/グリーンデプロイには、2 つのターゲットグループを作成する必要があります。

- ブルーサービスリビジョンのプライマリターゲットグループ (現在の本番トラフィック)
- グリーンサービスリビジョンの代替ターゲットグループ (新しいバージョン)

両方のターゲットグループは、次の設定で構成する必要があります。

- ターゲットタイプ: IP (awsvpcネットワークモードの Fargate または EC2 の場合)
- プロトコル: HTTP (またはアプリケーションが使用するプロトコル)
- ポート: アプリケーションがリッスンするポート (通常、HTTP は 80)
- VPC: Amazon ECS タスクと同じ VPC
- ヘルスチェック設定: アプリケーションの正常性を適切にチェックするように設定されています

ブルー/グリーンデプロイ中、デプロイステージに基づき、Amazon ECS によって適切なターゲットグループにタスクが自動的に登録されます。

Example Application Load Balancer 用ターゲットグループの作成

次の CLI コマンドでは、ブルー/グリーンデプロイの Application Load Balancer で使用する 2 つのターゲットグループが作成されます。

```
aws elbv2 create-target-group \
    --name blue-target-group \
    --protocol HTTP \
    --port 80 \
    --vpc-id vpc-abcd1234 \
    --target-type ip \
    --health-check-path / \
    --health-check-protocol HTTP \
    --health-check-interval-seconds 30 \
    --health-check-timeout-seconds 5 \
    --healthy-threshold-count 2 \
    --unhealthy-threshold-count 2

aws elbv2 create-target-group \
    --name green-target-group \
    --protocol HTTP \
    --port 80 \
    --vpc-id vpc-abcd1234 \
    --target-type ip \
    --health-check-path / \
    --health-check-protocol HTTP \
    --health-check-interval-seconds 30 \
    --health-check-timeout-seconds 5 \
    --healthy-threshold-count 2 \
    --unhealthy-threshold-count 2
```

Application Load Balancer

次の設定で Application Load Balancer を作成する必要があります。

- スキーム: 要件に応じて、インターネット向きまたは内部仕様
- IP アドレスタイプ: IPv4
- VPC: Amazon ECS タスクと同じ VPC
- サブネット: 異なるアベイラビリティーゾーンに少なくとも 2 つのサブネット
- セキュリティグループ: リスナーポートでトラフィックを許可するセキュリティグループ

Application Load Balancer にアタッチされたセキュリティグループには、Amazon ECS タスクにアタッチされたセキュリティグループへのトラフィックを可能にするアウトバウンドルールが必要です。

Example Application Load Balancer の作成

次の CLI コマンドでは、ブルー/グリーンデプロイに使用する Application Load Balancer が作成されます。

```
aws elbv2 create-load-balancer \
    --name my-application-load-balancer \
    --type application \
    --security-groups sg-abcd1234 \
    --subnets subnet-12345678 subnet-87654321
```

リスナーとルール

ブルー/グリーンデプロイの場合、Application Load Balancer にリスナーを設定する必要があります。

- 本番リスナー: 本番トラフィック (通常はポート 80 または 443) が処理されます
  - トラフィックは、最初にプライマリターゲットグループに転送されます (ブルーサービスリビジョン)
  - デプロイ後、トラフィックが代替ターゲットグループに転送されます (グリーンサービスリビジョン)
- テストリスナー (オプション): 本番トラフィックを移行する前にテストトラフィックが処理され、グリーンサービスリビジョンが検証されます
  - 別のポート (例えば、8080 や 8443) で設定できます。

- テスト中にトラフィックが代替ターゲットグループ (グリーンサービスリビジョン) に転送されます

ブルー/グリーンデプロイ中に Amazon ECS によってリスナールールが自動的に更新され、デプロイステージに基づいてトラフィックが適切なターゲットグループにルーティングされます。

Example本番リスナーの作成

次の CLI コマンドでは、プライマリ (ブルー) ターゲットグループにトラフィックを転送する本番リスナーをポート 80 で作成します。

```
aws elbv2 create-listener \
    --load-balancer-arn arn:aws:elasticloadbalancing:region:123456789012:loadbalancer/
app/my-application-load-balancer/abcdef123456 \
    --protocol HTTP \
    --port 80 \
    --default-actions
 Type=forward,TargetGroupArn=arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
blue-target-group/abcdef123456
```

Exampleテストリスナーの作成

次の CLI コマンドでは、代替 (グリーン) ターゲットグループにトラフィックを転送するテストリスナーをポート 8080 で作成します。

```
aws elbv2 create-listener \
    --load-balancer-arn arn:aws:elasticloadbalancing:region:123456789012:loadbalancer/
app/my-application-load-balancer/abcdef123456 \
    --protocol HTTP \
    --port 8080 \
    --default-actions
 Type=forward,TargetGroupArn=arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
green-target-group/ghijkl789012
```

Exampleパスベースのルーティング用にリスナールールを作成する

次の CLI コマンドでは、テスト用に特定のパスのトラフィックをグリーンターゲットグループに転送するルールを作成します。

```
aws elbv2 create-rule \
```

```
    --listener-arn arn:aws:elasticloadbalancing:region:123456789012:listener/app/my-
application-load-balancer/abcdef123456/ghijkl789012 \
    --priority 10 \
    --conditions Field=path-pattern,Values='/test/*' \
    --actions
 Type=forward,TargetGroupArn=arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
green-target-group/ghijkl789012
```

Exampleヘッダーベースのルーティング用にリスナールールを作成する

次の CLI コマンドでは、テスト用に特定のヘッダーを持つトラフィックをグリーンターゲットグループに転送するルールを作成します。

```
aws elbv2 create-rule \
    --listener-arn arn:aws:elasticloadbalancing:region:123456789012:listener/app/my-
application-load-balancer/abcdef123456/ghijkl789012 \
    --priority 20 \
    --conditions Field=http-header,HttpHeaderConfig='{Name=X-
Environment,Values=[test]}' \
    --actions
 Type=forward,TargetGroupArn=arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
green-target-group/ghijkl789012
```

サービス設定

Amazon ECS がユーザーに代わってクラスター内のロードバランサーリソースを管理するには、アクセス許可が必要です。詳細については、「ロードバランサー用の Amazon ECS インフラストラクチャの IAM ロール」を参照してください。

Elastic Load Balancing を使用したブルー/グリーンデプロイ用に Amazon ECS サービスを作成または更新する際には、次の設定を指定する必要があります。

user-input を独自の値に置き換えます。

この設定の主なコンポーネントは次のとおりです。

- `targetGroupArn`: プライマリターゲットグループの ARN (ブルーサービスリビジョン)。
- `alternateTargetGroupArn`: 代替ターゲットグループの ARN (グリーンサービスリビジョン)。
- `productionListenerRule`: 本番トラフィックのリスナールールの ARN。
- `roleArn`: Amazon ECS が Elastic Load Balancing リソースを管理できるようにするためのロールの ARN。

- strategy: BLUE_GREEN に設定してブルー/グリーンデプロイを有効にします。
- bakeTimeInMinutes: 本番トラフィックが移行した後に、ブルーおよびグリーンのサービスリビジョンの両方が同時に実行される期間。
- TestListenerRule: テストトラフィックのリスナールールの ARN。このパラメータはオプションです。

```
{
    "loadBalancers": [
        {
            "targetGroupArn":
 "arn:aws:elasticloadbalancing:region:123456789012:targetgroup/primary-target-group/
abcdef123456",
            "containerName": "container-name",
            "containerPort": 80,
            "advancedConfiguration": {
                "alternateTargetGroupArn":
 "arn:aws:elasticloadbalancing:region:account-id:targetgroup/alternate-target-group/
ghijkl789012",
                "productionListenerRule": "arn:aws:elasticloadbalancing:region:account-
id:listener-rule/app/load-balancer-name/abcdef123456/listener/ghijkl789012/rule/
mnopqr345678",
                "roleArn": "arn:aws:iam::123456789012:role/ecs-elb-role"
            }
        }
    ],
    "deploymentConfiguration": {
        "strategy": "BLUE_GREEN",
        "maximumPercent": 200,
        "minimumHealthyPercent": 100,
        "bakeTimeInMinutes": 5
    }
}
```

デプロイ中のトラフィックフロー

Elastic Load Balancing を使用したブルー/グリーンデプロイ中、トラフィックは次のようにシステムを通過します。

1. 初期状態: すべての本番トラフィックはプライマリターゲットグループ (ブルーサービスリビジョン) にルーティングされます。

2. グリーンサービスリビジョンのデプロイ: Amazon ECS は新しいタスクをデプロイしたら、代替ターゲットグループに登録します。
3. テストトラフィック: テストリスナーが設定されている場合、テストトラフィックは代替ターゲットグループにルーティングされ、グリーンサービスリビジョンが検証されます。
4. 本番トラフィック移行: Amazon ECS によって本番リスナールールが更新され、トラフィックが代替ターゲットグループ (グリーンサービスリビジョン) にルーティングされます。
5. ベイク時間: 本番トラフィックが移行された後、ブルーおよびグリーンのサービスリビジョンの両方が同時に実行される期間。
6. 完了: デプロイが正常に処理されると、ブルーサービスリビジョンは終了します。

デプロイ中に問題が検出された場合、Amazon ECS はトラフィックをプライマリターゲットグループ (ブルーサービスリビジョン) に戻すことで、自動的にロールバックできます。

Amazon ECS ブルー/グリーン、リニアおよびカナリアデプロイの Network Load Balancer リソース

Amazon ECS ブルー/グリーンデプロイに Network Load Balancer を使用するには、ブルーおよびグリーンのサービスリビジョン間のトラフィックルーティングを有効にする特定のリソースを設定する必要があります。このセクションでは、必要なコンポーネントとその設定について説明します。

設定に Network Load Balancer が含まれている場合、Amazon ECS によって次のライフサイクルステージに 10 分の遅延が追加されます。

- TEST_TRAFFIC_SHIFT
- PRODUCTION_TRAFFIC_SHIFT

この遅延は、設定されたトラフィックの重みおよびデータプレーンにある実際のトラフィックルーティングの間に不一致を引き起こす可能性がある Network Load Balancer のタイミング問題を考慮しています。

ターゲットグループ

Network Load Balancer を使用したブルー/グリーンデプロイの場合、2 つのターゲットグループを作成する必要があります。

- ブルーサービスリビジョンのプライマリターゲットグループ (現在の本番トラフィック)
- グリーンサービスリビジョン (新しいサービスリビジョン) の代替ターゲットグループ

両方のターゲットグループは、次の設定で構成する必要があります。

- ターゲットタイプ: `ip` (`awsvpc`ネットワークモードの Fargate または EC2 の場合)
- プロトコル: TCP (またはアプリケーションが使用するプロトコル)
- ポート: アプリケーションがリッスンするポート (通常、HTTP は 80)
- VPC: Amazon ECS タスクと同じ VPC
- ヘルスチェック設定: アプリケーションの正常性を適切にチェックするように設定されています

  TCP ヘルスチェックの場合、Network Load Balancer はターゲットと TCP 接続を確立します。接続が正常に処理された場合、ターゲットは正常と見なされます。

  HTTP/HTTPS ヘルスチェックの場合、Network Load Balancer によって HTTP/HTTPS リクエストがターゲットに送信され、レスポンスが検証されます。

ブルー/グリーンデプロイ中、デプロイステージに基づき、Amazon ECS によって適切なターゲットグループにタスクが自動的に登録されます。

Example Network Load Balancer 用にターゲットグループの作成

次の AWS CLI コマンドでは、ブルー/グリーンデプロイで Network Load Balancer に使用する 2 つのターゲットグループを作成します。

```
aws elbv2 create-target-group \
    --name blue-target-group \
    --protocol TCP \
    --port 80 \
    --vpc-id vpc-abcd1234 \
    --target-type ip \
    --health-check-protocol TCP

aws elbv2 create-target-group \
    --name green-target-group \
    --protocol TCP \
    --port 80 \
    --vpc-id vpc-abcd1234 \
    --target-type ip \
    --health-check-protocol TCP
```

Network Load Balancer

次の設定で Network Load Balancer を作成する必要があります。

- スキーム: 要件に応じて、インターネット向きまたは内部仕様
- IP アドレスタイプ: IPv4
- VPC: Amazon ECS タスクと同じ VPC
- サブネット: 異なるアベイラビリティーゾーンに少なくとも 2 つのサブネット

Application Load Balancer とは違って、Network Load Balancer はトランスポートレイヤー (レイヤー 4) で動作し、セキュリティグループは使用しません。代わりに、Amazon ECS タスクに関連付けられたセキュリティグループが、リスナーポートの Network Load Balancer からのトラフィックを許可することを確認する必要があります。

Example Network Load Balancer を作成する

次の AWS CLI コマンドでは、ブルー/グリーンデプロイで使用する Network Load Balancer を作成します。

```
aws elbv2 create-load-balancer \
    --name my-network-load-balancer \
    --type network \
    --subnets subnet-12345678 subnet-87654321
```

ブルー/グリーンデプロイで NLB を使用する際の考慮事項

ブルー/グリーンデプロイに Network Load Balancer を使用する場合、次の内容を考慮してください。

- レイヤー 4 オペレーション: Network Load Balancer はトランスポートレイヤー (レイヤー 4) で動作し、アプリケーションレイヤー (レイヤー 7) のコンテンツは検査しません。つまり、ルーティングの決定に HTTP ヘッダーまたはパスを使用することはできません。
- ヘルスチェック: Network Load Balancer のヘルスチェックは、TCP、HTTP、HTTPS プロトコルに制限されています。TCP ヘルスチェックの場合、Network Load Balancer は接続を確立できるかどうかのみが検証されます。
- 接続の保存: Network Load Balancer はクライアントのソース IP アドレスを保存します。これはセキュリティおよびログ記録目的で役立ちます。

- 静的 IP アドレス: Network Load Balancer は、各サブネットに静的 IP アドレスを提供します。ホワイトリストの登録や、クライアントが固定 IP アドレスに接続する必要がある場合に役立ちます。
- テストトラフィック: Network Load Balancer はコンテンツベースのルーティングをサポートしていないため、テストトラフィックは本番トラフィックとは異なるポートに送信する必要があります。

リスナーとルール

Network Load Balancer を使用したブルー/グリーンデプロイの場合、リスナーを設定する必要があります。

- 本番リスナー: 本番トラフィック (通常はポート 80 または 443) が処理されます
  - トラフィックは、最初にプライマリターゲットグループに転送されます (ブルーサービスリビジョン)
  - デプロイ後、トラフィックが代替ターゲットグループに転送されます (グリーンサービスリビジョン)
- テストリスナー (オプション): 本番トラフィックを移行する前にテストトラフィックが処理され、グリーンサービスリビジョンが検証されます
  - 別のポート (8080 や 8443 など) で設定できます
  - テスト中にトラフィックが代替ターゲットグループ (グリーンサービスリビジョン) に転送されます

Application Load Balancer とは違って、Network Load Balancer はコンテンツベースのルーティングルールをサポートしていません。代わりに、トラフィックはリスナーポートおよびプロトコルに基づいてルーティングされます。

次の AWS CLI コマンドでは、Network Load Balancer の本番リスナーおよびテストリスナーを作成します。

user-input を独自の値に置き換えます。

```
aws elbv2 create-listener \
    --load-balancer-arn arn:aws:elasticloadbalancing:region:123456789012:loadbalancer/
net/my-network-lb/1234567890123456 \
    --protocol TCP \
    --port 80 \
```

```
    --default-actions Type=forward,
 TargetGroupArn=arn:aws:elasticloadbalancing:region:123456789012:targetgroup/blue-
target-group/1234567890123456

aws elbv2 create-listener \
    --load-balancer-arn arn:aws:elasticloadbalancing:region:123456789012:loadbalancer/
net/my-network-lb/1234567890123456 \
    --protocol TCP \
    --port 8080 \
    --default-actions Type=forward,
 TargetGroupArn=arn:aws:elasticloadbalancing:region:123456789012:targetgroup/green-
target-group/1234567890123456
```

サービス設定

Amazon ECS がユーザーに代わってクラスター内のロードバランサーリソースを管理するには、アクセス許可が必要です。詳細については、「ロードバランサー用の Amazon ECS インフラストラクチャの IAM ロール」を参照してください。

Network Load Balancer を使用したブルー/グリーンデプロイ用の Amazon ECS サービスを作成または更新する場合、次の設定を指定する必要があります。

user-input を独自の値に置き換えます。

この設定の主なコンポーネントは次のとおりです。

- `targetGroupArn`: プライマリターゲットグループの ARN (ブルーサービスリビジョン)
- `alternateTargetGroupArn`: 代替ターゲットグループの ARN (グリーンサービスリビジョン)
- `productionListenerRule`: 本番トラフィック用リスナーの ARN
- `testListenerRule`: (オプション) テストトラフィック用リスナーの ARN
- `roleArn`: Amazon ECS に Network Load Balancer リソースの管理を可能にするロールの ARN
- `strategy`: `BLUE_GREEN` に設定してブルー/グリーンデプロイを有効にします
- `bakeTimeInMinutes`: 本番トラフィックが移行した後に、ブルーおよびグリーンのサービスリビジョンの両方が同時に実行される期間

```
{
    "loadBalancers": [
        {
```

```
            "targetGroupArn":
 "arn:aws:elasticloadbalancing:region:123456789012:targetgroup/blue-target-
group/1234567890123456",
            "containerName": "container-name",
            "containerPort": 80,
            "advancedConfiguration": {
                "alternateTargetGroupArn":
 "arn:aws:elasticloadbalancing:region:123456789012:targetgroup/green-target-
group/1234567890123456",
                "productionListenerRule":
 "arn:aws:elasticloadbalancing:region:123456789012:listener/net/my-network-
lb/1234567890123456/1234567890123456",
                "testListenerRule":
 "arn:aws:elasticloadbalancing:region:123456789012:listener/net/my-network-
lb/1234567890123456/2345678901234567",
                "roleArn": "arn:aws:iam::123456789012:role/ecs-nlb-role"
            }
        }
    ],
    "deploymentConfiguration": {
        "strategy": "BLUE_GREEN",
        "maximumPercent": 200,
        "minimumHealthyPercent": 100,
        "bakeTimeInMinutes": 5
    }
}
```

デプロイ中のトラフィックフロー

Network Load Balancer を使用したブルー/グリーンデプロイ中、トラフィックは次のようにシステムを通過します。

1. 初期状態: すべての本番トラフィックはプライマリターゲットグループ (ブルーサービスリビジョン) にルーティングされます。
2. グリーンサービスリビジョンのデプロイ: Amazon ECS は新しいタスクをデプロイしたら、代替ターゲットグループに登録します。
3. テストトラフィック: テストリスナーが設定されている場合、テストトラフィックは代替ターゲットグループにルーティングされ、グリーンサービスリビジョンが検証されます。
4. 本番トラフィック移行: Amazon ECS によって本番リスナーが更新され、トラフィックは代替ターゲットグループ (グリーンサービスリビジョン) にルーティングされます。

5. ベイク時間: 本番トラフィックが移行された後、ブルーおよびグリーンのサービスリビジョンの両方が同時に実行される期間。
6. 完了: デプロイが正常に処理されると、ブルーサービスリビジョンは終了します。

デプロイ中に問題が検出された場合、Amazon ECS はトラフィックをプライマリターゲットグループ (ブルーサービスリビジョン) に戻すことで、自動的にロールバックできます。

Amazon ECS ブルー/グリーンデプロイ、リニアデプロイおよびカナリアデプロイ用の Service Connect リソース

ブルー/グリーンデプロイで Service Connect を使用すると、ブルーおよびグリーンのサービスリビジョン間に適切なトラフィックルーティングを有効にするため、特定のコンポーネントを設定する必要があります。このセクションでは、必要なコンポーネントとその設定について説明します。

アーキテクチャの概要

Service Connect により、Amazon ECS タスクに自動的に挿入されたマネージドサイドカープロキシを介して、サービス検出およびサービスメッシュ機能の両方が構築されます。これらのプロキシはルーティングの決定、再試行、メトリクス収集を処理し、AWS Cloud Map はサービスレジストリのバックエンドを提供します。Service Connect を有効にしてサービスをデプロイするとサービス自体が AWS Cloud Map で登録され、クライアントサービスは名前空間を介してサービスを検出します。

標準の Service Connect 実装では、クライアントサービスは論理サービス名に接続し、サイドカープロキシは実際のサービスインスタンスへのルーティングを処理します。ブルー/グリーンデプロイを使用すると、このモデルが拡張されて `testTrafficRules` 設定を介したテストトラフィックのルーティングが含まれます。

ブルー/グリーンデプロイ中は、次の主要コンポーネントが連携します。

- Service Connect Proxy: サービス間のすべてのトラフィックは Service Connect プロキシを通過し、設定に基づいてルーティングを決定します。
- AWS Cloud Map 登録: ブルーデプロイおよびグリーンデプロイの両方が AWS Cloud Map に登録されますが、グリーンデプロイは最初は「テスト」エンドポイントとして登録されます。
- テストトラフィックのルーティング: Service Connect 設定の `testTrafficRules` は、テストトラフィックを識別してグリーンデプロイにルーティングする方法を決定します。リクエストの特定の HTTP ヘッダーがトラフィックをテストリビジョンに転送するヘッダーベースのルーティングによって実現されます。デフォルトでは、カスタムルールが指定されていない場合、Service Connect は HTTP ベースプロトコルの `x-amzn-ecs-blue-green-test` ヘッダーを認識します。

- クライアント設定: 名前空間内のすべてのクライアントは、本番ルートおよびテストルートの両方を自動的に受けますが、テストルールに一致するリクエストのみがグリーンデプロイに移行されます。

このアプローチが強力なのは、移行中のサービス検出の複雑さを処理するからです。トラフィックがブルーデプロイからグリーンデプロイに移行すると、すべての接続および検出メカニズムが自動的に更新されます。サービスメッシュがすべてを処理するため、DNS レコードを更新したり、ロードバランサーを再設定したり、個別にサービス検出の変更をデプロイしたりする必要はありません。

トラフィックのルーティングとテスト

Service Connect はブルー/グリーンデプロイ用の高度なトラフィックルーティング機能を提供します (テストシナリオ用のヘッダーベースのルーティングやクライアントエイリアス設定など)。

テストトラフィックヘッダーのルール

ブルー/グリーンデプロイ中に、テスト目的のために特定のリクエストをグリーン (新規) サービスリビジョンにルーティングするため、テストトラフィックヘッダーのルールを設定できます。デプロイを完了する前に、トラフィックが制御された新しいバージョンを検証できます。

Service Connect はヘッダーベースのルーティングを使用し、テストトラフィックを識別します。デフォルトでは、カスタムルールが指定されていない場合、Service Connect は HTTP ベースプロトコルの `x-amzn-ecs-blue-green-test` ヘッダーを認識します。このヘッダーがリクエストに存在すると、Service Connect プロキシによってテストのためにリクエストがグリーンデプロイへと自動的にルーティングされます。

テストトラフィックヘッダーのルールを使用すると、以下を実行できます。

- 特定のヘッダーを持つリクエストをグリーンサービスリビジョンにルーティングする
- トラフィックのサブセットで新機能をテストする
- フルトラフィックのカットオーバー前に、サービス動作を検証する
- Canary テスト戦略を実装する
- 本番のような環境で統合テストを実行する

ヘッダーベースのルーティングメカニズムは、既存のアプリケーションアーキテクチャとシームレスに連携します。クライアントサービスがブルー/グリーンデプロイプロセスを認識する必要はありません。テストリクエストを送信するときに適切なヘッダーを含めるだけで、Service Connect プロキシはルーティングロジックを自動的に処理します。

> **Important**
>
> テストトラフィックヘッダールールでは、Service Connect サービスポートマッピングの `appProtocol` が `http`、`http2`、または `grpc` に設定されている必要があります。サービスがアプリケーションレイヤープロトコルを使用せずにプレーン TCP を使用している場合 (つまり、`appProtocol` が設定されていない場合)、ブルー/グリーンデプロイにヘッダーベースの `testTrafficRules` を使用することはできません。この場合、代わりにポートベースのテストトラフィックルーティングを使用します。

テストトラフィックヘッダーのルール設定の詳細については、「Amazon Elastic Container Service API リファレンス」の「ServiceConnectTestTrafficHeaderRules」を参照してください。

ヘッダーマッチングルール

ヘッダーマッチングルールは、ブルー/グリーンデプロイ中にテストトラフィックをルーティングする際の基準を定義します。グリーンサービスリビジョンにルーティングされるリクエストを正確に制御するために、複数の一致条件を設定できます。

ヘッダーマッチングは以下の内容をサポートします。

- ヘッダー値の完全一致
- ヘッダープレゼンスチェック
- パターンベースのマッチング
- 複数ヘッダーの組み合わせ

ユースケース例では、特定のユーザーエージェント文字列、API バージョン、機能フラグのあるリクエストをテストするためにグリーンサービスにルーティングします。

ヘッダーマッチング設定の詳細については、「Amazon Elastic Container Service API リファレンス」の「ServiceConnectTestTrafficHeaderMatchRules」を参照してください。

ブルー/グリーンデプロイのクライアントエイリアス

クライアントエイリアスは、ブルー/グリーンデプロイ中にサービスに安定した DNS エンドポイントを提供します。クライアントアプリケーションが接続エンドポイントを変更する必要がなく、ブルーおよびグリーンのサービスリビジョン間にシームレスなトラフィックルーティングを実現します。

ブルー/グリーンデプロイ中、クライアントエイリアスは以下を実行します。

- クライアント接続に一貫した DNS 名を維持する
- サービスリビジョン間で自動トラフィック切り替えを有効にする
- 段階的なトラフィック移行戦略をサポートする
- トラフィックをブルーリビジョンにリダイレクトし、ロールバック機能を提供する

異なるポートまたはプロトコルに複数のクライアントエイリアスを設定でき、複雑なサービスアーキテクチャがデプロイ中に接続を維持できるようにします。

クライアントエイリアスグ設定の詳細については、「Amazon Elastic Container Service API リファレンス」の「ServiceConnectClientAlias」を参照してください。

トラフィックルーティングのベストプラクティス

Service Connect を使用したブルー/グリーンデプロイ用にトラフィックルーティングを実装する際は、次のベストプラクティスを考慮してください。

- ヘッダーベースのテストから開始: テストトラフィックヘッダーのルールを使用して、すべてのトラフィックを切り替える前に、制御されたトラフィックでグリーンサービスを検証します。
- ヘルスチェックの設定: 異常なインスタンスにトラフィックをルーティングしないように、ブルーサービスおよびグリーンサービスの両方に適切なヘルスチェックが設定されていることを確認します。
- サービスメトリクスのモニタリング: デプロイ中に両方のサービスリビジョンの主要なパフォーマンス指標を追跡し、問題を早期に特定します。
- ロールバック戦略の計画: クライアントエイリアスおよびルーティングルールを設定すると、問題が検出されたときにブルーサービスへと迅速にロールバックできます。
- ヘッダー一致ロジックのテスト: 本番以外の環境でヘッダー一致ルールを検証してから、本番デプロイに適用します。

Service Connect ブルー/グリーンデプロイのワークフロー

Service Connect がブルー/グリーンデプロイプロセスを管理するしくみを理解すると、デプロイを効果的に実装してトラブルシューティングできます。次のワークフローは、デプロイの各フェーズ中にでさまざまなコンポーネントが相互作用する様子を示しています。

デプロイフェーズ

Service Connect のブルー/グリーンデプロイはいくつかの異なるフェーズを通ります。

1. 初期状態: ブルーサービスが本番トラフィックの 100% を処理します。名前空間内のすべてのクライアントサービスは、Service Connect で設定された論理サービス名を介してブルーサービスに接続されます。
2. グリーンサービス登録: グリーンデプロイが開始されると、「テスト」エンドポイントとして AWS Cloud Map に登録されます。クライアントサービスの Service Connect プロキシは、本番およびテスト用のルート設定の両方を自動的に提供します。
3. テストトラフィックのルーティング: テストトラフィックヘッダー (`x-amzn-ecs-blue-green-test` など) を含むリクエストは、Service Connect プロキシによってグリーンサービスに自動的にルーティングされます。本番トラフィックは引き続きブルーサービスに流れます。
4. トラフィック移行の準備: テストが正常に処理されると、デプロイプロセスは本番トラフィック移行に備えます。ブルーサービスおよびグリーンサービスの両方ともで、登録状態および正常性が維持されます。
5. 本番トラフィック移行: Service Connect 設定は、本番トラフィックをグリーンサービスにルーティングするために更新されます。クライアントサービスの更新や DNS の変更を必要とせず、自動的に行われます。
6. ベイク期間: 本番トラフィックが移行した後、ブルーおよびグリーンのサービスリビジョンの両方が同時に実行される期間。
7. ブルーサービスの登録解除: トラフィック移行および検証が正常に処理されたら、ブルーサービスは AWS Cloud Map から登録解除されて終了し、デプロイを完了します。

Service Connect プロキシの動作

Service Connect プロキシは、ブルー/グリーンデプロイ中のトラフィックを管理するうえで重要な役割を果たします。動作を理解することで、効果的なテストおよびデプロイ戦略を設計できます。

ブルー/グリーンデプロイ中の主要なプロキシ動作

- 自動ルート検出: アプリケーションの再起動や設定の変更を必要とせず、プロキシは AWS Cloud Map から本番およびテストのルートの両方を自動的に検出します。
- ヘッダーベースのルーティング: プロキシは受信リクエストヘッダーを検査し、設定されたテストトラフィックのルールに基づいてトラフィックを適切なサービスリビジョンにルーティングします。
- ヘルスチェックの統合: プロキシはトラフィックを正常なサービスインスタンスにのみルーティングし、ルーティングプールから異常なタスクを自動的に除外します。
- 再試行とサーキットブレーク: プロキシはビルトインの再試行ロジックおよびサーキットブレーク機能を提供し、デプロイ中の耐障害性を向上させます。

- メトリクス収集: プロキシはブルーサービスおよびグリーンサービスの両方向けに詳細なメトリクスを収集し、デプロイ中の包括的なモニタリングを可能にします。

サービス検出の更新

ブルー/グリーンデプロイに Service Connect を使用する主な利点の 1 つは、サービス検出の更新を自動的に処理することです。従来のブルー/グリーンデプロイには、多くの場合は複雑な DNS 更新またはロードバランサーの再設定が必要ですが、Service Connect によってこれらの変更が明確に管理されます。

デプロイ中、Service Connect は以下のものを処理します。

- 名前空間の更新: Service Connect 名前空間には、適切なルーティングルールが適用されて、ブルーおよびグリーンのサービスエンドポイントの両方が自動的に含まれます。
- クライアント設定: 名前空間内のすべてのクライアントサービスは再起動や再デプロイを必要とせず、更新されたルーティング情報を自動的に受けとります。
- 段階的な移行: サービス検出の更新は段階的かつ安全に行われ、進行中のリクエストが中断されることはありません。
- ロールバックサポート: ロールバックが必要な場合、Service Connect はサービス検出の設定を迅速に戻して、トラフィックをブルーサービスに戻すことができます。

Amazon ECS ブルー/グリーンデプロイの作成

Amazon ECS ブルー/グリーンデプロイを使用すると、本番環境に実装する前にサービスを変更してテストできます。

前提条件

ブルー/グリーンデプロイを開始する前に、次の操作を実行します。

1. 適切なアクセス許可を設定します。
   - Elastic Load Balancing のアクセス許可の詳細については、「ロードバランサー用の Amazon ECS インフラストラクチャの IAM ロール」を参照してください。
   - Lambda のアクセス許可の詳細については、「Amazon ECS ブルー/グリーンデプロイの Lambda 関数に必要となるアクセス許可」を参照してください。
2. (オプション) マネージドトラフィックシフトについて、次のいずれかのリソースを設定します。サービスがヘッドレス (ロードバランサーも Service Connect もなし) の場合は、このステップを

省略できます。Amazon ECS はヘッドレスサービスのトラフィックシフトの自動管理を行いません。

- Application Load Balancer – 詳細については、「ブルー/グリーンデプロイ、リニアデプロイおよびカナリアデプロイ用の Application Load Balancer リソース」を参照してください。
- Network Load Balancer – 詳細については、「Amazon ECS ブルー/グリーン、リニアおよびカナリアデプロイの Network Load Balancer リソース」を参照してください。
- Service Connect – 詳細については、「Amazon ECS ブルー/グリーンデプロイ、リニアデプロイおよびカナリアデプロイ用の Service Connect リソース」を参照してください。

3. ライフサイクルステージに Lambda 関数を実行するかどうかを決定します。
   - PRE_SCALE_UP
   - POST_SCALE_UP
   - TEST_TRAFFIC_SHIFT
   - POST_TEST_TRAFFIC_SHIFT
   - PRODUCTION_TRAFFIC_SHIFT
   - POST_PRODUCTION_TRAFFIC_SHIFT

   詳細については、「AWS Lambda デベロッパーガイド」の「コンソールで Lambda 関数の作成」を参照してください。

手順

コンソールまたは AWS CLI を使用して、Amazon ECS ブルー/グリーンサービスを作成できます。

Console

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. サービスを起動するリソースを決定します。

サービスの起動元	Steps	
クラスター	a. [Clusters] ページで、サービスを作成するクラスターを選択します。 クラスター詳細のページが表示されます。	

サービスの起動元	Steps
	b. [Services (サービス)] タブで、[Create (作成)] を選択します。
タスク定義	a. [タスク定義] ページで、タスク定義を選択します。 b. [デプロイ] メニューで、[サービスの作成] を選択します。

[サービスの作成] ページが表示されます。

3. [サービスの詳細] で、次の操作を行います。

   a. [タスク定義ファミリー] では、使用するタスク定義を選択します。次に [タスク定義リビジョン] には、使用するリビジョンを入力します。

   b. [Service name] (サービス名) でサービスの名前を入力します。

4. 既存のクラスターでサービスを実行するには、[既存のクラスター] にクラスターを選択します。新しいクラスターでサービスを実行するには、[クラスターの作成] を選択します。

5. クラスターのインフラストラクチャ全体にタスクを分散する方法を選択します。[コンピュート設定] で、オプションを選択します。

コンピューティングオプション	Steps
キャパシティープロバイダー戦略	a. [コンピューティングオプション] で、[キャパシティプロバイダー戦略] を選択します。 b. 戦略を選択: • デフォルトのキャパシティープロバイダー戦略を使用する

コンピューティングオプション	Steps
	には、[Use cluster default] (デフォルトのクラスターを使用) を選択します。 • クラスターにデフォルトのキャパシティプロバイダー戦略がない場合、または カスタム戦略を使用する場合は、[カスタムを使用]、[キャパシティプロバイダー戦略を追加] を選択し、[ベース]、[キャパシティプロバイダー]、[重み] を指定して、カスタムキャパシティプロバイダー戦略を定義します。 **Note** 戦略でキャパシティープロバイダーを使用するには、キャパシティープロバイダーをクラスターに関連付ける必要があります。

コンピューティングオプション	Steps	
起動タイプ	a. [Compute options] (コンピューティングオプション) セクションで、[Launch type] (起動タイプ) を選択します。 b. [Launch type] (起動タイプ) で、起動タイプを選択します。 c. (オプション) Fargate が指定されている場合は、[プラットフォームバージョン] で、使用するプラットフォームのバージョンを指定します。プラットフォームバージョンが指定されない場合、LATEST プラットフォームバージョンが使用されます。	

6. [デプロイ設定] で、次の操作を行います。

   a. [サービスタイプ] では、[レプリカ] を選択します。

   b. [Desired tasks] (必要なタスク) で、サービス内で起動および維持するタスクの数を入力します。

   c. Amazon ECS がアベイラビリティーゾーン間のタスクの分散をモニタリングし、不均衡が発生したときにタスクを再分散するには、[アベイラビリティーゾーンのサービス再調整] で [アベイラビリティーゾーンのサービス再調整] を選択します。

   d. [ヘルスチェックの猶予期間] には、タスクの初回起動後にサービススケジューラが異常な Elastic Load Balancing、VPC Lattice、コンテナヘルスチェックを無視する時間 (秒単位) を入力します。ヘルスチェックの猶予期間値を指定しない場合、デフォルト値 0 が使用されます。

7. a. ベイク時間には、ブルーリビジョンが終了する前に、ブルーおよびグリーンのサービスリビジョンの両方が同時に実行される分数を入力します。これにより、検証およびテストの時間を確保できます。

   b. (オプション) デプロイの特定段階で実行されるように、ライフサイクルフックを設定します。[デプロイライフサイクルフック] で、[追加] を選択します。

   Lambda フックの場合:

   - [ターゲットタイプ] で [Lambda] を選択します。
   - [Lambda 関数] には、ARN の関数名を入力します。
   - [ロール] では、Lambda 関数を呼び出すアクセス許可を持つ IAM ロールを選択します。
   - [ライフサイクルステージ] では、Lambda 関数を実行するステージを選択します。

   一時停止フックの場合:

   - [ターゲットタイプ] で、[一時停止] を選択します。
   - [ライフサイクルステージ] で、デプロイを一時停止するステージを選択します。
   - (オプション) [タイムアウト] の場合は、タイムアウト期間とアクションを設定します。

8. Amazon ECS がデプロイの障害を検出して処理する方法を設定するには、[Deployment failure detection] (デプロイ障害検出) を展開し、オプションを選択します。

   a. タスクを開始できない場合にデプロイを停止するには、[Use the Amazon ECS deployment circuit breaker] (Amazon ECS デプロイサーキットブレーカーを使用する) を選択します。

   デプロイサーキットブレーカーによってデプロイが失敗状態に設定されたときに、ソフトウェアがデプロイを最後に完了したデプロイ状態に自動的にロールバックするようにするには、[失敗時のロールバック] を選択します。

   b. アプリケーションメトリクスに基づいてデプロイを停止するには、[CloudWatch アラームを使用する] を選択します。次に、[CloudWatch アラーム名] からアラームを選択します。新しいアラームを作成するには、CloudWatch コンソールに移動します。

   CloudWatch アラームによってデプロイが失敗状態に設定されたときに、ソフトウェアがデプロイを最後に完了したデプロイ状態に自動的にロールバックするようにするには、[失敗時のロールバック] を選択します。

9. (オプション) Service Connect を使用してサービスを相互接続するには、[Service Connect] を展開して以下を指定します。

   a. [Service Connect をオンにする] を選択します。

   b. [Service Connect configuration] (Service Connect 設定) で、クライアントモードを指定します。

      - サービスが名前空間内の他のサービスへの接続のみを必要とするネットワーククライアントアプリケーションを実行している場合は、[クライアント側のみ] を選択します。
      - サービスがネットワークまたは Web サービスアプリケーションを実行していて、このサービスにエンドポイントを提供し、名前空間内の他のサービスに接続する必要がある場合は、[Client and server] (クライアントとサーバー) を選択します。

   c. デフォルトのクラスター名前空間ではない名前空間を使用するには、[Namespace] (名前空間) でサービス名前空間を選択します。この場合、AWS アカウントの同じ AWS リージョンで個別に作成された名前空間、またはAWS Resource Access Manager (AWS RAM) を使用してアカウントと共有されている同じリージョンの名前空間のいずれかを選択できます。共有 AWS Cloud Map 名前空間の詳細については、「AWS Cloud Map デベロッパーガイド」の「クロスアカウント AWS Cloud Map 名前空間の共有」を参照してください。

   d. (オプション) ブルー/グリーンデプロイのテストトラフィックヘッダーのルールを設定します。[テストトラフィックのルーティング] では、以下を指定します。

      i. [テストトラフィックヘッダーのルールの有効化] を選択し、テスト中に特定のリクエストをグリーンサービスリビジョンにルーティングします。

      ii. [ヘッダー一致ルール] では、テストトラフィックをルーティングする条件を設定します。

         - [ヘッダー名]: 一致する HTTP ヘッダーの名前を入力します (例えば、`X-Test-Version` または `User-Agent`)。
         - [一致タイプ]: 一致する条件を選択します。
            - [完全一致]: ヘッダー値が指定された値と完全に一致するリクエストをルーティングする
            - [ヘッダーが存在する]: 値に関係なく、指定されたヘッダーを含むリクエストをルーティングする

- [パターン一致]: ヘッダー値が指定されたパターンと一致するリクエストをルーティングする
- [ヘッダー値] (完全一致またはパターン一致を使用する場合): 照合する値またはパターンを入力します。

複数のヘッダー一致ルールを追加し、複雑なルーティングロジックを作成できます。設定されたルールに一致するリクエストは、テストのためにグリーンサービスリビジョンにルーティングされます。

iii. [ヘッダールールの追加] を選択し、追加のヘッダー一致条件を設定します。

> **Note**
>
> テストトラフィックヘッダーのルールにより、完全なデプロイを完了する前に、制御されたトラフィックで新機能を検証できます。これにより、ブルーサービスリビジョンへの通常のトラフィックフローを維持しながら、特定のリクエスト (内部テストツールやベータユーザーなどによるもの) でグリーンサービスリビジョンをテストできます。

e. (オプション) ログ設定を指定します。[ログコレクションを使用] を選択します。デフォルトのオプションでは、コンテナログを CloudWatch Logs に送信します。その他のログドライバオプションは、AWS FireLens を使用して構成されます。詳細については、「Amazon ECS ログを AWS サービスまたは AWS Partner に送信する」を参照してください。

以下では、各コンテナログの送信先について詳しく説明します。

- Amazon CloudWatch — コンテナログを CloudWatch Logs に送信するようにタスクを設定します。デフォルトのログドライバーオプションが提供され、ユーザーに代わり CloudWatch ロググループを作成します。別のロググループ名を指定するには、ドライバーオプションの値を変更します。
- [Amazon Data Firehose] — Firehose にコンテナログを送信するようタスクを設定します。Firehose 配信ストリームにログを送信するデフォルトのログドライバーオプションが提供されています。別の配信ストリーム名を指定するには、ドライバーオプションの値を変更します。
- Amazon Kinesis Data Streams — Kinesis Data Streams にコンテナログを送信するようタスクを設定します。Kinesis Data Streams のストリームにログを送信するデフォ

ルトのログドライバーオプションが提供されています。別のストリーム名を指定するには、ドライバーオプションの値を変更します。

- Amazon OpenSearch Service — コンテナログを OpenSearch Service ドメインに送信するようタスクを設定します。ログドライバーオプションを提供する必要があります。
- Amazon S3 — Amazon S3 バケットにコンテナログを送信するようタスクを設定します。デフォルトのログドライバーオプションはデフォルトで設定されていますが、有効な Amazon S3 バケット名を指定する必要があります。

10. (オプション) ブルー/グリーンデプロイ用に [ロードバランシング] を設定します。

Elastic Load Balancing のタイプ	Steps	
Application Load Balancer	a. [ロードバランサーのタイプ] では [Application Load Balancer] を選択します。 b. 選択新しいロードバランサーを作成するをクリックして新しいApplication Load Balancer を作成するか、既存のロードバランサーを使用する既存のApplication Load Balancer を選択します。 c. [コンテナ] では、サービスをホストするコンテナを選択します。 d. [Load balancer name] (ロードバランサー名) では、一意な名前を入力します。 e. [Listener] (リスナー) で、Application Load Balancer が接続リクエ	

Elastic Load Balancing のタイプ	Steps	
	ストをリッスンするポートとプロトコルを入力します。デフォルトでは、ロードバランサーはポート 80 と HTTP を使用するように設定されます。 • [本番ルール] には、ルールの [評価順序] および [パスパターン] を入力します。 このルールは、本番サービスリビジョンのトラフィック用です。 • [テストルール] には、ルールの [評価順序] および [パスパターン] を入力します。 このルールは、テストサービスリビジョンのトラフィック用です。 f. [ターゲットグループ] では、以下を設定します。 • [Target group name] (ターゲットグループ名) では、Application Load Balancer がリクエストをルーティングするターゲットグループの名前とプロトコルを入力します。	

Elastic Load Balancing のタイプ	Steps	
	• [プロトコル] では、Application Load Balancer がリクエストをルーティングするターゲットグループのプロトコルを選択します。デフォルトでは、ターゲットグループは、タスク定義で定義されている最初のコンテナにリクエストをルーティングします。 • [登録解除の遅延] には、ロードバランサーがターゲットの状態を UNUSED に変更するまでの秒数を入力します。デフォルトは 300 秒です。 • [Health check path] (ヘルスチェックパス) では、Application Load Balancer とコンテナ間の接続の正常性を検証するために、Application Load Balancer が定期的にリクエストを送信するコンテナ内に存在するパスを入力します。デフォルトはルートディレクトリ (/)。	

Elastic Load Balancing のタイプ	Steps	
	• [代替グループ名] には、テスト (緑) サービスリビジョンのターゲットグループのグループ名を入力します。	

Elastic Load Balancing のタイプ	Steps	
Network Load Balancer	a. [Load balancer type] (ロードバランサータイプ) では、[Network Load Balancer] を選択します。 b. [Load Balancer] (ロードバランサー) では、既存の Network Load Balancer を選択します。 c. [Choose container to load balance] (負荷分散するコンテナを選択) で、サービスをホストするコンテナを選択します。 d. [本番リスナー] では、[本番リスナーポート] および [本番リスナープロトコル] を選択します。 これは、本番サービスリビジョントのラフィックのリスナーです。 e. [テストリスナー] では、[テストリスナーポート] および [テストリスナープロトコル] を選択します。 これは、テストサービスリビジョンのトラフィックのリスナーです。	

Elastic Load Balancing のタイプ	Steps	
	f. [ターゲットグループ] では、以下を設定します。 • [Target group name] (ターゲットグループ名) では、Network Load Balancer がリクエストをルーティングするターゲットグループの名前とプロトコルを入力します。 • [プロトコル] では、Network Load Balancer がリクエストをルーティングするターゲットグループのプロトコルを選択します。デフォルトでは、ターゲットグループは、タスク定義で定義されている最初のコンテナにリクエストをルーティングします。 • [登録解除の遅延] には、ロードバランサーがターゲットの状態を UNUSED に変更するまでの秒数を入力します。デフォルトは 300 秒です。 • [Health check path] (ヘルスチェックパス) では、Application Load Balancer とコンテナ	

Elastic Load Balancing のタイプ	Steps	
	間の接続の正常性を検証するために、Application Load Balancer が定期的にリクエストを送信するコンテナ内に存在するパスを入力します。デフォルトはルートディレクトリ (/)。 • [代替グループ名] には、テスト (緑) サービスリビジョンのターゲットグループのグループ名を入力します。	

11. （オプション）サービスとタスクを識別しやすくするには、[Tags] (タグ) セクションを展開し、タグを設定します。

    新しく起動したすべてのタスクに対して、Amazon ECS がクラスター名とタスク定義タグで自動的にタグ付けするようにするには、[Amazon ECS マネージドタグを有効にする] を選択し、[タグの伝播元] で [タスク定義] を選択します。

    新しく起動したすべてのタスクに対して、Amazon ECS がクラスター名とサービスタグで自動的にタグ付けするようにするには、[Amazon ECS マネージドタグを有効にする] を選択し、[タグの伝播元] で [サービス] を選択します。

    タグを追加または削除します。

    - [タグを追加] [Add tag] (タグを追加) を選択し、以下を実行します。
      - [キー] にはキー名を入力します。
      - [値] にキー値を入力します。
    - [タグの削除] タグの横にある [タグの削除] を選択します。
12. [作成] を選択します。

AWS CLI

1. service-definition.json という名前のファイルを作成し、次の内容を記述します。

   user-input を独自の値に置き換えます。

```
{
  "serviceName": "myBlueGreenService",
  "cluster": "arn:aws:ecs:us-west-2:123456789012:cluster/sample-fargate-
cluster",
  "taskDefinition": "sample-fargate:1",
  "desiredCount": 5,
  "launchType": "FARGATE",
  "networkConfiguration": {
    "awsvpcConfiguration": {
      "subnets": [
        "subnet-09ce6e74c116a2299",
        "subnet-00bb3bd7a73526788",
        "subnet-0048a611aaec65477"
      ],
      "securityGroups": [
        "sg-09d45005497daa123"
      ],
      "assignPublicIp": "ENABLED"
    }
  },
  "deploymentController": {
    "type": "ECS"
  },
  "deploymentConfiguration": {
    "strategy": "BLUE_GREEN",
    "maximumPercent": 200,
    "minimumHealthyPercent": 100,
    "bakeTimeInMinutes": 2,
    "alarms": {
      "alarmNames": [
        "myAlarm"
      ],
      "rollback": true,
      "enable": true
    },
    "lifecycleHooks": [
      {
```

```
        "hookTargetArn": "arn:aws:lambda:us-
west-2:7123456789012:function:checkExample",
        "roleArn": "arn:aws:iam::123456789012:role/ECSLifecycleHookInvoke",
        "lifecycleStages": [
          "PRE_SCALE_UP"
        ],
        "hookDetails": {
          "MY_CONFIG_KEY": "my-config-value"
        }
      }
    ]
  },
  "loadBalancers": [
    {
      "targetGroupArn": "arn:aws:elasticloadbalancing:us-
west-2:123456789012:targetgroup/blue-target-group/54402ff563af1197",
      "containerName": "fargate-app",
      "containerPort": 80,
      "advancedConfiguration": {
        "alternateTargetGroupArn": "arn:aws:elasticloadbalancing:us-
west-2:123456789012:targetgroup/green-target-group/cad10a56f5843199",
        "productionListenerRule": "arn:aws:elasticloadbalancing:us-
west-2:123456789012:listener-rule/app/my-blue-green-
demo/32e0e4f946c3c05b/9cfa8c482e204f7d/831dbaf72edb911",
        "roleArn": "arn:aws:iam::123456789012:role/LoadBalancerManagementforECS"
      }
    }
  ]
}
```

2. create-service を実行します。

   user-input を独自の値に置き換えます。

```
aws ecs create-service --cli-input-json file://service-definition.json
```

   または、次の例を使用して、ロードバランサー設定を持つ ブルー/グリーンデプロイサービスを作成することもできます。

```
aws ecs create-service \
   --cluster "arn:aws:ecs:us-west-2:123456789012:cluster/MyCluster" \
   --service-name "blue-green-example-service" \
   --task-definition "nginxServer:1" \
```

```
   --launch-type "FARGATE" \
   --network-configuration
 "awsvpcConfiguration={subnets=[subnet-12345,subnet-67890,subnet-abcdef,subnet-
fedcba],securityGroups=[sg-12345],assignPublicIp=ENABLED}" \
   --desired-count 3 \
   --deployment-controller "type=ECS" \
   --deployment-configuration
 "strategy=BLUE_GREEN,maximumPercent=200,minimumHealthyPercent=100,bakeTimeInMinutes=0"
 \
   --load-balancers "targetGroupArn=arn:aws:elasticloadbalancing:us-
west-2:123456789012:targetgroup/MyBGtg1/
abcdef1234567890,containerName=nginx,containerPort=80,advancedConfiguration={alternateTa
west-2:123456789012:targetgroup/
MyBGtg2/0987654321fedcba,productionListenerRule=arn:aws:elasticloadbalancing:us-
west-2:123456789012:listener-rule/app/
MyLB/1234567890abcdef/1234567890abcdef,roleArn=arn:aws:iam::123456789012:role/
ELBManagementRole}"
```

次のステップ

- サービスを更新してデプロイを開始します。詳細については、「Amazon ECS サービスを更新する」を参照してください。
- デプロイプロセスをモニタリングし、ブルー/グリーンのパターンに従っていることを確認します。
  - グリーンサービスリビジョンが作成され、スケールアップされる
  - テストトラフィックがグリーンリビジョンにルーティングされている (設定されている場合)
  - 本番トラフィックがグリーンリビジョンに移行されている
  - ベイク時間が過ぎたら、ブルーリビジョンが終了する

Amazon ECS ブルー/グリーンデプロイのトラブルシューティング

以下は、Amazon ECS でブルー/グリーンデプロイを使用する際に発生する可能性がある一般的な問題の解決策です。ブルー/グリーンデプロイエラーは、次のフェーズで発生する可能性があります。

- 同期パス: `CreateService` または `UpdateService` API コールに応答してすぐに表示されるエラー。
- 非同期パス: `DescribeServiceDeployments` の `statusReason` フィールドに表示され、デプロイのロールバックを引き起こすエラー

> **Tip**
>
> Amazon ECS MCP サーバー と AI アシスタントを使用して、デプロイのモニタリングとデプロイ問題のトラブルシューティングを自然言語で実行できます。

ロードバランサーの設定問題

ロードバランサーの設定は、Amazon ECS でのブルー/グリーンデプロイの重要なコンポーネントです。リスナールール、ターゲットグループ、ロードバランサータイプの適切な設定は、デプロイを成功させるために不可欠です。このセクションでは、ブルー/グリーンデプロイが失敗する原因となる一般的なロードバランサー設定の問題について説明します。

ロードバランサーの問題をトラブルシューティングするときは、リスナールールとターゲットグループの関係を理解することが重要です。ブルー/グリーンデプロイでは:

- 本番リスナールールは、現在アクティブな (ブルー) サービスリビジョンにトラフィックをルーティングします
- テストリスナールールを使用して、本番トラフィックに移行する前に新しい (グリーン) サービスリビジョンを検証できます
- ターゲットグループを使用して、各サービスリビジョンからのコンテナインスタンスが登録されます
- デプロイ中、リスナールールでターゲットグループの重みを調整することで、トラフィックはブルーサービスリビジョンからグリーンサービスリビジョンに徐々に移行されます

リスナールール設定エラー

以下の問題は、ブルー/グリーンデプロイのリスナールールの誤った設定に関連しています。

リスナールール ARN の代わりに Application Load Balancer リスナー ARN を使用する

エラーメッセージ: `productionListenerRule has an invalid ARN format. Must be RuleArn for ALB or ListenerArn for NLB. Got: arn:aws:elasticloadbalancing:us-west-2:123456789012:listener/app/my-alb/abc123/def456`

解決策: Application Load Balancer を使用する場合は、リスナー ARN ではなく、`productionListenerRule` と `testListenerRule` のリスナールール ARN を指定する必要があります。Network Load Balancer の場合は、リスナー ARN を使用する必要があります。

リスナー ARN を確認する方法については、「Application Load Balancer ユーザーガイド」の「Application Load Balancer のリスナー」を参照してください。ルールの ARN の形式は *arn:aws:elasticloadbalancing:region:account-id:listener-rule/app/...* です。

本番リスナーとテストリスナーの両方に同じルールを使用する

エラーメッセージ: `The following rules cannot be used as both production and test listener rules: arn:aws:elasticloadbalancing:us-west-2:123456789012:listener-rule/app/my-alb/abc123/def456/ghi789`

解決策: 本番トラフィックとテストトラフィックには異なるリスナールールを使用する必要があります。テストターゲットグループにルーティングするテストトラフィック用に別のリスナールールを作成します。

ターゲットグループがリスナールールに関連付けられていない

エラーメッセージ: `Service deployment rolled back because of invalid networking configuration: Target group arn:aws:elasticloadbalancing:us-west-2:123456789012:targetgroup/myAlternateTG/abc123 is not associated with either productionListenerRule or testListenerRule.`

解決策: プライマリターゲットグループと代替ターゲットグループの両方を、本番リスナールールまたはテストリスナールールに関連付ける必要があります。ロードバランサー設定を更新して、両方のターゲットグループがリスナールールに適切に関連付けられるようにします。

Application Load Balancer のテストリスナールールがない

エラーメッセージ: `For Application LoadBalancer, testListenerRule is required when productionListenerRule is not associated with both targetGroup and alternateTargetGroup`

解決策: Application Load Balancer を使用する場合、両方のターゲットグループが本番リスナールールに関連付けられているのでなければ、テストリスナールールを指定する必要があります。設定に `testListenerRule` を追加し、両方のターゲットグループが本番リスナールールまたはテストリスナールールに関連付けられるようにします。詳細については、「Application Load Balancer ユーザーガイド」の「Application Load Balancer のリスナー」を参照してください。

ターゲットグループ設定のエラー

以下の問題は、ブルー/グリーンデプロイのターゲットグループ設定が正しくないことに関連しています。

リスナールールに、トラフィックがある複数のターゲットグループがある

エラーメッセージ: `Service deployment rolled back because of invalid networking configuration. productionListenerRule arn:aws:elasticloadbalancing:us-west-2:123456789012:listener-rule/app/my-alb/abc123/def456/ghi789 should have exactly one target group serving traffic but found 2 target groups which are serving traffic`

解決策: ブルー/グリーンデプロイを開始する前に、リスナールールでトラフィックを受信している (重みがゼロ以外の) ターゲットグループが 1 つだけであることを確認します。リスナールール設定を更新して、トラフィックを受信すべきでないターゲットグループの重みをゼロに設定します。

ロードバランサーエントリ間でターゲットグループが重複している

エラーメッセージ: `Duplicate targetGroupArn found: arn:aws:elasticloadbalancing:us-west-2:123456789012:targetgroup/myecs-targetgroup/abc123`

解決策: 各ターゲットグループ ARN は、サービス定義のすべてのロードバランサーエントリで一意である必要があります。設定を確認し、ロードバランサーエントリごとに異なるターゲットグループを使用するようにします。

本番リスナールールに想定されないターゲットグループがある

エラーメッセージ: `Service deployment rolled back because of invalid networking configuration. Production listener rule is forwarding traffic to unexpected target group arn:aws:elasticloadbalancing:us-west-2:123456789012:targetgroup/random-nlb-tg/abc123. Expected traffic to be forwarded to either targetGroupArn: arn:aws:elasticloadbalancing:us-west-2:123456789012:targetgroup/nlb-targetgroup/def456 or alternateTargetGroupArn: arn:aws:elasticloadbalancing:us-west-2:123456789012:targetgroup/nlb-tg-alternate/ghi789`

解決策: 本番リスナールールは、サービス定義で指定されていないターゲットグループにトラフィックを転送しています。サービス定義で指定されたターゲットグループにのみトラフィックを転送するようにリスナールールを設定するようにします。

詳細については、「Application Load Balancer のユーザーガイド」の「転送アクション」を参照してください。

ロードバランサータイプの設定エラー

以下の問題は、ブルー/グリーンデプロイのロードバランサータイプの設定が正しくないことに関連しています。

Classic Load Balancer 設定と、Application Load Balancer または Network Load Balancer 設定が混在している

エラーメッセージ: `All loadBalancers must be strictly either ELBv1 (defining loadBalancerName) or ELBv2 (defining targetGroupArn)`

> **Note**
>
> Classic Load Balancer は、Elastic Load Balancing の前世代のロードバランサーです。現行世代のロードバランサーに移行することをお勧めします。詳細については、「Classic Load Balancer の移行」を参照してください。

解決策: すべて Classic Load Balancer を使用するか、すべて Application Load Balancer または Network Load Balancer を使用します。

Application Load Balancer と Network Load Balancer では、`targetGroupArn` フィールドのみを指定します。

Classic Load Balancer で高度な設定が使用されている

エラーメッセージ: `advancedConfiguration field is not allowed with ELBv1 loadBalancers`

解決策: ブルー/グリーンデプロイの高度な設定は、Application Load Balancer と Network Load Balancer でのみサポートされます。Classic Load Balancer を使用 (`loadBalancerName` で指定) する場合は、`advancedConfiguration` フィールドを使用できません。Application Load Balancer に切り替えるか、`advancedConfiguration` フィールドを削除します。

ロードバランサー間で高度な設定に一貫性がない

エラーメッセージ: `Either all or none of the provided loadBalancers must have advancedConfiguration defined`

解決策: 複数のロードバランサーを使用している場合は、`advancedConfiguration` をすべてのロードバランサーに定義するか、いずれのロードバランサーにも定義しないようにする必要があります。設定を更新して、すべてのロードバランサーエントリ間で整合性を確保します。

ブルー/グリーンデプロイの高度な設定がない

エラーメッセージ: `advancedConfiguration field is required for all loadBalancers when using a non-ROLLING deployment strategy`

解決策: Application Load Balancer でブルー/グリーンデプロイ戦略を使用する場合は、すべてのロードバランサーエントリの `advancedConfiguration` フィールドを指定する必要があります。必要な `advancedConfiguration` をロードバランサー設定に追加します。

アクセス許可の問題

以下の問題は、ブルー/グリーンデプロイのアクセス許可が不十分であることに関連しています。

インフラストラクチャロールに信頼ポリシーがない

エラーメッセージ: `Service deployment rolled back because of invalid networking configuration. ECS was unable to manage the ELB resources due to missing permissions on ECS Infrastructure Role 'arn:aws:iam::123456789012:role/Admin'.`

解決策: ロードバランサーリソースの管理に指定された IAM ロールに正しい信頼ポリシーがありません。ロールの信頼ポリシーを更新して、サービスにロールの引き受けを許可します。信頼ポリシーには以下を含める必要があります。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

ロードバランサーロールに読み取りアクセス許可がない

エラーメッセージ: `service myService failed to describe target health on target-group myTargetGroup with (error User: arn:aws:sts::123456789012:assumed-role/myELBRole/ecs-service-scheduler is not authorized to perform: elasticloadbalancing:DescribeTargetHealth because no identity-based policy allows the elasticloadbalancing:DescribeTargetHealth action)`

解決策: ロードバランサーリソースの管理に使用される IAM ロールに、ターゲットのヘルス情報を読み取るアクセス許可がありません。`elasticloadbalancing:DescribeTargetHealth` アクセス許可をロールのポリシーに追加します。Elastic Load Balancing のアクセス許可の詳細については、「ロードバランサー用の Amazon ECS インフラストラクチャの IAM ロール」を参照してください。

ロードバランサーロールに書き込みアクセス許可がない

エラーメッセージ: `service myService failed to register targets in target-group myTargetGroup with (error User: arn:aws:sts::123456789012:assumed-role/myELBRole/ecs-service-scheduler is not authorized to perform: elasticloadbalancing:RegisterTargets on resource: arn:aws:elasticloadbalancing:us-west-2:123456789012:targetgroup/myTargetGroup/abc123 because no identity-based policy allows the elasticloadbalancing:RegisterTargets action)`

解決策: ロードバランサーリソースの管理に使用される IAM ロールに、ターゲットを登録するアクセス許可がありません。`elasticloadbalancing:RegisterTargets` アクセス許可をロールのポリシーに追加します。Elastic Load Balancing のアクセス許可の詳細については、「ロードバランサー用の Amazon ECS インフラストラクチャの IAM ロール」を参照してください。

リスナールールを変更するアクセス許可がありません

エラーメッセージ: `Service deployment rolled back because TEST_TRAFFIC_SHIFT lifecycle hook(s) failed. User: arn:aws:sts::123456789012:assumed-role/myELBRole/ECSNetworkingWithELB is not authorized to perform: elasticloadbalancing:ModifyListener on resource: arn:aws:elasticloadbalancing:us-west-2:123456789012:listener/app/my-alb/abc123/def456 because no identity-based policy allows the elasticloadbalancing:ModifyListener action`

解決策: ロードバランサーリソースの管理に使用される IAM ロールに、リスナーを変更するアクセス許可がありません。elasticloadbalancing:ModifyListener アクセス許可をロールのポリシーに追加します。Elastic Load Balancing のアクセス許可の詳細については、「ロードバランサー用の Amazon ECS インフラストラクチャの IAM ロール」を参照してください。

ブルー/グリーンデプロイでは、AmazonECS-ServiceLinkedRolePolicy マネージドポリシーをインフラストラクチャロールにアタッチすることをお勧めします。これには、ロードバランサーリソースを管理するために必要なすべてのアクセス許可が含まれます。

ライフサイクルフックの問題

以下の問題は、ブルー/グリーンデプロイのライフサイクルフックに関連しています。

Lambda フックロールの信頼ポリシーが正しくない

エラーメッセージ: Service deployment rolled back because TEST_TRAFFIC_SHIFT lifecycle hook(s) failed. ECS was unable to assume role arn:aws:iam::123456789012:role/Admin

解決策: Lambda ライフサイクルフックに指定された IAM ロールに正しい信頼ポリシーがありません。ロールの信頼ポリシーを更新して、サービスにロールの引き受けを許可します。信頼ポリシーには以下を含める必要があります。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Lambda フックが FAILED ステータスを返す

エラーメッセージ: `Service deployment rolled back because TEST_TRAFFIC_SHIFT lifecycle hook(s) failed. Lifecycle hook target arn:aws:lambda:us-west-2:123456789012:function:myHook returned FAILED status.`

解決策: ライフサイクルフックとして指定された Lambda 関数が FAILED ステータスを返しました。Amazon CloudWatch Logs の Lambda 関数ログをチェックして失敗の理由を確認し、デプロイイベントを正しく処理するように関数を更新します。

Lambda 関数を呼び出すアクセス許可がない

エラーメッセージ: `Service deployment rolled back because TEST_TRAFFIC_SHIFT lifecycle hook(s) failed. ECS was unable to invoke hook target arn:aws:lambda:us-west-2:123456789012:function:myHook due to User: arn:aws:sts::123456789012:assumed-role/myLambdaRole/ECS-Lambda-Execution is not authorized to perform: lambda:InvokeFunction on resource: arn:aws:lambda:us-west-2:123456789012:function:myHook because no identity-based policy allows the lambda:InvokeFunction action`

解決策: Lambda ライフサイクルフックに使用される IAM ロールに、Lambda 関数を呼び出すアクセス許可がありません。特定の Lambda 関数 ARN のロールのポリシーに `lambda:InvokeFunction` アクセス許可を追加します。Lambda のアクセス許可の詳細については、「Amazon ECS ブルー/グリーンデプロイの Lambda 関数に必要となるアクセス許可」を参照してください。

Lambda 関数のタイムアウトまたは無効なレスポンス

エラーメッセージ: `Service deployment rolled back because TEST_TRAFFIC_SHIFT lifecycle hook(s) failed. ECS was unable to parse the response from arn:aws:lambda:us-west-2:123456789012:function:myHook due to HookStatus must not be null`

解決策: Lambda 関数がタイムアウトしたか、無効なレスポンスを返しました。`hookStatus` フィールドを `SUCCEEDED` または `FAILED` のいずれかに設定して、Lambda 関数が有効なレスポンスを返すようにします。また、Lambda 関数のタイムアウトが検証ロジックに対して適切に設定されていることも確認します。詳細については、「Amazon ECS サービスデプロイのライフサイクルフック」を参照してください。

有効な Lambda レスポンスの例:

```
{
  "hookStatus": "SUCCEEDED",
  "reason": "Validation passed"
}
```

ライフサイクルフックの targetType が無効です

エラーメッセージ: `Invalid targetType. Valid values are: AWS_LAMBDA, PAUSE`

解決策: ライフサイクルフック設定の `targetType` フィールドを確認します。有効な値は、`AWS_LAMBDA` および `PAUSE` です。

一時停止フックのライフサイクルステージが無効です

エラーメッセージ: `Invalid lifecycle stage provided for Pause Hook`

解決策: 一時停止フックは、`PRE_SCALE_UP`、`POST_SCALE_UP`、`POST_TEST_TRAFFIC_SHIFT`、`PRE_PRODUCTION_TRAFFI` または `POST_PRODUCTION_TRAFFIC_SHIFT` でのみ設定できます。一時停止フックに `TEST_TRAFFIC_SHIFT` または `PRODUCTION_TRAFFIC_SHIFT` を使用することはできません。

一時停止フックがタイムアウトしました

エラーメッセージ: `Service deployment rolled back because POST_TEST_TRAFFIC_SHIFT pause hook timed out.`

解決策: `ContinueServiceDeployment` 呼び出しを受信する前に一時停止フックがタイムアウトしました。設定したタイムアウトの有効期限が切れる前に、自動化ワークフローまたは承認ワークフローが `ContinueServiceDeployment` を呼び出していることを確認します。`DescribeServiceDeployments` で `expiresAt` フィールドをチェックして、フックがいつ期限切れに設定されたかを確認します。さらに時間が必要な場合は、一時停止フック設定の `timeoutInMinutes` 値を増やします (最大 20,160 分/14 日) 。ロールバックではなくタイムアウト時にデプロイを自動的に継続する場合は、`timeoutConfiguration.action` を `CONTINUE` に設定します。

hookId 形式が無効です

エラーメッセージ: `Invalid hookId: hookId must follow the format: ecs-{hookType}-{id}`

解決策: `ContinueServiceDeployment` に提供された `hookId` が、予想される形式と一致しません。`DescribeServiceDeployments` を呼び出して `lifecycleHookDetails` 配列を確認し、正しい `hookId` を取得します。hookId を手動でコンストラクトしないでください。

hookId が見つからないか、有効期限が切れています

エラーメッセージ: `Provided hookId is either completed, unavailable, or not associated with the given Resource ARN`

解決策: hookId は無効になりました。これは、フックが既にタイムアウトしたか、既に続行またはロールバックされているか、新しいデプロイがトリガーされた (以前の hookIds を無効化している) か、hookId が別のサービスデプロイ ARN に属している場合に発生する可能性があります。アクティブなデプロイで `DescribeServiceDeployments` を呼び出して、現在の hookId を取得します。

ターミナルステータスのデプロイ

エラーメッセージ: `Continuing a service deployment is not allowed for service deployment in terminal states.`

解決策: デプロイは既に完了、ロールバック、または停止しています。`ContinueServiceDeployment` を呼び出すことができるのは、AWAITING_ACTION ステータスの一時停止フックが IN_PROGRESS のデプロイのみです。`DescribeServiceDeployments` を呼び出して、現在のデプロイステータスを確認します。

サービスデプロイが見つかりません

エラーメッセージ: `The service deployment does not exist.`

解決策: 提供された `serviceDeploymentArn` は、既存のサービスのデプロイと一致しません。`ListServiceDeployments` を使用して正しい ARN を見つけます。

Amazon ECS リニアデプロイ

リニアデプロイでは、時間の経過とともに古いサービスリビジョンから新しいサービスリビジョンにトラフィックが同じ割合ずつ徐々に移行されるため、次のステップに進む前に各ステップをモニタリングできます。Amazon ECS リニアデプロイでは、トラフィックの移行速度を制御し、本番稼働トラフィックの量が増加している新しいサービスリビジョンを検証します。このアプローチにより、各増分でパフォーマンスをモニタリングできる制御された方法で変更をデプロイできます。

リニアデプロイに関連するリソース

Amazon ECS リニアデプロイに関連するリソースは次のとおりです。

- トラフィックの移行 – Amazon ECS が本番稼働トラフィックを移行するために使用するプロセス。Amazon ECS リニアデプロイの場合、トラフィックは同じ割合の増分で移行され、各増分間の待機時間を設定できます。
- ステップパーセント – リニアデプロイ中に各増分で移行するトラフィックの割合。このフィールドの値は 2 倍で、有効な値は 3.0 ~ 100.0 です。
- ステップベイク時間 – リニアデプロイ中における各トラフィック移行の増分間の待機時間。有効な値は 0 ~ 1,440 分です。
- デプロイベイク時間 – すべての本番稼働トラフィックを新しいサービスリビジョンに移行した後、古いサービスリビジョンを終了するまでに Amazon ECS が待機する時間 (分単位)。これは本番トラフィックが移行した後に、ブルーおよびグリーンのサービスリビジョンの両方が同時に実行される期間です。
- ライフサイクルステージ – 「本番トラフィックの移行後」など、デプロイオペレーションの一連のイベント。
- ライフサイクルフック – 特定のライフサイクルステージの Lambda 関数または一時停止ポイント。Lambda フックは、カスタムコードを実行するために定義した Lambda 関数を呼び出します。一時停止フックはデプロイを一時停止し、`ContinueServiceDeployment` を呼び出して続行を待ちます。`PRODUCTION_TRAFFIC_SHIFT` または `PRE_PRODUCTION_TRAFFIC_SHIFT` 用に設定されたフックは、本番トラフィックのシフトステップごとに呼び出されます。
- ターゲットグループ – リクエストを 1 つ以上の登録済みターゲットにルーティングするために使用される Elastic Load Balancing リソース (EC2 インスタンスなど)。リスナーを作成するときは、デフォルトアクションのターゲットグループを指定します。トラフィックは、リスナー規則で指定されたターゲットグループに転送されます。
- リスナー – 設定したプロトコルおよびポートを使用し、接続リクエストを確認する Elastic Load Balancing リソース。リスナーに対して定義したルールにより、Amazon ECS が登録済みターゲットにリクエストをルーティングする方法が決定します。
- ルール – リスナーに関連付けられた Elastic Load Balancing リソース。ルールはリクエストのルーティング方法を定義し、アクション、条件、優先度で構成されます。

考慮事項

デプロイタイプを選択する際、次の内容を考慮してください。

- リソースの使用量: リニアデプロイではブルーおよびグリーンのサービスリビジョンの両方が一時的に実行されるため、デプロイ中にリソースの使用量が 2 倍になる場合があります。

- デプロイモニタリング: リニアデプロイではデプロイステータスの詳細な情報が提示されるため、デプロイプロセスの各ステージとトラフィック移行の増分をモニタリングできます。
- ロールバック: ブルーリビジョンがベイク時間が終了するまで実行が継続されるため、問題が検出された場合、リニアデプロイでは前のバージョンへのロールバックが容易になります。
- 段階的な検証: リニアデプロイでは、本番稼働トラフィックの量を増やして新しいリビジョンを検証できるため、デプロイの信頼性が向上します。
- デプロイ期間: ステップ間のトラフィック移行の増分と待機時間のため、リニアデプロイは一括デプロイよりも完了に時間を要します。

リニアデプロイの仕組み

Amazon ECS リニアデプロイプロセスは構造化されたアプローチに従っており、安全かつ信頼性の高いアプリケーション更新を実現する 6 つの異なるフェーズで構成されます。各フェーズは、アプリケーションを現在のバージョン (ブルー) から新しいバージョン (グリーン) に検証および移行するためにそれ特有の目的を果たします。

1. 準備フェーズ: 既存のブルー環境と一緒にグリーン環境を作成します。
2. デプロイフェーズ: 新しいサービスリビジョンをグリーン環境にデプロイします。Amazon ECS は更新されたサービスリビジョンを使用して新しいタスクを起動し、ブルー環境は引き続き本番トラフィックを処理します。
3. テストフェーズ: テストトラフィックルーティングを使用してグリーン環境を検証します。Application Load Balancer によってテストリクエストがグリーン環境に送信され、本番トラフィックはブルー環境の状態にとどまります。
4. リニアトラフィック移行のフェーズ: 設定されたデプロイ戦略に基づき、本番稼働トラフィックをブルーからグリーンに一定割合で移行します。
5. モニタリングフェーズ: ベイク期間中のアプリケーションの正常性、パフォーマンスメトリクス、アラーム状態がモニタリングされます。ロールバックオペレーションは問題が検出されると開始されます。
6. 完了フェーズ: ブルー環境を終了してデプロイを確定します。

リニアトラフィック移行フェーズは、以下のステップに従います。

- 初期 – デプロイは、トラフィックが 100% ブルー (現在の) サービスリビジョンにルーティングされた状態で開始されます。グリーン (新しい) サービスリビジョンはテストトラフィックを受信しますが、初期段階では本番トラフィックを受信しません。

- 増分トラフィック移行 – トラフィックは、同じ割合の増分でブルーからグリーンに徐々に移行されます。例えば、ステップ設定が 10.0% の場合、トラフィックの移行は次のように行われます。
  - ステップ 1: 10.0% がグリーンに、90.0% がブルーに
  - ステップ 2: 20.0% がグリーンに、80.0% がブルーに
  - ステップ 3: 30.0% がグリーンに、70.0% がブルーに
  - 100% がグリーンになるまで続きます。
- ステップベイク時間 – 各トラフィック移行の増分間で、デプロイは設定可能な期間 (ステップベイク時間) 待機し、トラフィック負荷の増加に伴う新しいリビジョンのパフォーマンスのモニタリングと検証を可能にします。トラフィックが 100.0% 移行されると、最後のステップのベイク時間はスキップされることに注意してください。
- ライフサイクルフック – オプションの Lambda 関数または一時停止フックは、デプロイ中のさまざまなライフサイクル段階で設定され、自動検証、モニタリング、またはカスタムロジックを実行できます。PRODUCTION_TRAFFIC_SHIFT または PRE_PRODUCTION_TRAFFIC_SHIFT 用に設定されたフックは、本番トラフィックのシフトステップごとに呼び出されます。

デプロイのライフサイクルステージ

リニアデプロイプロセスは、個別のライフサイクル段階を経て進行し、それぞれに特定の責任と検証チェックポイントがあります。これらのステージを理解すると、デプロイの進行状況をモニタリングして問題を効果的にトラブルシューティングできます。

各ライフサイクルステージは最大 24 時間、さらに PRODUCTION_TRAFFIC_SHIFT の各トラフィック移行ステップは最大 24 時間継続できます。値を 24 時間未満にとどめることをお勧めします。非同期プロセスがフックをトリガーするために時間が必要になるからです。システムがタイムアウトしてデプロイに失敗し、ステージが 24 時間に達した後にロールバックが開始されます。

CloudFormation デプロイには追加のタイムアウト制限があります。24 時間のステージ制限は有効性が維持されますが、CloudFormation によってデプロイ全体に 36 時間の制限が実施されます。CloudFormation によってデプロイが失敗し、36 時間以内にプロセスが完了しない場合はロールバックが開始されます。

一時停止フックの場合、タイムアウトを最大 20,160 分 (14 日間) まで設定できます。デプロイ全体のタイムアウトは 30 日です。

ライフサイクルのステージ	説明	ライフサイクルフックのサポート
RECONCILE_SERVICE	このステージは、複数のサービスリビジョンが ACTIVE 状態で新しいサービスのデプロイを開始した場合にのみ発生します。	はい
PRE_SCALE_UP	グリーンサービスリビジョンは開始されていません。ブルーサービスリビジョンにより、本番トラフィックの 100% が処理されています。テストトラフィックがありません。	はい
SCALE_UP	グリーンサービスリビジョンが 100% までにスケールアップし、新しいタスクを起動する時間。現時点では、グリーンサービスリビジョンはトラフィックを処理していません。	いいえ
POST_SCALE_UP	グリーンサービスリビジョンが開始されました。ブルーサービスリビジョンにより、本番トラフィックの 100% が処理されています。テストトラフィックがありません。	はい
TEST_TRAFFIC_SHIFT	ブルーおよびグリーンのサービスリビジョンが実行されています。ブルーサービスリビジョンにより、本番トラフィックの 100% が処理されます。グリーンサービスリビジョンは、テストトラフィックの 0% から 100% に移行しています。	はい (Lambda のみ)
POST_TEST_TRAFFIC_SHIFT	テストトラフィックの移行が完了しました。グリーンサービスリビジョンにより、テストトラフィックの 100% が処理されます。	はい
PRE_PRODUCTION_TRAFFIC_SHIFT	本番トラフィックシフトが増分される前に発生します。このステージに設定されたライフサイクルフックは、トラフィックシフトの各ステップで呼び出されます。	はい

ライフサイクルのステージ	説明	ライフサイクルフックのサポート
PRODUCTION_TRAFFIC_SHIFT	トラフィックは、グリーンがトラフィックの 100% を受信するまで、同じ割合でブルーからグリーンに徐々に移行されます。各トラフィック移行は、24 時間のタイムアウトでライフサイクルフックを呼び出します。	はい (Lambda のみ)
POST_PRODUCTION_TRAFFIC_SHIFT	本番トラフィックの移行が完了しました。	はい
BAKE_TIME	ブルーおよびグリーンのサービスリビジョンの両方が同時に実行される期間。	いいえ
CLEAN_UP	ブルーサービスリビジョンにより、実行中のタスク 0 へと完全にスケールダウンされました。グリーンサービスリビジョンは、このステージの後に本番サービスリビジョンになります。	いいえ

Amazon ECS リニアデプロイに必要なリソース

マネージドトラフィック移行が実施されるリニアデプロイを使用するには、サービスが次のいずれかの機能を使用する必要があります。

- Elastic Load Balancing
- Service Connect

次のリストは、Amazon ECS リニアデプロイに設定する必要がある内容について概要を示しています。

- サービスは Application Load Balancer、Network Load Balancer、Service Connect を使用します。適切なリソースを設定します。
  - Application Load Balancer – 詳細については、「ブルー/グリーンデプロイ、リニアデプロイおよびカナリアデプロイ用の Application Load Balancer リソース」を参照してください。

  - Network Load Balancer – 詳細については、「Amazon ECS ブルー/グリーン、リニアおよびカナリアデプロイの Network Load Balancer リソース」を参照してください。
  - Service Connect – 詳細については、「Amazon ECS ブルー/グリーンデプロイ、リニアデプロイおよびカナリアデプロイ用の Service Connect リソース」を参照してください。
- サービスのデプロイコントローラーを ECS に設定します。
- サービス定義でデプロイ戦略を `linear` として設定します。
- 必要に応じて、次のように追加のパラメータを設定します。
  - 新しいデプロイのベイク時間
  - 各増分で移行するトラフィックのパーセンテージ。
  - 各トラフィック移行の増分間の待機時間を分単位で表します。
  - 自動ロールバックの CloudWatch アラーム
  - デプロイのライフサイクルフック (指定されたデプロイステージで実行される Lambda 関数または一時停止フック)

ベストプラクティス

Amazon ECS リニアデプロイを正常に処理するには、次のベストプラクティスに従ってください。

- アプリケーションが同時に実行されている両方のサービスリビジョンを処理できることを確認します。
- デプロイ中に両方のサービスリビジョンを処理するため、十分なクラスターキャパシティの確保を計画します。
- ロールバック手順は、本番環境での実装前にテストします。
- アプリケーションの正常性を正確に反映する、適切なヘルスチェックを設定します。
- 新しいサービスリビジョンを十分にテストできるベイク時間を設定します。
- CloudWatch アラームを実装し、問題を自動的に検出してロールバックをトリガーします。
- デプロイ速度と検証ニーズのバランスを取るためのステップの割合とベイク時間を選択します。
- リスクの露出を最小限に抑えるために、重要なアプリケーションには小さなステップの割合 (5～10%) を使用します。
- ウォームアップや安定に時間がかかるアプリケーションには、長いステップベイク時間を設定します。
- CloudWatch アラームを実装し、問題を自動的に検出して任意のトラフィック増分でロールバックをトリガーします。

- 各トラフィック移行中にアプリケーションメトリクスを注意深くモニタリングし、パフォーマンスの低下を早期に検出します。
- アプリケーションが同時に実行されている両方のサービスリビジョンを処理できることを確認します。
- ロールバック手順を本番環境に実装する前に異なるトラフィックの割合でテストします。

Amazon ECS リニアデプロイの作成

Amazon ECS リニアデプロイを使用すると、指定した時間間隔でトラフィックを均等に徐々に移行できます。これにより、デプロイプロセスの各ステップで実施する検証を制御できます。

前提条件

リニアデプロイを開始する前に、次の操作を行います。

1. 適切なアクセス許可を設定します。
   - Elastic Load Balancing のアクセス許可の詳細については、「ロードバランサー用の Amazon ECS インフラストラクチャの IAM ロール」を参照してください。
   - Lambda のアクセス許可の詳細については、「Amazon ECS ブルー/グリーンデプロイの Lambda 関数に必要となるアクセス許可」を参照してください。
2. Amazon ECS リニアデプロイでは、サービスが次のいずれかの機能を使用する必要があります。適切なリソースを設定してください。
   - Application Load Balancer – 詳細については、「ブルー/グリーンデプロイ、リニアデプロイおよびカナリアデプロイ用の Application Load Balancer リソース」を参照してください。
   - Network Load Balancer – 詳細については、「Amazon ECS ブルー/グリーン、リニアおよびカナリアデプロイの Network Load Balancer リソース」を参照してください。
   - Service Connect – 詳細については、「Amazon ECS ブルー/グリーンデプロイ、リニアデプロイおよびカナリアデプロイ用の Service Connect リソース」を参照してください。

手順

コンソールまたは AWS CLI を使用して、Amazon ECS リニアデプロイサービスを作成できます。

Console

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. サービスを起動するリソースを決定します。

サービスの起動元	Steps
クラスター	a. [Clusters] ページで、サービスを作成するクラスターを選択します。 クラスター詳細のページが表示されます。 b. [Services (サービス)] タブで、[Create (作成)] を選択します。
タスク定義	a. [タスク定義] ページで、タスク定義を選択します。 b. [デプロイ] メニューで、[サービスの作成] を選択します。

[サービスの作成] ページが表示されます。

3. [サービスの詳細] で、次の操作を行います。

   a. [タスク定義ファミリー] では、使用するタスク定義を選択します。次に [タスク定義リビジョン] には、使用するリビジョンを入力します。

   b. [Service name] (サービス名) でサービスの名前を入力します。

4. 既存のクラスターでサービスを実行するには、[既存のクラスター] にクラスターを選択します。新しいクラスターでサービスを実行するには、[クラスターの作成] を選択します。

5. クラスターのインフラストラクチャ全体にタスクを分散する方法を選択します。[コンピュート設定] で、オプションを選択します。

コンピューティングオプション	Steps	
キャパシティープロバイダー戦略	a. [コンピューティングオプション] で、[キャパシティプロバイダー戦略] を選択します。 b. 戦略を選択: • デフォルトのキャパシティープロバイダー戦略を使用するには、[Use cluster default] (デフォルトのクラスターを使用) を選択します。 • クラスターにデフォルトのキャパシティプロバイダー戦略がない場合、またはカスタム戦略を使用する場合は、[カスタムを使用]、[キャパシティプロバイダー戦略を追加] を選択し、[ベース]、[キャパシティプロバイダー]、[重み] を指定して、カスタムキャパシティプロバイダー戦略を定義します。	

コンピューティングオプション	Steps
	Note 戦略でキャパシティープロバイダーを使用するには、キャパシティープロバイダーをクラスターに関連付ける必要があります。
起動タイプ	a. [Compute options] (コンピューティングオプション) セクションで、[Launch type] (起動タイプ) を選択します。 b. [Launch type] (起動タイプ) で、起動タイプを選択します。 c. (オプション) Fargate が指定されている場合は、[プラットフォームバージョン] で、使用するプラットフォームのバージョンを指定します。プラットフォームバージョンが指定されない場合、LATEST プラットフォームバージョンが使用されます。

6. [デプロイ設定] で、次の操作を行います。

a. [サービスタイプ] では、[レプリカ] を選択します。

b. [Desired tasks] (必要なタスク) で、サービス内で起動および維持するタスクの数を入力します。

c. Amazon ECS がアベイラビリティーゾーン間のタスクの分散をモニタリングし、不均衡が発生したときにタスクを再分散するには、[アベイラビリティーゾーンのサービス再調整] で [アベイラビリティーゾーンのサービス再調整] を選択します。

d. [ヘルスチェックの猶予期間] には、タスクの初回起動後にサービススケジューラが異常な Elastic Load Balancing、VPC Lattice、コンテナヘルスチェックを無視する時間 (秒単位) を入力します。ヘルスチェックの猶予期間値を指定しない場合、デフォルト値 0 が使用されます。

7. [デプロイ設定] で、リニアデプロイ設定を構成します。

a. [デプロイ戦略] で、[リニア] を選択します。

b. [トラフィック増分の割合] には、各増分で移行するトラフィックの割合を入力します (例えば、10% 単位でトラフィックを移行するには 10%)。

c. [増分間隔]には、各トラフィック移行増分間の待機時間を分単位で入力します。

d. [ベイク時間] には、ブルーリビジョンが終了する前に、トラフィックの最終移行後にブルーおよびグリーンのサービスリビジョンの両方が同時に実行される時間 (分) を入力します。

e. (オプション) デプロイの特定段階で実行されるように、ライフサイクルフックを設定します。[デプロイライフサイクルフック] で、[追加] を選択します。

Lambda フックの場合:

- [ターゲットタイプ] で [Lambda] を選択します。
- [Lambda 関数] には、ARN の関数名を入力します。
- [ロール] では、Lambda 関数を呼び出すアクセス許可を持つ IAM ロールを選択します。
- [ライフサイクルステージ] では、Lambda 関数を実行するステージを選択します。

一時停止フックの場合:

- [ターゲットタイプ] で、[一時停止] を選択します。
- [ライフサイクルステージ] で、デプロイを一時停止するステージを選択します。

- (オプション) [タイムアウト] の場合は、タイムアウト期間とアクションを設定します。

8. Amazon ECS がデプロイの障害を検出して処理する方法を設定するには、[Deployment failure detection] (デプロイ障害検出) を展開し、オプションを選択します。

   a. タスクを開始できない場合にデプロイを停止するには、[Use the Amazon ECS deployment circuit breaker] (Amazon ECS デプロイサーキットブレーカーを使用する) を選択します。

      デプロイサーキットブレーカーによってデプロイが失敗状態に設定されたときに、ソフトウェアがデプロイを最後に完了したデプロイ状態に自動的にロールバックするようにするには、[失敗時のロールバック] を選択します。

   b. アプリケーションメトリクスに基づいてデプロイを停止するには、[CloudWatch アラームを使用する] を選択します。次に、[CloudWatch アラーム名] からアラームを選択します。新しいアラームを作成するには、CloudWatch コンソールに移動します。

      CloudWatch アラームによってデプロイが失敗状態に設定されたときに、ソフトウェアがデプロイを最後に完了したデプロイ状態に自動的にロールバックするようにするには、[失敗時のロールバック] を選択します。

9. (オプション) Service Connect を使用してサービスを相互接続するには、[Service Connect] を展開して以下を指定します。

   a. [Service Connect をオンにする] を選択します。

   b. [Service Connect configuration] (Service Connect 設定) で、クライアントモードを指定します。

      - サービスが名前空間内の他のサービスへの接続のみを必要とするネットワークアプリケーションを実行している場合は、[クライアント側のみ] を選択します。
      - サービスがネットワークまたは Web サービスアプリケーションを実行していて、このサービスにエンドポイントを提供し、名前空間内の他のサービスに接続する必要がある場合は、[Client and server] (クライアントとサーバー) を選択します。

   c. デフォルトのクラスター名前空間ではない名前空間を使用するには、[Namespace] (名前空間) でサービス名前空間を選択します。この場合、AWS アカウントの同じ AWS リージョンで個別に作成された名前空間、またはAWS Resource Access Manager (AWS RAM) を使用してアカウントと共有されている同じリージョンの名前空間のいずれかを選択できます。共有 AWS Cloud Map 名前空間の詳細については、「AWS Cloud Map

デベロッパーガイド」の「クロスアカウント AWS Cloud Map 名前空間の共有」を参照してください。

d. (オプション) リニアデプロイのテストトラフィックヘッダーのルールを設定します。[テストトラフィックのルーティング] では、以下を指定します。

i. [テストトラフィックヘッダーのルールの有効化] を選択し、テスト中に特定のリクエストをグリーンサービスリビジョンにルーティングします。

ii. [ヘッダー一致ルール] では、テストトラフィックをルーティングする条件を設定します。

- [ヘッダー名]: 一致する HTTP ヘッダーの名前を入力します (例えば、`X-Test-Version` または `User-Agent`)。
- [一致タイプ]: 一致する条件を選択します。
  - [完全一致]: ヘッダー値が指定された値と完全に一致するリクエストをルーティングする
  - [ヘッダーが存在する]: 値に関係なく、指定されたヘッダーを含むリクエストをルーティングする
  - [パターン一致]: ヘッダー値が指定されたパターンと一致するリクエストをルーティングする
- [ヘッダー値] (完全一致またはパターン一致を使用する場合): 照合する値またはパターンを入力します。

複数のヘッダー一致ルールを追加し、複雑なルーティングロジックを作成できます。設定されたルールに一致するリクエストは、テストのためにグリーンサービスリビジョンにルーティングされます。

iii. [ヘッダールールの追加] を選択し、追加のヘッダー一致条件を設定します。

> **Note**
>
> テストトラフィックヘッダーのルールにより、完全なデプロイを完了する前に、制御されたトラフィックで新機能を検証できます。これにより、ブルーサービスリビジョンへの通常のトラフィックフローを維持しながら、特定のリクエスト (内部テストツールやベータユーザーなどによるもの) でグリーンサービスリビジョンをテストできます。

e. （オプション) ログ設定を指定します。[ログコレクションを使用] を選択します。デフォルトのオプションでは、コンテナログを CloudWatch Logs に送信します。その他のログドライバオプションは、AWS FireLens を使用して構成されます。詳細については、「Amazon ECS ログを AWS サービスまたは AWS Partner に送信する」を参照してください。

以下では、各コンテナログの送信先について詳しく説明します。

- Amazon CloudWatch — コンテナログを CloudWatch Logs に送信するようにタスクを設定します。デフォルトのログドライバーオプションが提供され、ユーザーに代わり CloudWatch ロググループを作成します。別のロググループ名を指定するには、ドライバーオプションの値を変更します。
- [Amazon Data Firehose] — Firehose にコンテナログを送信するようタスクを設定します。Firehose 配信ストリームにログを送信するデフォルトのログドライバーオプションが提供されています。別の配信ストリーム名を指定するには、ドライバーオプションの値を変更します。
- Amazon Kinesis Data Streams — Kinesis Data Streams にコンテナログを送信するようタスクを設定します。Kinesis Data Streams のストリームにログを送信するデフォルトのログドライバーオプションが提供されています。別のストリーム名を指定するには、ドライバーオプションの値を変更します。
- Amazon OpenSearch Service — コンテナログを OpenSearch Service ドメインに送信するようタスクを設定します。ログドライバーオプションを提供する必要があります。
- Amazon S3 — Amazon S3 バケットにコンテナログを送信するようタスクを設定します。デフォルトのログドライバーオプションはデフォルトで設定されていますが、有効な Amazon S3 バケット名を指定する必要があります。

10. (オプション) リニアデプロイ用に [ロードバランシング] を設定します。

Elastic Load Balancing のタイプ	Steps	
Application Load Balancer	a. [ロードバランサーのタイプ] では [Application Load Balancer] を選択します。	

Elastic Load Balancing のタイプ	Steps	
	b. 選択新しいロードバランサーを作成するをクリックして新しいApplication Load Balancer を作成するか、既存のロードバランサーを使用する既存のApplication Load Balancer を選択します。 c. [コンテナ] では、サービスをホストするコンテナを選択します。 d. [Load balancer name] (ロードバランサー名) では、一意な名前を入力します。 e. [Listener] (リスナー) で、Application Load Balancer が接続リクエストをリッスンするポートとプロトコルを入力します。デフォルトでは、ロードバランサーはポート 80 と HTTP を使用するように設定されます。 • [本番ルール] には、ルールの [評価順序] および [パスパターン] を入力します。 このルールは、本番サービスリビジョンのトラフィック用です。	

Elastic Load Balancing のタイプ	Steps	
	• [テストルール] には、ルールの [評価順序] および [パスパターン] を入力します。 このルールは、テストサービスリビジョンのトラフィック用です。 f. [ターゲットグループ] では、以下を設定します。 • [Target group name] (ターゲットグループ名) では、Application Load Balancer がリクエストをルーティングするターゲットグループの名前とプロトコルを入力します。 • [プロトコル] では、Application Load Balancer がリクエストをルーティングするターゲットグループのプロトコルを選択します。デフォルトでは、ターゲットグループは、タスク定義で定義されている最初のコンテナにリクエストをルーティングします。 • [登録解除の遅延] には、ロードバランサーがターゲットの状態	

Elastic Load Balancing のタイプ	Steps	
	を UNUSED に変更するまでの秒数を入力します。デフォルトは 300 秒です。 • [Health check path] (ヘルスチェックパス) では、Application Load Balancer とコンテナ間の接続の正常性を検証するために、Application Load Balancer が定期的にリクエストを送信するコンテナ内に存在するパスを入力します。デフォルトはルートディレクトリ (/)。 • [代替グループ名] には、テストサービスリビジョンのターゲットグループのグループ名を入力します。	

Elastic Load Balancing のタイプ	Steps	
Network Load Balancer	a. [Load balancer type] (ロードバランサータイプ) では、[Network Load Balancer] を選択します。 b. [Load Balancer] (ロードバランサー) では、既存の Network Load Balancer を選択します。 c. [Choose container to load balance] (負荷分散するコンテナを選択) で、サービスをホストするコンテナを選択します。 d. [本番リスナー] では、[本番リスナーポート] および [本番リスナープロトコル] を選択します。 これは、本番サービスリビジョントのラフィックのリスナーです。 e. [テストリスナー] では、[テストリスナーポート] および [テストリスナープロトコル] を選択します。 これは、テストサービスリビジョンのトラフィックのリスナーです。	

Elastic Load Balancing のタイプ	Steps	
	f. [ターゲットグループ] では、以下を設定します。 • [Target group name] (ターゲットグループ名) では、Network Load Balancer がリクエストをルーティングするターゲットグループの名前とプロトコルを入力します。 • [プロトコル] では、Network Load Balancer がリクエストをルーティングするターゲットグループのプロトコルを選択します。デフォルトでは、ターゲットグループは、タスク定義で定義されている最初のコンテナにリクエストをルーティングします。 • [登録解除の遅延] には、ロードバランサーがターゲットの状態を UNUSED に変更するまでの秒数を入力します。デフォルトは 300 秒です。 • [Health check path] (ヘルスチェックパス) では、Application Load Balancer とコンテナ	

Elastic Load Balancing のタイプ	Steps	
	間の接続の正常性を検証するために、Application Load Balancer が定期的にリクエストを送信するコンテナ内に存在するパスを入力します。デフォルトはルートディレクトリ (/)。 • [代替グループ名] には、テスト (緑) サービスリビジョンのターゲットグループのグループ名を入力します。	

11. （オプション）サービスとタスクを識別しやすくするには、[Tags] (タグ) セクションを展開し、タグを設定します。

    新しく起動したすべてのタスクに対して、Amazon ECS がクラスター名とタスク定義タグで自動的にタグ付けするようにするには、[Amazon ECS マネージドタグを有効にする] を選択し、[タグの伝播元] で [タスク定義] を選択します。

    新しく起動したすべてのタスクに対して、Amazon ECS がクラスター名とサービスタグで自動的にタグ付けするようにするには、[Amazon ECS マネージドタグを有効にする] を選択し、[タグの伝播元] で [サービス] を選択します。

    タグを追加または削除します。

    - [タグを追加] [Add tag] (タグを追加) を選択し、以下を実行します。
      - [キー] にはキー名を入力します。
      - [値] にキー値を入力します。
    - [タグの削除] タグの横にある [タグの削除] を選択します。
12. [作成] を選択します。

AWS CLI

1. linear-service-definition.json という名前のファイルを作成し、次の内容を記述します。

   user-input を独自の値に置き換えます。

```
{
  "serviceName": "myLinearService",
  "cluster": "arn:aws:ecs:us-west-2:123456789012:cluster/sample-fargate-
cluster",
  "taskDefinition": "sample-fargate:1",
  "desiredCount": 5,
  "launchType": "FARGATE",
  "networkConfiguration": {
    "awsvpcConfiguration": {
      "subnets": [
        "subnet-09ce6e74c116a2299",
        "subnet-00bb3bd7a73526788",
        "subnet-0048a611aaec65477"
      ],
      "securityGroups": [
        "sg-09d45005497daa123"
      ],
      "assignPublicIp": "ENABLED"
    }
  },
  "deploymentController": {
    "type": "ECS"
  },
  "deploymentConfiguration": {
    "strategy": "LINEAR",
    "maximumPercent": 200,
    "minimumHealthyPercent": 100,
    "linearConfiguration": {
      "stepPercentage": 10.0,
      "stepBakeTimeInMinutes":6
    },
    "bakeTimeInMinutes": 10,
    "alarms": {
      "alarmNames": [
        "myAlarm"
      ],
```

```
      "rollback": true,
      "enable": true
    }
  },
  "loadBalancers": [
    {
      "targetGroupArn": "arn:aws:elasticloadbalancing:us-
west-2:123456789012:targetgroup/blue-target-group/54402ff563af1197",
      "containerName": "fargate-app",
      "containerPort": 80,
      "advancedConfiguration": {
        "alternateTargetGroupArn": "arn:aws:elasticloadbalancing:us-
west-2:123456789012:targetgroup/green-target-group/cad10a56f5843199",
        "productionListenerRule": "arn:aws:elasticloadbalancing:us-
west-2:123456789012:listener-rule/app/my-linear-
demo/32e0e4f946c3c05b/9cfa8c482e204f7d/831dbaf72edb911",
        "roleArn": "arn:aws:iam::123456789012:role/LoadBalancerManagementforECS"
      }
    }
  ]
}
```

2. `create-service` を実行します。

```
aws ecs create-service --cli-input-json file://linear-service-definition.json
```

次のステップ

- サービスを更新してデプロイを開始します。詳細については、「Amazon ECS サービスを更新する」を参照してください。
- デプロイプロセスをモニタリングし、リニアパターンに従っていることを確認します。
  - グリーンサービスリビジョンが作成され、スケールアップされる
  - トラフィックは指定された間隔で段階的に移行されます。
  - 各増分は、次の移行の前に指定された間隔を待機します。
  - すべてのトラフィックが移行されると、ベイク時間が開始されます。
  - ベイク時間が過ぎたら、ブルーリビジョンが終了する

Amazon ECS カナリアデプロイ

Canary デプロイでは、まず初期テストのためにトラフィックのごく一部を新しいリビジョンにルーティングし、Canary フェーズが正常に完了したら、残りのすべてのトラフィックを一度に移行します。Amazon ECS カナリアデプロイでは、リスクの露出を最小限に抑えながら、実際のユーザートラフィックで新しいサービスリビジョンを検証します。このアプローチでは、パフォーマンスをモニタリングし、問題が検出された場合には迅速にロールバックしながら変更をデプロイできます。

カナリアデプロイに関連するリソース

Amazon ECS カナリアデプロイに関連するリソースは次のとおりです。

- トラフィックの移行 – Amazon ECS が本番稼働トラフィックを移行するために使用するプロセス。Amazon ECS カナリアデプロイの場合、トラフィックは 2 つのフェーズで移行されます。最初に Canary の割合に応じて移行され、次にデプロイを完了するために移行されます。
- Canary の割合 – 評価期間中に新しいバージョンにルーティングされたトラフィックの割合。
- Canary ベイク時間 – フルデプロイに進む前に Canary バージョンをモニタリングする期間。
- デプロイベイク時間 – すべての本番稼働トラフィックを新しいサービスリビジョンに移行した後、古いサービスリビジョンを終了するまでに Amazon ECS が待機する時間 (分単位)。これは本番トラフィックが移行した後に、ブルーおよびグリーンのサービスリビジョンの両方が同時に実行される期間です。
- ライフサイクルステージ – 「本番トラフィックの移行後」など、デプロイオペレーションの一連のイベント。
- ライフサイクルフック – 特定のライフサイクルステージの Lambda 関数または一時停止ポイント。Lambda フックは、カスタムコードを実行するために定義した Lambda 関数を呼び出します。一時停止フックはデプロイを一時停止し、`ContinueServiceDeployment` を呼び出して続行を待ちます。
- ターゲットグループ – リクエストを 1 つ以上の登録済みターゲットにルーティングするために使用される Elastic Load Balancing リソース (EC2 インスタンスなど)。リスナーを作成するときは、デフォルトアクションのターゲットグループを指定します。トラフィックは、リスナー規則で指定されたターゲットグループに転送されます。
- リスナー – 設定したプロトコルおよびポートを使用し、接続リクエストを確認する Elastic Load Balancing リソース。リスナーに対して定義したルールにより、Amazon ECS が登録済みターゲットにリクエストをルーティングする方法が決定します。
- ルール – リスナーに関連付けられた Elastic Load Balancing リソース。ルールはリクエストのルーティング方法を定義し、アクション、条件、優先度で構成されます。

考慮事項

デプロイタイプを選択する際、次の内容を考慮してください。

- リソースの使用状況: カナリアデプロイでは、評価期間中に元のタスクセットと Canary タスクセットの両方が同時に実行され、リソースの使用量が増加します。
- トラフィック量: Canary の割合が、新しいバージョンの有意な検証に十分なトラフィックを生成していることを確認します。
- 複雑さのモニタリング: Canary デプロイでは、2 つの異なるバージョン間でメトリクスを同時にモニタリングおよび比較する必要があります。
- ロールバック速度: カナリアデプロイでは、トラフィックを元のタスクセットに戻すことで、クイックロールバックが可能になります。
- リスク軽減: カナリアデプロイでは、少数のユーザーへの露出を制限することで、優れたリスクを軽減できます。
- デプロイ期間: カナリアデプロイには、全体的なデプロイ時間を延長するものの、検証の機会が設けられている評価期間が含まれます。

カナリアデプロイの仕組み

Amazon ECS カナリアデプロイプロセスは構造化されたアプローチに従っており、安全かつ信頼性の高いアプリケーション更新を実現する 6 つの異なるフェーズで構成されます。各フェーズは、アプリケーションを現在のバージョン (ブルー) から新しいバージョン (グリーン) に検証および移行するためにそれ特有の目的を果たします。

1. 準備フェーズ: 既存のブルー環境と一緒にグリーン環境を作成します。
2. デプロイフェーズ: 新しいサービスリビジョンをグリーン環境にデプロイします。Amazon ECS は更新されたサービスリビジョンを使用して新しいタスクを起動し、ブルー環境は引き続き本番トラフィックを処理します。
3. テストフェーズ: テストトラフィックルーティングを使用してグリーン環境を検証します。Application Load Balancer によってテストリクエストがグリーン環境に送信され、本番トラフィックはブルー環境の状態にとどまります。
4. Canary トラフィック移行フェーズ: Canary フェーズ中に設定されたトラフィックの割合を新しいグリーンサービスリビジョンに移行し、次にトラフィックの 100.0% をグリーンサービスリビジョンに移行します。

5. モニタリングフェーズ: ベイク期間中のアプリケーションの正常性、パフォーマンスメトリクス、アラーム状態がモニタリングされます。ロールバックオペレーションは問題が検出されると開始されます。
6. 完了フェーズ: ブルー環境を終了してデプロイを確定します。

Canary トラフィックの移行フェーズは、次のステップに従います。

- 初期 – デプロイは、トラフィックが 100% ブルー (現在の) サービスリビジョンにルーティングされた状態で開始されます。グリーン (新しい) サービスリビジョンはテストトラフィックを受信しますが、初期段階では本番トラフィックを受信しません。
- Canary トラフィックの移行 – これは 2 ステップのトラフィック移行戦略です。
  - ステップ 1: 10.0% がグリーンに、90.0% がブルーに
  - ステップ 2: 100.0% がグリーンに、0.0% がブルーに
- Canary ベイク時間 – Canary トラフィックの移行後に設定可能な時間 (Canary ベイク時間) 待機し、トラフィック負荷の増加に伴う新しいリビジョンのパフォーマンスのモニタリングと検証を可能にします。
- ライフサイクルフック – オプションの Lambda 関数または一時停止フックは、デプロイ中のさまざまなライフサイクル段階で設定され、自動検証、モニタリング、またはカスタムロジックを実行できます。PRODUCTION_TRAFFIC_SHIFT または PRE_PRODUCTION_TRAFFIC_SHIFT 用に設定されたフックは、本番トラフィックのシフトステップごとに呼び出されます。

デプロイのライフサイクルステージ

カナリアデプロイプロセスは、個別のライフサイクルステージを経て進行し、それぞれのステージに特定の責任と検証チェックポイントがあります。これらのステージを理解すると、デプロイの進行状況をモニタリングして問題を効果的にトラブルシューティングできます。

各ライフサイクルステージは最大 24 時間、さらに PRODUCTION_TRAFFIC_SHIFT の各トラフィック移行ステップは最大 24 時間継続できます。値を 24 時間未満にとどめることをお勧めします。非同期プロセスがフックをトリガーするために時間が必要になるからです。システムがタイムアウトしてデプロイに失敗し、ステージが 24 時間に達した後にロールバックが開始されます。

CloudFormation デプロイには追加のタイムアウト制限があります。24 時間のステージ制限は有効性が維持されますが、CloudFormation によってデプロイ全体に 36 時間の制限が実施されます。CloudFormation によってデプロイが失敗し、36 時間以内にプロセスが完了しない場合はロールバックが開始されます。

一時停止フックの場合、タイムアウトを最大 20,160 分 (14 日間) まで設定できます。デプロイ全体のタイムアウトは 30 日です。

ライフサイクルのステージ

ライフサイクルのステージ	説明	ライフサイクルフックのサポート
RECONCILE_SERVICE	このステージは、複数のサービスリビジョンが ACTIVE 状態で新しいサービスのデプロイを開始した場合にのみ発生します。	はい
PRE_SCALE_UP	グリーンサービスリビジョンは開始されていません。ブルーサービスリビジョンにより、本番トラフィックの 100% が処理されています。テストトラフィックがありません。	はい
SCALE_UP	グリーンサービスリビジョンが 100% までにスケールアップし、新しいタスクを起動する時間。現時点では、グリーンサービスリビジョンはトラフィックを処理していません。	いいえ
POST_SCALE_UP	グリーンサービスリビジョンが開始されました。ブルーサービスリビジョンにより、本番トラフィックの 100% が処理されています。テストトラフィックがありません。	はい
TEST_TRAFFIC_SHIFT	ブルーおよびグリーンのサービスリビジョンが実行されています。ブルーサービスリビジョンにより、本番トラフィックの 100% が処理されます。グリーンサービスリビジョンは、テストトラフィックの 0% から 100% に移行しています。	はい (Lambda のみ)
POST_TEST_TRAFFIC_SHIFT	テストトラフィックの移行が完了しました。グリーンサービスリビジョンにより、テストトラフィックの 100% が処理されます。	はい

ライフサイクルのステージ	説明	ライフサイクルフックのサポート
PRE_PRODUCTION_TRAFFIC_SHIFT	本番トラフィックの各シフトステップの前に発生します。Canary の場合、これは Canary トラフィックのシフト前と残りのトラフィックのシフト前に発生します。	はい
PRODUCTION_TRAFFIC_SHIFT	Canary 本番稼働トラフィックはグリーンリビジョンにルーティングされ、ライフサイクルフックは 24 時間のタイムアウトで呼び出されます。2 番目のステップでは、残りの本番稼働トラフィックをグリーンリビジョンに移行します。	はい (Lambda のみ)
POST_PRODUCTION_TRAFFIC_SHIFT	本番トラフィックの移行が完了しました。	はい
BAKE_TIME	ブルーおよびグリーンのサービスリビジョンの両方が同時に実行される期間。	いいえ
CLEAN_UP	ブルーサービスリビジョンにより、実行中のタスク 0 へと完全にスケールダウンされました。グリーンサービスリビジョンは、このステージの後に本番サービスリビジョンになります。	いいえ

設定パラメータ

カナリアデプロイデプロイには、次の設定パラメータが必要です。

- Canary の割合 – Canary フェーズ中に新しいサービスリビジョンにルーティングするトラフィックの割合。これにより、本番稼働トラフィックの制御されたサブセットによるテストが可能になります。
- Canary ベイク時間 – Canary フェーズ中に残りのトラフィックを新しいサービスリビジョンに移行するまで待機する時間。これにより、新しいバージョンをモニタリングして検証する時間を確保できます。

トラフィック管理

カナリアデプロイでは、ロードバランサーのターゲットグループを使用してトラフィックの分散を管理します。

- 元のターゲットグループ – 現在の安定バージョンのタスクが含まれ、トラフィックの大部分を受信します。
- Canary ターゲットグループ – 新しいバージョンのタスクが含まれ、テスト用のトラフィックのごく一部を受信します。
- 加重ルーティング – ロードバランサーは加重ルーティングルールを使用して、設定された Canary の割合に基づいてターゲットグループ間でトラフィックを分散します。

モニタリングと評価

効果的なカナリアデプロイは、包括的なモニタリングに基づきます。

- ヘルスチェック – 両方のタスクセットは、トラフィックを受信する前にヘルスチェックに合格する必要があります。
- メトリクスの比較 – 応答時間、エラー率、スループットなど、元のバージョンと Canary バージョン間で主要なパフォーマンス指標を比較します。
- 自動ロールバック – Canary バージョンのパフォーマンスが低下した場合に自動的にロールバックをトリガーするように CloudWatch アラームを設定します。
- 手動検証 – 評価期間を使用して、続行する前にログ、メトリクス、ユーザーフィードバックを手動で確認します。

カナリアデプロイのベストプラクティス

サービスを使用したカナリアデプロイを成功させるには、以下のベストプラクティスに従ってください。

適切なトラフィックの割合を選択する

Canary トラフィックの割合を選択するときは、次の要因を考慮してください。

- 小規模から開始する – 問題が発生した場合の影響を最小限に抑えるために、トラフィックの 5～10% から開始します。
- アプリケーションの重要度を検討する – ミッションクリティカルなアプリケーションにはより小さな割合を使用し、重要度の低いサービスにはより大きな割合を使用します。

- トラフィック量を考慮する – Canary の割合が有意な検証のために十分なトラフィックを生成していることを確認します。

適切な評価期間を設定する

以下の考慮事項に基づいて評価期間を設定します。

- 十分な時間 – 有意なパフォーマンスデータをキャプチャするのに十分な評価期間を設定します。通常は 10 ~ 30 分です。
- トラフィックパターンを検討する – アプリケーションのトラフィックパターンとピーク使用時間を考慮します。
- 速度と安全性のバランス – 評価期間が長いほどデータが増加しますが、デプロイ速度は低下します。

包括的なモニタリングを実装する

カナリアデプロイのパフォーマンスを追跡するようにモニタリングを設定します。

- 主要メトリクス – 両方のタスクセットの応答時間、エラー率、スループット、リソース使用率をモニタリングします。
- アラームベースのロールバック – メトリクスがしきい値を超えたときにロールバックを自動的にトリガーするように CloudWatch アラームを設定します。
- 比較分析 – ダッシュボードを設定して、元のバージョンと Canary バージョン間のメトリクスを並べて比較します。
- ビジネスメトリクス – 変換率やユーザーエンゲージメントなどのビジネス固有のメトリクスを技術メトリクスとともに設定します。

ロールバック戦略を計画する

以下の戦略を使用して、潜在的なロールバックシナリオに備えます。

- 自動ロールバック – ヘルスチェックとパフォーマンスメトリクスに基づいて自動ロールバックトリガーを設定します。
- 手動ロールバック手順 – 自動トリガーがすべての問題をキャプチャしない場合の手動ロールバックの明確な手順を文書化します。

- ロールバックのテスト実施 – ロールバック手順を定期的にテストして、必要に応じて正しく動作することを確認します。

デプロイ前に徹底的に検証する

カナリアデプロイを続行する前に、徹底的な検証を実施するようにしてください。

- デプロイ前のテスト実施 – カナリアデプロイ前にステージング環境の変更を徹底的にテストします。
- ヘルスチェック設定 – ヘルスチェックがアプリケーションの準備状況と機能を正確に反映していることを確認します。
- 依存関係の検証 – 新しいバージョンにダウンストリームおよびアップストリームサービスとの互換性があることを確認します。
- データ整合性 – データベーススキーマの変更とデータ移行に下位互換性があることを確認します。

チームの関与を調整する

カナリアデプロイ中に効率的なチームの調整を実施します。

- デプロイウィンドウ – チームがモニタリングして対応できる営業時間内にカナリアデプロイをスケジュールします。
- コミュニケーションチャネル – デプロイステータスと問題のエスカレーションに関する明確なコミュニケーションチャネルを確立します。
- ロールの割り当て – モニタリング、意思決定、ロールバック実行の役割と責任を定義します。

Amazon ECS カナリアデプロイに必要なリソース

マネージドトラフィック移行が実施されるカナリアデプロイを使用するには、サービスが次のいずれかの機能を使用する必要があります。

- Elastic Load Balancing
- Service Connect

次のリストは、Amazon ECS カナリアデプロイに設定する必要がある内容について概要を示しています。

- サービスは Application Load Balancer、Network Load Balancer、Service Connect を使用します。適切なリソースを設定します。
  - Application Load Balancer – 詳細については、「ブルー/グリーンデプロイ、リニアデプロイおよびカナリアデプロイ用の Application Load Balancer リソース」を参照してください。
  - Network Load Balancer – 詳細については、「Amazon ECS ブルー/グリーン、リニアおよびカナリアデプロイの Network Load Balancer リソース」を参照してください。
  - Service Connect – 詳細については、「Amazon ECS ブルー/グリーンデプロイ、リニアデプロイおよびカナリアデプロイ用の Service Connect リソース」を参照してください。
- サービスのデプロイコントローラーを ECS に設定します。
- サービス定義でデプロイ戦略を `canary` として設定します。
- 必要に応じて、次のように追加のパラメータを設定します。
  - 新しいデプロイのベイク時間
  - Canary フェーズ中に新しいサービスリビジョンにルーティングするトラフィックの割合。
  - Canary フェーズ中に残りのトラフィックを新しいサービスリビジョンに移行するまで待機する期間。
  - 自動ロールバックの CloudWatch アラーム
  - デプロイのライフサイクルフック (指定されたデプロイステージで実行される Lambda 関数または一時停止フック)

ベストプラクティス

Amazon ECS カナリアデプロイを正常に処理するには、次のベストプラクティスに従ってください。

- アプリケーションが同時に実行されている両方のサービスリビジョンを処理できることを確認します。
- デプロイ中に両方のサービスリビジョンを処理するため、十分なクラスターキャパシティの確保を計画します。
- ロールバック手順は、本番環境での実装前にテストします。
- アプリケーションの正常性を正確に反映する、適切なヘルスチェックを設定します。
- グリーンデプロイの十分なテストを可能にするベイク時間を設定します。
- CloudWatch アラームを実装し、問題を自動的に検出してロールバックをトリガーします。
- ライフサイクルフックを使用し、各デプロイステージで自動テストを実行します。

- 小さな Canary の割合 (5 ~ 10%) から始めて、問題が発生した場合の影響を最小限に抑えます。
- 有意義なパフォーマンスデータ収集のための十分な時間を確保できる適切な評価期間を設定します。
- 自動ロールバックトリガーの CloudWatch アラームを使用して包括的なモニタリングを実装します。
- アプリケーションの準備状況と機能を正確に反映するヘルスチェックを設定します。
- 評価中に、技術的メトリクス (応答時間、エラー率) とビジネスメトリクスの両方をモニタリングします。
- アプリケーションがセッションや状態に関する問題が発生しない状態でトラフィックの分割を処理できることを確認します。
- ロールバック手順を計画し、定期的にテストして、必要に応じて機能することを確認します。
- チームがモニタリングして対応できる営業時間内にカナリアデプロイをスケジュール設定します。
- カナリアデプロイを実施する前に、ステージング環境で変更を詳細に検証します。
- 手動介入とロールバックの決定に関する明確な手順を文書化します。

Amazon ECS カナリアデプロイの作成

Amazon ECS カナリアデプロイを使用すると、トラフィックのごく一部を新しいサービスリビジョン (「Canary」) に移行できます。デプロイを検証し、指定した間隔で残りのトラフィックをすべて一度に移行します。このアプローチでは、フルデプロイの前に最小限のリスクで新機能をテストできます。

前提条件

カナリアデプロイを開始する前に、次の操作を実行します。

1. 適切なアクセス許可を設定します。
   - Elastic Load Balancing のアクセス許可の詳細については、「ロードバランサー用の Amazon ECS インフラストラクチャの IAM ロール」を参照してください。
   - Lambda のアクセス許可の詳細については、「Amazon ECS ブルー/グリーンデプロイの Lambda 関数に必要となるアクセス許可」を参照してください。
2. Amazon ECS カナリアデプロイでは、サービスが次のいずれかの機能を使用する必要があります。適切なリソースを設定してください。
   - Application Load Balancer – 詳細については、「ブルー/グリーンデプロイ、リニアデプロイおよびカナリアデプロイ用の Application Load Balancer リソース」を参照してください。

- Network Load Balancer – 詳細については、「Amazon ECS ブルー/グリーン、リニアおよびカナリアデプロイの Network Load Balancer リソース」を参照してください。
- Service Connect – 詳細については、「Amazon ECS ブルー/グリーンデプロイ、リニアデプロイおよびカナリアデプロイ用の Service Connect リソース」を参照してください。

手順

コンソールまたは AWS CLI を使用して、Amazon ECS カナリアデプロイサービスを作成できます。

Console

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. サービスを起動するリソースを決定します。

サービスの起動元	Steps
クラスター	a. [Clusters] ページで、サービスを作成するクラスターを選択します。 クラスター詳細のページが表示されます。 b. [Services (サービス)] タブで、[Create (作成)] を選択します。
タスク定義	a. [タスク定義] ページで、タスク定義を選択します。 b. [デプロイ] メニューで、[サービスの作成] を選択します。

[サービスの作成] ページが表示されます。

3. [サービスの詳細] で、次の操作を行います。

a. [タスク定義ファミリー] では、使用するタスク定義を選択します。次に [タスク定義リビジョン] には、使用するリビジョンを入力します。

b. [Service name] (サービス名) でサービスの名前を入力します。

4. 既存のクラスターでサービスを実行するには、[既存のクラスター] にクラスターを選択します。新しいクラスターでサービスを実行するには、[クラスターの作成] を選択します。

5. クラスターのインフラストラクチャ全体にタスクを分散する方法を選択します。[コンピュート設定] で、オプションを選択します。

コンピューティングオプション	Steps	
キャパシティープロバイダー戦略	a. [コンピューティングオプション] で、[キャパシティプロバイダー戦略] を選択します。 b. 戦略を選択: • デフォルトのキャパシティープロバイダー戦略を使用するには、[Use cluster default] (デフォルトのクラスターを使用) を選択します。 • クラスターにデフォルトのキャパシティプロバイダー戦略がない場合、またはカスタム戦略を使用する場合は、[カスタムを使用]、[キャパシティプロバイダー戦略を追加] を選択し、[ベース]、[キャパシティプロバイダー]、[重み] を	

コンピューティングオプション	Steps	
	指定して、カスタムキャパシティプロバイダー戦略を定義します。 **Note** 戦略でキャパシティープロバイダーを使用するには、キャパシティープロバイダーをクラスターに関連付ける必要があります。	

コンピューティングオプション	Steps	
起動タイプ	a. [Compute options] (コンピューティングオプション) セクションで、[Launch type] (起動タイプ) を選択します。 b. [Launch type] (起動タイプ) で、起動タイプを選択します。 c. (オプション) Fargate が指定されている場合は、[プラットフォームバージョン] で、使用するプラットフォームのバージョンを指定します。プラットフォームバージョンが指定されない場合、LATEST プラットフォームバージョンが使用されます。	

6. [デプロイ設定] で、次の操作を行います。

   a. [サービスタイプ] では、[レプリカ] を選択します。

   b. [Desired tasks] (必要なタスク) で、サービス内で起動および維持するタスクの数を入力します。

   c. Amazon ECS がアベイラビリティーゾーン間のタスクの分散をモニタリングし、不均衡が発生したときにタスクを再分散するには、[アベイラビリティーゾーンのサービス再調整] で [アベイラビリティーゾーンのサービス再調整] を選択します。

   d. [ヘルスチェックの猶予期間] には、タスクの初回起動後にサービススケジューラが異常な Elastic Load Balancing、VPC Lattice、コンテナヘルスチェックを無視する時間 (秒単位) を入力します。ヘルスチェックの猶予期間値を指定しない場合、デフォルト値 0 が使用されます。

7. [デプロイ設定] で、カナリアデプロイ設定を構成します。

a. [デプロイ戦略] で、[Canary] を選択します。

b. [Canary の割合] には、最初のステージでグリーンサービスリビジョンに移行するトラフィックの割合を入力します (例えば、最初の Canary トラフィックの場合は 10%)。

c. [Canary ベイク時間] には、残りのトラフィックをグリーンサービスリビジョンに移行するまでに待機する時間を分単位で入力します。

d. [ベイク時間] には、ブルーリビジョンが終了する前に、トラフィックの最終移行後にブルーおよびグリーンのサービスリビジョンの両方が同時に実行される時間 (分) を入力します。

e. (オプション) デプロイの特定段階で実行されるように、ライフサイクルフックを設定します。[デプロイライフサイクルフック] で、[追加] を選択します。

Lambda フックの場合:

- [ターゲットタイプ] で [Lambda] を選択します。
- [Lambda 関数] には、ARN の関数名を入力します。
- [ロール] では、Lambda 関数を呼び出すアクセス許可を持つ IAM ロールを選択します。
- [ライフサイクルステージ] では、Lambda 関数を実行するステージを選択します。

一時停止フックの場合:

- [ターゲットタイプ] で、[一時停止] を選択します。
- [ライフサイクルステージ] で、デプロイを一時停止するステージを選択します。
- (オプション) [タイムアウト] の場合は、タイムアウト期間とアクションを設定します。

8. Amazon ECS がデプロイの障害を検出して処理する方法を設定するには、[Deployment failure detection] (デプロイ障害検出) を展開し、オプションを選択します。

a. タスクを開始できない場合にデプロイを停止するには、[Use the Amazon ECS deployment circuit breaker] (Amazon ECS デプロイサーキットブレーカーを使用する) を選択します。

デプロイサーキットブレーカーによってデプロイが失敗状態に設定されたときに、ソフトウェアがデプロイを最後に完了したデプロイ状態に自動的にロールバックするようにするには、[失敗時のロールバック] を選択します。

b. アプリケーションメトリクスに基づいてデプロイを停止するには、[CloudWatch アラームを使用する] を選択します。次に、[CloudWatch アラーム名] からアラームを選択します。新しいアラームを作成するには、CloudWatch コンソールに移動します。

CloudWatch アラームによってデプロイが失敗状態に設定されたときに、ソフトウェアがデプロイを最後に完了したデプロイ状態に自動的にロールバックするようにするには、[失敗時のロールバック] を選択します。

9. (オプション) Service Connect を使用してサービスを相互接続するには、[Service Connect] を展開して以下を指定します。

a. [Service Connect をオンにする] を選択します。

b. [Service Connect configuration] (Service Connect 設定) で、クライアントモードを指定します。

- サービスが名前空間内の他のサービスへの接続のみを必要とするネットワーククライアントアプリケーションを実行している場合は、[クライアント側のみ] を選択します。
- サービスがネットワークまたは Web サービスアプリケーションを実行していて、このサービスにエンドポイントを提供し、名前空間内の他のサービスに接続する必要がある場合は、[Client and server] (クライアントとサーバー) を選択します。

c. デフォルトのクラスター名前空間ではない名前空間を使用するには、[Namespace] (名前空間) でサービス名前空間を選択します。この場合、AWS アカウントの同じ AWS リージョンで個別に作成された名前空間、またはAWS Resource Access Manager (AWS RAM) を使用してアカウントと共有されている同じリージョンの名前空間のいずれかを選択できます。共有 AWS Cloud Map 名前空間の詳細については、「AWS Cloud Map デベロッパーガイド」の「クロスアカウント AWS Cloud Map 名前空間の共有」を参照してください。

d. (オプション) カナリアデプロイのテストトラフィックヘッダーのルールを設定します。[テストトラフィックのルーティング] では、以下を指定します。

i. [テストトラフィックヘッダーのルールの有効化] を選択し、テスト中に特定のリクエストをグリーンサービスリビジョンにルーティングします。

ii. [ヘッダー一致ルール] では、テストトラフィックをルーティングする条件を設定します。

- [ヘッダー名]: 一致する HTTP ヘッダーの名前を入力します (例えば、`X-Test-Version` または `User-Agent`)。

- [一致タイプ]: 一致する条件を選択します。
  - [完全一致]: ヘッダー値が指定された値と完全に一致するリクエストをルーティングする
  - [ヘッダーが存在する]: 値に関係なく、指定されたヘッダーを含むリクエストをルーティングする
  - [パターン一致]: ヘッダー値が指定されたパターンと一致するリクエストをルーティングする
- [ヘッダー値] (完全一致またはパターン一致を使用する場合): 照合する値またはパターンを入力します。

複数のヘッダー一致ルールを追加し、複雑なルーティングロジックを作成できます。設定されたルールに一致するリクエストは、テストのためにグリーンサービスリビジョンにルーティングされます。

iii. [ヘッダールールの追加] を選択し、追加のヘッダー一致条件を設定します。

> **Note**
>
> テストトラフィックヘッダーのルールにより、完全なデプロイを完了する前に、制御されたトラフィックで新機能を検証できます。これにより、ブルーサービスリビジョンへの通常のトラフィックフローを維持しながら、特定のリクエスト (内部テストツールやベータユーザーなどによるもの) でグリーンサービスリビジョンをテストできます。

e. (オプション) ログ設定を指定します。[ログコレクションを使用] を選択します。デフォルトのオプションでは、コンテナログを CloudWatch Logs に送信します。その他のログドライバオプションは、AWS FireLens を使用して構成されます。詳細については、「Amazon ECS ログを AWS サービスまたは AWS Partner に送信する」を参照してください。

以下では、各コンテナログの送信先について詳しく説明します。

- Amazon CloudWatch — コンテナログを CloudWatch Logs に送信するようにタスクを設定します。デフォルトのログドライバーオプションが提供され、ユーザーに代わり CloudWatch ロググループを作成します。別のロググループ名を指定するには、ドライバーオプションの値を変更します。

- [Amazon Data Firehose] — Firehose にコンテナログを送信するようタスクを設定します。Firehose 配信ストリームにログを送信するデフォルトのログドライバーオプションが提供されています。別の配信ストリーム名を指定するには、ドライバーオプションの値を変更します。
- Amazon Kinesis Data Streams — Kinesis Data Streams にコンテナログを送信するようタスクを設定します。Kinesis Data Streams のストリームにログを送信するデフォルトのログドライバーオプションが提供されています。別のストリーム名を指定するには、ドライバーオプションの値を変更します。
- Amazon OpenSearch Service — コンテナログを OpenSearch Service ドメインに送信するようタスクを設定します。ログドライバーオプションを提供する必要があります。
- Amazon S3 — Amazon S3 バケットにコンテナログを送信するようタスクを設定します。デフォルトのログドライバーオプションはデフォルトで設定されていますが、有効な Amazon S3 バケット名を指定する必要があります。

10. (オプション) カナリアデプロイ用に [ロードバランシング] を設定します。

Elastic Load Balancing のタイプ	Steps	
Application Load Balancer	a. [ロードバランサーのタイプ] では [Application Load Balancer] を選択します。 b. 選択新しいロードバランサーを作成するをクリックして新しいApplication Load Balancer を作成するか、既存のロードバランサーを使用する既存のApplication Load Balancer を選択します。 c. [コンテナ] では、サービスをホストするコンテナを選択します。	

Elastic Load Balancing のタイプ	Steps	
	d. [Load balancer name] (ロードバランサー名) では、一意な名前を入力します。 e. [Listener] (リスナー) で、Application Load Balancer が接続リクエストをリッスンするポートとプロトコルを入力します。デフォルトでは、ロードバランサーはポート 80 と HTTP を使用するように設定されます。 • [本番ルール] には、ルールの [評価順序] および [パスパターン] を入力します。 このルールは、本番 (ブルー) サービスリビジョンのトラフィック用です。 • [テストルール] には、ルールの [評価順序] および [パスパターン] を入力します。 このルールは、テスト (グリーン) サービスリビジョンのトラフィック用です。 f. [ターゲットグループ] では、以下を設定します。	

Elastic Load Balancing のタイプ	Steps	
	• [Target group name] (ターゲットグループ名) では、Application Load Balancer がリクエストをルーティングするターゲットグループの名前とプロトコルを入力します。 • [プロトコル] では、Application Load Balancer がリクエストをルーティングするターゲットグループのプロトコルを選択します。デフォルトでは、ターゲットグループは、タスク定義で定義されている最初のコンテナにリクエストをルーティングします。 • [登録解除の遅延] には、ロードバランサーがターゲットの状態を UNUSED に変更するまでの秒数を入力します。デフォルトは 300 秒です。 • [Health check path] (ヘルスチェックパス) では、Application Load Balancer とコンテナ間の接続の正常性を検証するために、Appli	

Elastic Load Balancing のタイプ	Steps	
	cation Load Balancer が定期的にリクエストを送信するコンテナ内に存在するパスを入力します。デフォルトはルートディレクトリ (/)。 • [代替グループ名] には、テストサービスリビジョンのターゲットグループのグループ名を入力します。	

Elastic Load Balancing のタイプ	Steps	
Network Load Balancer	a. [Load balancer type] (ロードバランサータイプ) では、[Network Load Balancer] を選択します。 b. [Load Balancer] (ロードバランサー) では、既存の Network Load Balancer を選択します。 c. [Choose container to load balance] (負荷分散するコンテナを選択) で、サービスをホストするコンテナを選択します。 d. [本番リスナー] では、[本番リスナーポート] および [本番リスナープロトコル] を選択します。 これは、本番サービスリビジョントのラフィックのリスナーです。 e. [テストリスナー] では、[テストリスナーポート] および [テストリスナープロトコル] を選択します。 これは、テストサービスリビジョンのトラフィックのリスナーです。	

Elastic Load Balancing のタイプ	Steps	
	f. [ターゲットグループ] では、以下を設定します。 • [Target group name] (ターゲットグループ名) では、Network Load Balancer がリクエストをルーティングするターゲットグループの名前とプロトコルを入力します。 • [プロトコル] では、Network Load Balancer がリクエストをルーティングするターゲットグループのプロトコルを選択します。デフォルトでは、ターゲットグループは、タスク定義で定義されている最初のコンテナにリクエストをルーティングします。 • [登録解除の遅延] には、ロードバランサーがターゲットの状態を UNUSED に変更するまでの秒数を入力します。デフォルトは 300 秒です。 • [Health check path] (ヘルスチェックパス) では、Application Load Balancer とコンテナ	

Elastic Load Balancing のタイプ	Steps	
	間の接続の正常性を検証するために、Application Load Balancer が定期的にリクエストを送信するコンテナ内に存在するパスを入力します。デフォルトはルートディレクトリ (/)。 • [代替グループ名] には、テスト (緑) サービスリビジョンのターゲットグループのグループ名を入力します。	

11. （オプション）サービスとタスクを識別しやすくするには、[Tags] (タグ) セクションを展開し、タグを設定します。

    新しく起動したすべてのタスクに対して、Amazon ECS がクラスター名とタスク定義タグで自動的にタグ付けするようにするには、[Amazon ECS マネージドタグを有効にする] を選択し、[タグの伝播元] で [タスク定義] を選択します。

    新しく起動したすべてのタスクに対して、Amazon ECS がクラスター名とサービスタグで自動的にタグ付けするようにするには、[Amazon ECS マネージドタグを有効にする] を選択し、[タグの伝播元] で [サービス] を選択します。

    タグを追加または削除します。

    - [タグを追加] [Add tag] (タグを追加) を選択し、以下を実行します。
      - [キー] にはキー名を入力します。
      - [値] にキー値を入力します。
    - [タグの削除] タグの横にある [タグの削除] を選択します。
12. [作成] を選択します。

AWS CLI

1. canary-service-definition.json という名前のファイルを作成し、次の内容を記述します。

   user-input を独自の値に置き換えます。

```
{
  "serviceName": "myCanaryService",
  "cluster": "arn:aws:ecs:us-west-2:123456789012:cluster/sample-fargate-
cluster",
  "taskDefinition": "sample-fargate:1",
  "desiredCount": 5,
  "launchType": "FARGATE",
  "networkConfiguration": {
    "awsvpcConfiguration": {
      "subnets": [
        "subnet-09ce6e74c116a2299",
        "subnet-00bb3bd7a73526788",
        "subnet-0048a611aaec65477"
      ],
      "securityGroups": [
        "sg-09d45005497daa123"
      ],
      "assignPublicIp": "ENABLED"
    }
  },
  "deploymentController": {
    "type": "ECS"
  },
  "deploymentConfiguration": {
    "strategy": "CANARY",
    "maximumPercent": 200,
    "minimumHealthyPercent": 100,
    "canaryConfiguration" : {
        "canaryPercent" : 5.0,
        "canaryBakeTime" : 10
    },
    "bakeTimeInMinutes": 10,
    "alarms": {
      "alarmNames": [
        "myAlarm"
      ],
```

```
      "rollback": true,
      "enable": true
    }
  },
  "loadBalancers": [
    {
      "targetGroupArn": "arn:aws:elasticloadbalancing:us-
west-2:123456789012:targetgroup/blue-target-group/54402ff563af1197",
      "containerName": "fargate-app",
      "containerPort": 80,
      "advancedConfiguration": {
        "alternateTargetGroupArn": "arn:aws:elasticloadbalancing:us-
west-2:123456789012:targetgroup/green-target-group/cad10a56f5843199",
        "productionListenerRule": "arn:aws:elasticloadbalancing:us-
west-2:123456789012:listener-rule/app/my-canary-
demo/32e0e4f946c3c05b/9cfa8c482e204f7d/831dbaf72edb911",
        "roleArn": "arn:aws:iam::123456789012:role/LoadBalancerManagementforECS"
      }
    }
  ]
}
```

2. `create-service` を実行します。

```
aws ecs create-service --cli-input-json file://canary-service-definition.json
```

次のステップ

カナリアデプロイを設定したら、以下の手順を実行します。

- サービスを更新してデプロイを開始します。詳細については、「Amazon ECS サービスを更新する」を参照してください。
- デプロイプロセスをモニタリングし、Canary のパターンに従っていることを確認します。
  - グリーンサービスリビジョンが作成され、スケールアップされる
  - トラフィック (Canary) のごく一部がグリーンリビジョンに移行されます
  - システムは Canary に対して指定された間隔の時間待機します
  - 残りのトラフィックはすべて一度にグリーンリビジョンに移行されます
  - ベイク時間が過ぎたら、ブルーリビジョンが終了する

デプロイに関する用語

Amazon ECS デプロイドキュメント全般で、以下の用語が使用されています。

ブルー/グリーンデプロイ

既存の環境 (ブルー) とともに新しい環境 (グリーン) を作成し、検証後にトラフィックをブルーからグリーンに切り替えるデプロイ戦略。

Canary デプロイ

トラフィックのごく一部を新しいバージョンにルーティングしながら、大部分のトラフィックを検証するために安定したバージョンに保持するデプロイ戦略。

リニアデプロイ

時間の経過とともにトラフィックを古いバージョンから新しいバージョンに同じ割合ずつ段階的に移行するデプロイ戦略。

ローリングデプロイ

古いバージョンのインスタンスを新しいバージョンのインスタンスに一度に 1 つずつ置き換えるデプロイ戦略。

タスクセット

デプロイ中にサービス内で同じタスク定義を実行するタスクのコレクション。

ターゲットグループ

デプロイ中にロードバランサーからトラフィックを受信するターゲットの論理グループ。

デプロイメントコントローラー

Amazon ECS、CodeDeploy、外部コントローラーなど、サービスの新しいバージョンをデプロイするために使用される方法。

ロールバック

デプロイ中に問題が検出されたときに、アプリケーションの以前のバージョンに戻すプロセス。

サードパーティーのコントローラーを使用して Amazon ECS サービスをデプロイする

この外部デプロイタイプでは、Amazon ECS サービスのデプロイプロセスを完全に制御するために、すべてのサードパーティーのデプロイコントローラーを使用できます。サービスの詳細はサービス管理 API アクション

(CreateService、UpdateService、DeleteService) または タスク設定管理 API アクション (CreateTaskSet、UpdateTaskSet、UpdateServicePrimaryTaskSet、DeleteTaskSet) のいずれかにより管理されます。各 API アクションは、サービス定義パラメータのサブセットを管理します。

UpdateService API アクションは、サービスの必要数およびヘルスチェック猶予期間パラメータを更新します。コンピューティングオプション、プラットフォームバージョン、ロードバランサーの詳細、ネットワーク構成、またはタスク定義を更新する必要がある場合、新しいタスクセットを作成する必要があります。

UpdateTaskSet API アクションは、タスクセットのスケールパラメータのみを更新します。

UpdateServicePrimaryTaskSet API アクションは、サービス内のどのタスクセットをプライマリタスクセットであるかを変更します。DescribeServices API アクションを呼び出すと、プライマリタスクセットに指定されたすべてのフィールドが返されます。サービスのプライマリタスクセットが更新されると、サービスの新しいプライマリタスクセットに存在し、古いプライマリタスクセットとは異なるタスクセットのパラメータ値はどれでも、新しいプライマリタスクセットが定義されるときに新しい値に更新されます。サービスにプライマリタスクセットが定義されていない場合にサービスを定義するとき、タスクセットフィールドは null になります。

外部デプロイに関する考慮事項

外部デプロイタイプを使用するときは、以下の点を考慮します。

- サポートされているロードバランサーのタイプは、Application Load Balancer または Network Load Balancer です。
- Fargate または EXTERNAL のデプロイコントローラータイプは、DAEMON スケジューリング戦略をサポートしません。
- Application AutoScaling と Amazon ECS を統合した場合、Amazon ECS サービスのみをターゲットとしてサポートします。Amazon ECS でサポートされているスケーラブルなディメンションは ecs:service:DesiredCount であり、これは Amazon ECS サービスのタスク数です。Application AutoScaling および Amazon ECS タスクセットの間には直接的な統合はありません。Amazon ECS タスクセットにより、Amazon ECS サービスの DesiredCount に基づいて ComputedDesiredCount が計算されます。

外部デプロイワークフロー

以下に示しているのは、Amazon ECS で外部デプロイを管理する基本的なワークフローです。

外部デプロイコントローラーを使用して Amazon ECS サービスを管理するには

1. Amazon ECS サービスを作成する 必須のパラメータは、サービス名のみです。サービスを作成するときに、外部デプロイコントローラーを使用して以下のパラメータを指定できます。その他すべてのサービスパラメータは、サービス内でタスクセットが作成されるときに指定されます。

   `serviceName`

   タイプ: 文字列

   必須: はい

   サービスの名前。最大 255 文字の英字 (大文字と小文字)、数字、ハイフン、アンダースコアを使用できます。サービス名は同じクラスター内で一意になるようにしてください。ただし、リージョン内の複数のクラスター間や複数のリージョンにまたがるクラスター間では、同様の名前のサービスがあっても構いません。

   `desiredCount`

   指定したタスクセットのタスク定義のインスタンスをサービス内で実行状態に保つ数を設定します。

   `deploymentConfiguration`

   デプロイ時に実行されるタスクの数と、タスクの停止および開始の順序を制御するオプションのデプロイパラメータ。

   `tags`

   タイプ: オブジェクトの配列

   必須: いいえ

   サービスに適用し、サービスの分類と整理に役立つメタデータ。各タグはキーとオプションの値で構成され、どちらもお客様側が定義します。サービスが削除されると、タグも削除されます。サービスには最大 50 個のタグを適用できます。詳細については、「Amazon ECS リソースにタグ付けする」を参照してください。

   サービスを更新しても、このパラメータは新しいサービスのデプロイをトリガーしません。

   `key`

   タイプ: 文字列

長さの制限: 最小長は 1 です。最大長は 128 です。

必須: いいえ

タグを構成するキーと値のペアの一部。キーは、より具体的なタグ値のカテゴリのように動作する、一般的なラベルです。

`value`

タイプ: 文字列

長さの制約: 最小長は 0 です。最大長は 256 です。

必須: いいえ

タグを構成するキーと値のペアのオプションの一部。値はタグカテゴリ (キー) の記述子として機能します。

`enableECSManagedTags`

サービス内のタスクに Amazon ECS マネージドタグを使用するか否かを指定します。詳細については、「請求にタグを使用する」を参照してください。

`propagateTags`

タイプ: 文字列

有効な値: `TASK_DEFINITION` | `SERVICE`

必須: いいえ

タグをタスク定義またはサービスからサービスのタスクへコピーするかどうかを指定します。値を指定しない場合、タグはコピーされません。タグは、サービス作成中のサービス内のタスクにのみコピーすることができます。タグをサービス作成後またはタスク作成後のタスクに追加するには、`TagResource` API アクションを使用します。

サービスを更新しても、このパラメータは新しいサービスのデプロイをトリガーしません。

`schedulingStrategy`

使用するスケジュール戦略。外部デプロイコントローラーを使用するサービスは、`REPLICA` スケジューリング戦略のみをサポートします。

`placementConstraints`

サービスのタスクに使用する、配置制約オブジェクトの配列。タスクごとに最大 10 個の制約を指定できます (この制限数には、タスク定義内の制約と、実行時に指定される制約が含まれます)。Fargate を使用している場合、タスク配置の制約事項はサポートされません。

`placementStrategy`

サービスのタスクで使用する配置戦略オブジェクト。サービスごとに最大 4 つの戦略ルールを指定できます。

次の例は、外部デプロイコントローラーを使用するサービスを作成するためのサービス定義です。

```
{
    "cluster": "",
    "serviceName": "",
    "desiredCount": 0,
    "role": "",
    "deploymentConfiguration": {
        "maximumPercent": 0,
        "minimumHealthyPercent": 0
    },
    "placementConstraints": [
        {
            "type": "distinctInstance",
            "expression": ""
        }
    ],
    "placementStrategy": [
        {
            "type": "binpack",
            "field": ""
        }
    ],
    "schedulingStrategy": "REPLICA",
    "deploymentController": {
        "type": "EXTERNAL"
    },
    "tags": [
        {
            "key": "",
```

```
            "value": ""
        }
    ],
    "enableECSManagedTags": true,
    "propagateTags": "TASK_DEFINITION"
}
```

2. 最初のタスクセットを作成します。タスクセットには、サービスに関する次の詳細が含まれています。

   `taskDefinition`

   使用するタスクセットのタスクのタスク定義。

   `launchType`

   タイプ: 文字列

   有効な値: EC2 | FARGATE | EXTERNAL

   必須: いいえ

   サービスを実行する起動タイプ。起動タイプを指定しない場合は、デフォルトで `capacityProviderStrategy` が使用されます。

   サービスを更新すると、このパラメータは新しいサービスのデプロイをトリガーします。

   `launchType` を指定した場合、`capacityProviderStrategy` パラメータを省略する必要があります。

   `platformVersion`

   タイプ: 文字列

   必須: いいえ

   サービス内のタスクが実行されているプラットフォームのバージョン。プラットフォームのバージョンは、Fargate 起動タイプを使用するタスクに対してのみ指定されています。指定されない場合、デフォルトで最新バージョン (LATEST) が使用されます。

   サービスを更新すると、このパラメータは新しいサービスのデプロイをトリガーします。

   AWS Fargate プラットフォームのバージョンを使って、Fargate タスクインフラストラクチャの特定のランタイム環境を参照できます。プラットフォームのバージョンに LATEST を

指定してタスクを実行またはサービスを作成すると、プラットフォームの最新バージョンをタスクで利用できるようになります。サービスをスケールアップする場合は、これらのタスクには、サービスの最新のデプロイで指定されたプラットフォームのバージョンが提供されます。詳細については、「Amazon ECS 向け Fargate プラットフォームバージョン」を参照してください。

> **Note**
>
> プラットフォームのバージョンは、EC2 起動タイプを使用するタスクには指定されません。

`loadBalancers`

サービスで使用するロードバランサーを表すロードバランサーオブジェクト。外部デプロイコントローラーを使用する場合、Application Load Balancer と Network Load Balancer のみがサポートされます。Application Load Balancer を使用している場合、タスクセットあたり 1 つの Application Load Balancer ターゲットグループのみが許可されます。

次のスニペットは、使用する `loadBalancer` オブジェクトの例を示しています。

```
"loadBalancers": [
        {
            "targetGroupArn": "",
            "containerName": "",
            "containerPort": 0
        }
]
```

> **Note**
>
> `loadBalancer` オブジェクトを指定する場合は、`targetGroupArn` を指定し、`loadBalancerName` パラメータを省略する必要があります。

`networkConfiguration`

サービスのネットワーク設定。このパラメータは `awsvpc` ネットワークモードを使用して独自の Elastic Network Interface を受け取るタスク定義の場合に必要です。その他のネット

ワークモードではサポートされていません。Fargate のネットワーク設定の詳細については、「Fargate における Amazon ECS タスクのネットワークオプション」を参照してください。

`serviceRegistries`

このサービスに割り当てるサービス検出レジストリの詳細。詳細については、「サービス検出を使用して Amazon ECS サービスを DNS 名で接続する」を参照してください。

`scale`

タスクセットに配置して実行し続けるために必要なタスク数の浮動小数点パーセンテージ。この値は、サービスの `desiredCount` の割合 (%) の合計として指定されます。許容値は 0 から 100 までの数字です。

外部デプロイコントローラー用のタスクセットを作成するための JSON の例を次に示します。

```
{
    "service": "",
    "cluster": "",
    "externalId": "",
    "taskDefinition": "",
    "networkConfiguration": {
        "awsvpcConfiguration": {
            "subnets": [
                ""
            ],
            "securityGroups": [
                ""
            ],
            "assignPublicIp": "DISABLED"
        }
    },
    "loadBalancers": [
        {
            "targetGroupArn": "",
            "containerName": "",
            "containerPort": 0
        }
    ],
    "serviceRegistries": [
        {
            "registryArn": "",
```

```
            "port": 0,
            "containerName": "",
            "containerPort": 0
        }
    ],
    "launchType": "EC2",
    "capacityProviderStrategy": [
        {
            "capacityProvider": "",
            "weight": 0,
            "base": 0
        }
    ],
    "platformVersion": "",
    "scale": {
        "value": null,
        "unit": "PERCENT"
    },
    "clientToken": ""
}
```

3. サービスの変更が必要な場合、更新するパラメータに応じて UpdateService、UpdateTaskSet、または CreateTaskSet のいずれかの API アクションを使用します。タスクセットを作成した場合は、サービスの各タスクセットに scale パラメータを使用して、サービス内で継続して実行するタスクの数を決定します。例えば、tasksetA を含むサービスがあり tasksetB を作成した場合、本番トラフィックを移行する前に tasksetB の有効性をテストする場合があります。両方のタスクの scale を 100 に設定し、すべての本番トラフィックを tasksetB に移行する準備が整ったときに、scale の tasksetA を 0 に更新して、スケールダウンすることもできます。

Amazon ECS 用の CodeDeploy ブルー/グリーンデプロイ

Amazon ECS ブルー/グリーンデプロイを使用することをお勧めします。詳細については、「Amazon ECS ブルー/グリーンデプロイの作成」を参照してください。

ブルー/グリーンデプロイタイプでは、CodeDeploy によって制御される ブルー/グリーンデプロイモデルを使用します。このデプロイタイプは、本番稼働用トラフィックを送信する前にサービスの新しいデプロイを検証するために使用します。詳細については、「AWS CodeDeploy ユーザーガイド」の「What Is CodeDeploy」を参照してください。デプロイ前に Amazon ECS サービスの状態を検証します。

ブルー/グリーンデプロイ中にトラフィックを移行するには、次の 3 つの方法があります。

- Canary - トラフィックは 2 つの増分で移行されます。事前定義された複数の Canary オプションから選択し、最初の増分および間隔でトラフィックを更新済みタスクセットに移行する割合 (%) を分単位で指定できます。次に 2 回目の増分で残りのトラフィックを移行します。
- Linear - トラフィックは等しい増分で移行され、増分間の間隔 (分) も同じです。事前定義された複数の線形オプションから選択し、増分ごとに移行するトラフィックの割合 (%) と増分間の間隔 (分) を指定できます。
- 一括 - すべてのトラフィックを元のタスクセットから更新済みタスクセットに同時に移行します。

以下に示しているのは、サービスが Blue/Green デプロイタイプを使用するときに Amazon ECS が使用する CodeDeploy のコンポーネントです。

CodeDeploy アプリケーション

CodeDeploy リソースのコレクションです。これは 1 つ以上のデプロイグループで構成されます。

CodeDeploy デプロイグループ

デプロイ設定。これには以下の構成要素があります。

- Amazon ECS クラスターとサービス
- ロードバランサーのターゲットグループとリスナー情報
- デプロイメントロールバック戦略
- トラフィックルーティング設定
- 元のリビジョンの終了設定
- デプロイ設定
- デプロイを停止するために設定できる CloudWatch アラームの設定
- 通知用の SNS または CloudWatch Eventsの設定

詳細については、AWS CodeDeployユーザーガイドの「デプロイグループの操作」を参照してください。

CodeDeploy デプロイ設定

デプロイメント中に本番トラフィックを置換タスクに CodeDeploy がルーティングする方法を指定します。次の事前定義された線形および Canary デプロイ設定を使用できます。また、カ

スタム定義の線形および Canary デプロイを作成することもできます。詳細については、AWS CodeDeploy ユーザーガイドの「デプロイ設定の操作」を参照してください。

- [CodeDeployDefault.ECSAllAtOnce]: すべてのトラフィックを同時に更新済みの Amazon ECS コンテナに移行します。
- CodeDeployDefault.ECSLinear10PercentEvery1Minutes: すべてのトラフィックがシフトされるまで、1 分ごとにトラフィックの 10 パーセントをシフトします。
- CodeDeployDefault.ECSLinear10PercentEvery3Minutes: すべてのトラフィックがシフトされるまで、3 分ごとにトラフィックの 10 パーセントをシフトします。
- [CodeDeployDefault.ECSCanary10Percent5Minutes]: 最初の増分で 10 パーセントのトラフィックをシフトします。残りの 90 パーセントは 5 分後にデプロイされます。
- [CodeDeployDefault.ECSCanary10percent15Minutes]: 最初の増分で 10 パーセントのトラフィックをシフトします。残りの 90 パーセントは 15 分後にデプロイされます。

リビジョン

リビジョンは、CodeDeploy アプリケーション仕様ファイル (AppSpec ファイル) です。AppSpec ファイルでは、タスク定義の完全な ARN と置換タスクセットのコンテナとポートを指定し、新しいデプロイが作成時にトラフィックがルーティングされるようにします。コンテナ名は、タスク定義内で参照されているコンテナ名のいずれかに設定する必要があります。ネットワーク設定またはプラットフォームのバージョンがサービス定義で更新された場合は、AppSpec ファイルでその詳細についても指定する必要があります。また、デプロイメントライフサイクルイベント中に実行する Lambda 関数も指定できます。Lambda 関数を使用することで、デプロイメント中にテストを実行し、メトリクスを返すことができます。詳細については、AWS CodeDeployユーザーガイドで「AppSpec ファイルリファレンス」を参照してください。

考慮事項

ブルー/グリーンデプロイタイプを使用するときは、以下の点を考慮します。

- Blue/Green デプロイタイプを使用して、Amazon ECS サービスが最初に作成されたとき、Amazon ECS タスクセットが作成されます。
- Application Load Balancer または Network Load Balancer の使用にサービスを設定する必要があります。ロードバランサーの要件は以下のとおりです。
  - 本番稼働用リスナーをロードバランサーに追加する必要があります。これは本番トラフィックをルーティングするために使用されます。
  - オプションテストリスナーをロードバランサーに追加することができます。これはテストトラフィックをルーティングするために使用されます。テストリスナーを指定する場

合、CodeDeploy はデプロイメント中にテストトラフィックを置換タスクセットにルーティングします。

- 本番稼働用とテストリスナーの両方が同じロードバランサーに属している必要があります。
- ロードバランサーに対してターゲットグループを定義する必要があります。ターゲットグループは本番稼働用リスナーを通じてトラフィックをサービスの元のタスクセットにルーティングします。
- Network Load Balancer を使用する場合、`CodeDeployDefault.ECSAllAtOnce` デプロイ設定のみがサポートされます。

- サービスの自動スケーリングと ブルー/グリーンデプロイタイプを使用するように設定されたサービスでは、自動スケーリングはデプロイ中にブロックされませんが、状況によってはデプロイが失敗する場合があります。以下では、この動作について詳しく説明します。
  - サービスがスケーリング中の状態でデプロイが開始されると、グリーンタスクセットが作成され、CodeDeploy は Green タスクセットが定常状態になるまで最大 1 時間待機し、完了するまでトラフィックをシフトしません。
  - サービスが ブルー/グリーンデプロイのプロセス中で、スケーリングイベントが発生した場合、トラフィックは 5 分間シフトし続けます。サービスが 5 分以内に定常状態にならない場合、CodeDeploy はデプロイを停止し、失敗としてマークします。
- Fargate または `CODE_DEPLOY` のデプロイコントローラータイプを使用するタスクは、`DAEMON` スケジューリング戦略をサポートしません。
- 最初に CodeDeploy アプリケーションおよびデプロイグループを作成する際に、以下を指定する必要があります。
  - ロードバランサーに対して 2 つのターゲットグループを定義する必要があります。1 つのターゲットグループは、Amazon ECS サービスの作成時に、ロードバランサーに対して定義された最初のターゲットグループです。2 番目のターゲットグループの唯一の要件は、サービスが使用するものとは別のロードバランサーに関連付けることはできないということです。
- Amazon ECS サービスに対して CodeDeploy デプロイを作成すると、CodeDeploy は置換タスクセット (または Green タスクセット) をデプロイで作成します。テストリスナーをロードバランサーに追加した場合、CodeDeploy はテストトラフィックを置換タスクセットにルーティングします。この時点で検証テストを実行できます。次に、CodeDeploy は本番稼働用トラフィックを元のタスクセットから置換タスクセットに再ルーティングします。このときデプロイグループへのトラフィックの再ルーティング設定に従います。

必要な IAM 許可

ブルー/グリーンデプロイは、Amazon ECS と CodeDeploy API の組み合わせによって実現されています。ユーザーは、Amazon ECS ブルー/グリーンデプロイを使用するためには、これらのサービスに対する適切なアクセス権限が必要です。AWS マネジメントコンソールまたは AWS CLI または SDK を使用します。

サービスの作成や更新に使用するデフォルトの IAM アクセス権限に加えて、Amazon ECS には次のアクセス権限が必要です。AmazonECS_FullAccess IAM ポリシーには、次の許可が追加されています。詳細については、「AmazonECS_FullAccess」を参照してください。

- codedeploy:CreateApplication
- codedeploy:CreateDeployment
- codedeploy:CreateDeploymentGroup
- codedeploy:GetApplication
- codedeploy:GetDeployment
- codedeploy:GetDeploymentGroup
- codedeploy:ListApplications
- codedeploy:ListDeploymentGroups
- codedeploy:ListDeployments
- codedeploy:StopDeployment
- codedeploy:GetDeploymentTarget
- codedeploy:ListDeploymentTargets
- codedeploy:GetDeploymentConfig
- codedeploy:GetApplicationRevision
- codedeploy:RegisterApplicationRevision
- codedeploy:BatchGetApplicationRevisions
- codedeploy:BatchGetDeploymentGroups
- codedeploy:BatchGetDeployments
- codedeploy:BatchGetApplications
- codedeploy:ListApplicationRevisions
- codedeploy:ListDeploymentConfigs
- codedeploy:ContinueDeployment

- `sns:ListTopics`
- `cloudwatch:DescribeAlarms`
- `lambda:ListFunctions`

> **Note**
>
> タスクおよびサービスを実行するために必要な標準の Amazon ECS アクセス権限を付与するだけでなく、ユーザーにはタスクの IAM ロールを使用する `iam:PassRole` アクセス権限も必要です。

CodeDeploy には、Amazon ECS API の呼び出し、Elastic Load Balancing の変更、Lambda 関数の起動、CloudWatch アラームの記述のアクセス権限が必要です。また、ユーザーの代わりにサービスの必要数を変更するためのアクセス許可も必要です。Blue/Green デプロイタイプを使用する Amazon ECS サービスを作成する前に、IAM ロール (`ecsCodeDeployRole`) を作成する必要があります。詳細については、「Amazon ECS CodeDeploy IAM ロール」を参照してください。

CodeDeploy ブルー/グリーンデプロイを Amazon ECS ブルー/グリーンデプロイに移行

CodeDeploy ブルー/グリーンデプロイと Amazon ECS ブルー/グリーンデプロイの機能は類似していますが、設定および管理方法が異なります。

CodeDeploy ブルー/グリーンデプロイの概要

CodeDeploy を使用して Amazon ECS サービスを作成すると、以下を実行できます。

1. 本番リスナーおよび (必要に応じて) テストリスナーを使用してロードバランサーを作成します。各リスナーは、すべてのトラフィックを単一のターゲットグループ (プライマリターゲットグループ) にルーティングする 1 つの (デフォルト) ルールで設定されます。
2. `deploymentController` タイプが `CODE_DEPLOY` に設定された状態で、リスナーおよびターゲットグループを使用するように設定された Amazon ECS サービスを作成します。サービスを作成すると、指定されたターゲットグループに登録された (ブルー) タスクセットが作成されます。
3. CodeDeploy のデプロイグループ (CodeDeploy アプリケーションの一部として) を作成し、デプロイ動作を制御する Amazon ECS クラスター、サービス名、ロードバランサーリスナー、2 つのターゲットグループ (本番のリスナールールで使用されるプライマリターゲットグループ、ならびに代替タスクに使用されるセカンダリターゲットグループ)、サービスロール (Amazon ECS および Elastic Load Balancing リソースを操作するアクセス許可を CodeDeploy に付与するため)、およびさまざまなパラメータの詳細を使って設定します。

CodeDeploy を使用すると、`CreateDeployment()` を使用してサービスの新しいバージョンがデプロイされ、CodeDeploy のアプリケーション名、デプロイグループ名、新しいリビジョンおよびオプションのライフサイクルフックの詳細を提供する AppSpec ファイルが指定されます。CodeDeploy デプロイでは代替 (緑) タスクセットが作成され、タスクがセカンダリターゲットグループに登録されます。正常になると、テスト (オプション) および本番で利用できます。どちらの場合も、グリーンタスクセットに関連付けられたセカンダリターゲットグループを指すようにリスナールールを変更することで、再ルーティングが実現します。ロールバックは、本番のリスナールールをプライマリターゲットグループに戻すことで実現します。

Amazon ECS ブルー/グリーンデプロイの概要

Amazon ECS ブルー/グリーンデプロイを使用すると、デプロイ設定が Amazon ECS サービス自体の一部になります。

1. 重みが 1 および 0 で構成される 2 つのターゲットグループを含むルールで、ロードバランサーの本番リスナーを事前設定する必要があります。
2. 次のリソースを指定するか、サービスリソースを更新する必要があります。
   - このリスナールールの ARN
   - 2 つのターゲットグループ
   - Elastic Load Balancing API を呼び出すアクセス許可を Amazon ECS に付与する IAM ロール
   - Lambda 関数を実行する IAM ロール (オプション)
   - `deploymentController` タイプを `ECS` に設定し、`deploymentConfiguration.strategy` を `BLUE_GREEN` に設定します。タスクがプライマリターゲットグループに登録されている (ブルー) サービスデプロイが作成されます。

Amazon ECS ブルー/グリーンを使用すると、Amazon ECS `UpdateService()` を呼び出すことで新しいサービスリビジョンが作成され、新しいリビジョンの詳細が渡されます。サービスデプロイによって新しい (グリーン) サービスリビジョンタスクが作成され、セカンダリターゲットグループに登録されます。Amazon ECS はリスナールールの重みを切り替えることで、再ルーティングおよびロールバックオペレーションを処理します。

主要な実装上の相違点

どちらのアプローチによってもタスクの初期セットが作成されますが、基盤となる実装は異なります。

- CodeDeploy ではタスクセットが使用されますが、Amazon ECS ではサービスリビジョンが使用されます。Amazon ECS のタスクセットは、Amazon ECS サービスリビジョンおよびデプロイに置き換えられた古いコンストラクトです。後者はデプロイプロセス、サービスデプロイ、サービスリビジョン履歴をさらに詳細に可視化します。
- CodeDeploy を使用すると、ライフサイクルフックは `CreateDeployment()` に提供される AppSpec ファイルの一部として指定されます。つまり、フックは 1 つのデプロイから次のデプロイに変更できます。Amazon ECS ブルー/グリーンを使用すると、フックはサービス設定の一部として指定され、更新には `UpdateService()` の呼び出しが必要になります。
- CodeDeploy および Amazon ECS ブルー/グリーンは両方ともフック実装に Lambda を使用しますが、予想される入力および出力は異なります。

  CodeDeploy を使用すると、関数は `PutLifecycleEventHookExecutionStatus()` を呼び出してフックステータスを返す必要があり、フックステータスは `SUCCEEDED` または `FAILED` になります。Amazon ECS を使用すると、Lambda レスポンスが使用されてフックステータスが示されます。
- CodeDeploy は各フックを 1 回限りの呼び出しとして呼び出し、最終的な実行ステータスは通常 1 時間以内に返されます。Amazon ECS フックは、`IN_PROGRESS` インジケータを返すことができるという点でより柔軟です。このインジケータにより、フックが `SUCCEEDED` または `FAILED` になるまで繰り返し呼び出される必要があることが示されます。詳細については、「Amazon ECS サービスデプロイのライフサイクルフック」を参照してください。

移行アプローチ

CodeDeploy ブルー/グリーンデプロイから Amazon ECS ブルー/グリーンデプロイに移行するアプローチには主に 3 つあります。各アプローチには、複雑さ、リスク、ロールバック機能、潜在的なダウンタイムの点で特性が異なります。

CodeDeploy に使用される Elastic Load Balancing リソースと同じものを再利用します

CodeDeploy デプロイコントローラーの代わりに、ブルー/グリーンデプロイ戦略で Amazon ECS デプロイコントローラーを使用するように、既存の Amazon ECS サービスを更新します。このアプローチを使用する場合は、次の点を考慮してください。

- 既存の Amazon ECS サービスデプロイコントローラーおよびデプロイ戦略を更新するため、移行手順がよりシンプルになります。
- 正しく設定および移行されている場合、ダウンタイムはありません。
- ロールバックには、サービスリビジョンを元に戻す必要があります。

- ブルー/グリーン設定が並行していないため、リスクが高くなります。

CodeDeploy に使用されるロードバランサーリスナーおよびターゲットグループと同じものを使用します。CloudFormation を使用している場合は、「CloudFormation CodeDeploy ブルー/グリーンデプロイのテンプレートを Amazon ECS ブルー/グリーン CloudFormation テンプレートへの移行」を参照してください。

1. 本番/テストのリスナーのデフォルトルールを変更して代替ターゲットグループを含め、プライマリターゲットグループの重みを 1 に設定し、代替ターゲットグループの重みを 0 に設定します。

   CodeDeploy の場合、サービスにアタッチされたロードバランサーのリスナーは、すべてのトラフィックを単一のターゲットグループにルーティングする、1 つの (デフォルト) ルールで設定されます。Amazon ECS ブルー/グリーンの場合、ロードバランサーリスナーは重みを持つ 2 つのターゲットグループが含まれるルールで事前設定する必要があります。プライマリターゲットグループは 1 に重み付けされ、代替ターゲットグループは 0 に重み付けされる必要があります。

2. `UpdateService` API を呼び出して `deploymentController` パラメータを `ECS` に設定し、`deploymentStrategy` パラメータを `BLUE_GREEN` に設定することで、既存の Amazon ECS サービスを更新します。ターゲットグループの ARN、代替ターゲットグループ、本番リスナー、オプションのテストリスナーを指定します。
3. サービスが期待どおりに動作することを確認します。
4. 現在は Amazon ECS ブルー/グリーンを使用しているため、この Amazon ECS サービスの CodeDeploy 設定を削除します。

既存のロードバランサーを含む新しいサービス

このアプローチでは、移行にブルー/グリーン戦略を使用します。

このアプローチを使用する場合は、次の点を考慮してください。

- 中断は最小限に抑えられます。Elastic Load Balancing のポートスワップ中にのみ発生します。
- ロールバックには、Elastic Load Balancing のポートスワップを元に戻す必要があります。
- 並列設定があるため、リスクが低くなります。したがって、トラフィックが移行する前にテストできます。

1. 必要に応じてこの設定に簡単にロールバックできるように、CodeDeploy 設定のリスナー、ターゲットグループ、Amazon ECS サービスはそのままにします。
2. 既存のロードバランサーの下で、新しいターゲットグループおよび新しいリスナー (元のリスナーとは異なるポートを所有) を作成します。次に、既存の Amazon ECS サービスに一致する新しい Amazon ECS サービスを作成しますが、ECS をデプロイコントローラー、BLUE_GREEN をデプロイ戦略として使用し、新しいターゲットグループおよび新しいリスナールールの ARN を渡します。
3. サービスに HTTP トラフィックを手動で送信し、新しい設定を確認します。問題がなければ、元のリスナーと新しいリスナーのポートをスワップし、トラフィックを新しい設定にルーティングします。
4. 新しい設定を確認します。依然として期待どおりにすべてが動作する場合は、CodeDeploy 設定を削除します。

新しいロードバランサーを持った新しいサービス

前のアプローチと同様に、このアプローチでは移行にブルー/グリーン戦略を使用します。主な違いは、CodeDeploy 設定から Amazon ECS ブルー/グリーンの設定への切り替えが、ロードバランサーの上にあるリバースプロキシレイヤーで行われることです。リバースプロキシレイヤーのサンプル実装は、Route 53 および CloudFront です。

このアプローチはこのリバースプロキシレイヤーを既に持っているお客様に適しており、サービスとのすべての通信がこのリバースプロキシレイヤーを通じて行われている場合にも適しています (例えば、ロードバランサーレベルで直接通信がない場合など)。

このアプローチを使用する場合は、次の点を考慮してください。

- リバースプロキシレイヤーが必要です。
- 既存の Amazon ECS サービスデプロイコントローラーおよびデプロイ戦略を更新する必要があるため、移行手順がより複雑になります。
- 中断は最小限に抑えられます。Elastic Load Balancing のポートスワップ中にのみ発生します。
- ロールバックには、プロキシ設定の変更を元に戻す必要があります。
- 並列設定があるため、リスクが低くなります。したがって、トラフィックが移行する前にテストできます。

1. 既存の CodeDeploy 設定を変更しないでください (ロードバランサー、リスナー、ターゲットグループ、Amazon ECS サービス、CodeDeploy デプロイグループ)。

2. Amazon ECS ブルー/グリーンデプロイ用に設定された新しいロードバランサー、ターゲットグループ、リスナーを作成します。

   適切なリソースを設定します。

   - Application Load Balancer – 詳細については、「ブルー/グリーンデプロイ、リニアデプロイおよびカナリアデプロイ用の Application Load Balancer リソース」を参照してください。
   - Network Load Balancer – 詳細については、「Amazon ECS ブルー/グリーン、リニアおよびカナリアデプロイの Network Load Balancer リソース」を参照してください。
3. ECS および BLUE_GREEN をそれぞれデプロイコントローラーおよびデプロイ戦略とし、新しいロードバランサーリソースを指す新しいサービスを作成します。
4. 新しいロードバランサーを通じてテストすることで、新しいセットアップを確認します。
5. リバースプロキシ設定を更新し、トラフィックを新しいロードバランサーにルーティングします。
6. 新しいサービスリビジョンに従って、期待どおりにすべてが動作し続けた場合、CodeDeploy 設定を削除します。

次のステップ

Amazon ECS ブルー/グリーンデプロイに移行した後

- CodeDeploy UpdateService API の代わりに、Amazon ECS CreateDeployment API を使用するように、デプロイスクリプトと CI/CD パイプラインを更新します。
- CodeDeploy デプロイの代わりに、Amazon ECS サービスのデプロイを追跡するように、モニタリングおよびアラートを更新します。
- 期待どおりに機能することを確認するため、新しいデプロイプロセスの自動テストを実装することを検討してください。

CodeDeploy ブルー/グリーンから Amazon ECS ブルー/グリーンサービスデプロイへの移行

Amazon ECS ブルー/グリーンデプロイを使用すると、本番環境に実装する前にサービスを変更してテストできます。

Amazon ECS ブルー/グリーンデプロイ用に新しいライフサイクルフックを作成する必要があります。

前提条件

ブルー/グリーンデプロイを開始する前に、次の操作を実行します。

1. Amazon ECS CodeDeploy IAM ロールを次のアクセス許可に置き換えてください。
   - Elastic Load Balancing のアクセス許可の詳細については、「ロードバランサー用の Amazon ECS インフラストラクチャの IAM ロール」を参照してください。
   - Lambda のアクセス許可の詳細については、「Amazon ECS ブルー/グリーンデプロイの Lambda 関数に必要となるアクセス許可」を参照してください。
2. CodeDeploy オートメーションをオフにします。詳細については、「CodeDeploy ユーザーガイド」の「Working with deployment groups in CodeDeploy」を参照してください。
3. CodeDeploy ブルー/グリーンデプロイに次の情報があることを確認してください。この情報は Amazon ECS ブルー/グリーンデプロイに再利用できます。
   - 本番ターゲットグループ
   - 本番リスナー
   - 本番ルール
   - テストターゲットグループ

     これはグリーンサービスリビジョンのターゲットグループです。
4. Application Load Balancer ターゲットグループがリスナールールに適切に関連付けられていることを確認してください。
   - テストリスナーを使用していない場合、両方のターゲットグループ (本番およびテスト) を本番のリスナールールに関連付ける必要があります。
   - テストリスナーを使用している場合、1 つのターゲットグループを本番のリスナールールにリンクし、もう 1 つのターゲットグループをテストのリスナールールにリンクする必要があります。

   この要件が満たされない場合、サービスのデプロイは失敗し、次のエラーが表示されます。`Service deployment rolled back because of invalid networking configuration. Both targetGroup and alternateTargetGroup must be associated with the productionListenerRule or testListenerRule.`
5. サービスに継続中のサービスデプロイがないことを確認します。詳細については、「Amazon ECS サービスデプロイを使用してサービス履歴を表示する」を参照してください。
6. Amazon ECS ブルー/グリーンデプロイでは、サービスが次のいずれかの機能を使用する必要があります。適切なリソースを設定してください。

- Application Load Balancer – 詳細については、「ブルー/グリーンデプロイ、リニアデプロイおよびカナリアデプロイ用の Application Load Balancer リソース」を参照してください。
- Network Load Balancer – 詳細については、「Amazon ECS ブルー/グリーン、リニアおよびカナリアデプロイの Network Load Balancer リソース」を参照してください。
- Service Connect – 詳細については、「Amazon ECS ブルー/グリーンデプロイ、リニアデプロイおよびカナリアデプロイ用の Service Connect リソース」を参照してください。

7. Amazon ECS ブルー/グリーンデプロイのステージのライフサイクルステージに Lambda 関数を実行するかどうかを決定します。
   - スケールアップ前
   - スケールアップ後
   - テストトラフィックの移行
   - テストトラフィックの移行後
   - 本番トラフィックの移行
   - 本番トラフィックの移行後

   ライフサイクルステージごとに Lambda 関数を作成します。詳細については、「AWS Lambda デベロッパーガイド」の「コンソールで Lambda 関数の作成」を参照してください。

サービスのデプロイコントローラーの更新の詳細については、「Amazon ECS サービスのパラメータの更新」を参照してください。

手順

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
2. [クラスター] ページで、クラスターを選択します。

   クラスター詳細のページが表示されます。
3. [サービス] タブでサービスを選択します。

   [サービスの詳細] ページが表示されます。
4. バナーで [デプロイコントローラータイプの更新] を選択します。

   [デプロイコントローラータイプの移行] ページが表示されます。
5. [新規] を展開し、次のパラメータを指定します。

   a. [デプロイコントローラータイプ] では [ECS] を選択します。

b. [デプロイ戦略] では、[ブルー/グリーン] を選択します。

c. [ベイク時間] には、ブルーおよびグリーンのサービスリビジョンの両方が実行される時間を入力します。

d. ライフサイクルステージに Lambda 関数を実行するには、[デプロイライフサイクルフック] で一意の Lambda 関数ごとに次の操作を行います。

i. [Add] (追加) を選択します。

実行する一意の関数ごとにこの操作を繰り返します。

ii. [Lambda 関数] には、関数名を入力します。

iii. [ロール] には、ブルー/グリーンのアクセス許可を持つ前提条件で作成したロールを選択します。

詳細については、「Amazon ECS ブルー/グリーンデプロイの Lambda 関数に必要となるアクセス許可」を参照してください。

iv. [ライフサイクルステージ] には、Lambda 関数が実行されるステージを選択します。

v. (オプション) [フックの詳細] には、フックに関する情報を提供するキーおよび値のペアを入力します。

6. [ロードバランシング] を展開し、次の内容を設定します。

a. [ロール] には、ブルー/グリーンのアクセス許可を持つ前提条件で作成したロールを選択します。

詳細については、「Amazon ECS ブルー/グリーンデプロイの Lambda 関数に必要となるアクセス許可」を参照してください。

b. [リスナー] には、CodeDeploy ブルー/グリーンデプロイから本番リスナーを選択します。

c. [本番ルール] には、CodeDeploy ブルー/グリーンデプロイから本番ルールを選択します。

d. [テストルール] には、CodeDeploy ブルー/グリーンデプロイからテストルールを選択します。

e. [ターゲットグループ] には、CodeDeploy ブルー/グリーンデプロイから本番ターゲットグループを選択します。

f. [代替ターゲットグループ] には、CodeDeploy ブルー/グリーンデプロイからテストターゲットグループを選択します。

7. [更新] を選択します。

次のステップ

- サービスを更新してデプロイを開始します。詳細については、「Amazon ECS サービスを更新する」を参照してください。
- デプロイプロセスをモニタリングし、ブルー/グリーンのパターンに従っていることを確認します。
  - グリーンサービスリビジョンが作成され、スケールアップされる
  - テストトラフィックがグリーンリビジョンにルーティングされている (設定されている場合)
  - 本番トラフィックがグリーンリビジョンに移行されている
  - ベイク時間が過ぎたら、ブルーリビジョンが終了する

CloudFormation CodeDeploy ブルー/グリーンデプロイのテンプレートを Amazon ECS ブルー/グリーン CloudFormation テンプレートへの移行

Amazon ECS サービスの CodeDeploy ブルー/グリーンデプロイを使用する CloudFormation テンプレートを、ネイティブの Amazon ECS ブルー/グリーンデプロイ戦略を使用するテンプレートに移行します。移行は、「CodeDeploy に使用した Elastic Load Balancing リソースと同じものを再利用する」アプローチに従います。詳細については、「the section called “CodeDeploy ブルー/グリーンデプロイを Amazon ECS ブルー/グリーンデプロイに移行”」を参照してください。

ソーステンプレート

このテンプレートは `AWS::CodeDeployBlueGreen` 変換および `AWS::CodeDeploy::BlueGreen` フックを使用し、Amazon ECS サービスにブルー/グリーンデプロイを実装します。

ソース

これは CodeDeploy ブルー/グリーンデプロイを使用する完全な CloudFormation テンプレートです。詳細については、「AWS CloudFormation ユーザーガイド」の「ブルー/グリーンデプロイテンプレートの例」を参照してください。

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Parameters": {
    "Vpc": {
      "Type": "AWS::EC2::VPC::Id"
    },
    "Subnet1": {
      "Type": "AWS::EC2::Subnet::Id"
```

```
    },
    "Subnet2": {
      "Type": "AWS::EC2::Subnet::Id"
    }
  },
  "Transform": [
    "AWS::CodeDeployBlueGreen"
  ],
  "Hooks": {
    "CodeDeployBlueGreenHook": {
      "Type": "AWS::CodeDeploy::BlueGreen",
      "Properties": {
        "TrafficRoutingConfig": {
          "Type": "TimeBasedCanary",
          "TimeBasedCanary": {
            "StepPercentage": 15,
            "BakeTimeMins": 5
          }
        },
        "Applications": [
          {
            "Target": {
              "Type": "AWS::ECS::Service",
              "LogicalID": "ECSDemoService"
            },
            "ECSAttributes": {
              "TaskDefinitions": [
                "BlueTaskDefinition",
                "GreenTaskDefinition"
              ],
              "TaskSets": [
                "BlueTaskSet",
                "GreenTaskSet"
              ],
              "TrafficRouting": {
                "ProdTrafficRoute": {
                  "Type": "AWS::ElasticLoadBalancingV2::Listener",
                  "LogicalID": "ALBListenerProdTraffic"
                },
                "TargetGroups": [
                  "ALBTargetGroupBlue",
                  "ALBTargetGroupGreen"
                ]
              }
```

```
            }
          }
        ]
      }
    }
  },
  "Resources": {
    "ExampleSecurityGroup": {
      "Type": "AWS::EC2::SecurityGroup",
      "Properties": {
        "GroupDescription": "Security group for ec2 access",
        "VpcId": {"Ref": "Vpc"},
        "SecurityGroupIngress": [
          {
            "IpProtocol": "tcp",
            "FromPort": 80,
            "ToPort": 80,
            "CidrIp": "0.0.0.0/0"
          },
          {
            "IpProtocol": "tcp",
            "FromPort": 8080,
            "ToPort": 8080,
            "CidrIp": "0.0.0.0/0"
          },
          {
            "IpProtocol": "tcp",
            "FromPort": 22,
            "ToPort": 22,
            "CidrIp": "0.0.0.0/0"
          }
        ]
      }
    },
    "ALBTargetGroupBlue": {
      "Type": "AWS::ElasticLoadBalancingV2::TargetGroup",
      "Properties": {
        "HealthCheckIntervalSeconds": 5,
        "HealthCheckPath": "/",
        "HealthCheckPort": "80",
        "HealthCheckProtocol": "HTTP",
        "HealthCheckTimeoutSeconds": 2,
        "HealthyThresholdCount": 2,
        "Matcher": {
```

```
          "HttpCode": "200"
        },
        "Port": 80,
        "Protocol": "HTTP",
        "Tags": [
          {
            "Key": "Group",
            "Value": "Example"
          }
        ],
        "TargetType": "ip",
        "UnhealthyThresholdCount": 4,
        "VpcId": {"Ref": "Vpc"}
      }
    },
    "ALBTargetGroupGreen": {
      "Type": "AWS::ElasticLoadBalancingV2::TargetGroup",
      "Properties": {
        "HealthCheckIntervalSeconds": 5,
        "HealthCheckPath": "/",
        "HealthCheckPort": "80",
        "HealthCheckProtocol": "HTTP",
        "HealthCheckTimeoutSeconds": 2,
        "HealthyThresholdCount": 2,
        "Matcher": {
          "HttpCode": "200"
        },
        "Port": 80,
        "Protocol": "HTTP",
        "Tags": [
          {
            "Key": "Group",
            "Value": "Example"
          }
        ],
        "TargetType": "ip",
        "UnhealthyThresholdCount": 4,
        "VpcId": {"Ref": "Vpc"}
      }
    },
    "ExampleALB": {
      "Type": "AWS::ElasticLoadBalancingV2::LoadBalancer",
      "Properties": {
        "Scheme": "internet-facing",
```

```
        "SecurityGroups": [
          {"Ref": "ExampleSecurityGroup"}
        ],
        "Subnets": [
          {"Ref": "Subnet1"},
          {"Ref": "Subnet2"}
        ],
        "Tags": [
          {
            "Key": "Group",
            "Value": "Example"
          }
        ],
        "Type": "application",
        "IpAddressType": "ipv4"
      }
    },
    "ALBListenerProdTraffic": {
      "Type": "AWS::ElasticLoadBalancingV2::Listener",
      "Properties": {
        "DefaultActions": [
          {
            "Type": "forward",
            "ForwardConfig": {
              "TargetGroups": [
                {
                  "TargetGroupArn": {"Ref": "ALBTargetGroupBlue"},
                  "Weight": 1
                }
              ]
            }
          }
        ],
        "LoadBalancerArn": {"Ref": "ExampleALB"},
        "Port": 80,
        "Protocol": "HTTP"
      }
    },
    "ALBListenerProdRule": {
      "Type": "AWS::ElasticLoadBalancingV2::ListenerRule",
      "Properties": {
        "Actions": [
          {
            "Type": "forward",
```

```
            "ForwardConfig": {
              "TargetGroups": [
                {
                  "TargetGroupArn": {"Ref": "ALBTargetGroupBlue"},
                  "Weight": 1
                }
              ]
            }
          }
        ],
        "Conditions": [
          {
            "Field": "http-header",
            "HttpHeaderConfig": {
              "HttpHeaderName": "User-Agent",
              "Values": [
                "Mozilla"
              ]
            }
          }
        ],
        "ListenerArn": {"Ref": "ALBListenerProdTraffic"},
        "Priority": 1
      }
    },
    "ECSTaskExecutionRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Sid": "",
              "Effect": "Allow",
              "Principal": {
                "Service": "ecs-tasks.amazonaws.com"
              },
              "Action": "sts:AssumeRole"
            }
          ]
        },
        "ManagedPolicyArns": [
          "arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy"
        ]
```

```
      }
    },
    "BlueTaskDefinition": {
      "Type": "AWS::ECS::TaskDefinition",
      "Properties": {
        "ExecutionRoleArn": {"Fn::GetAtt": ["ECSTaskExecutionRole", "Arn"]},
        "ContainerDefinitions": [
          {
            "Name": "DemoApp",
            "Image": "nginxdemos/hello:latest",
            "Essential": true,
            "PortMappings": [
              {
                "HostPort": 80,
                "Protocol": "tcp",
                "ContainerPort": 80
              }
            ]
          }
        ],
        "RequiresCompatibilities": [
          "FARGATE"
        ],
        "NetworkMode": "awsvpc",
        "Cpu": "256",
        "Memory": "512",
        "Family": "ecs-demo"
      }
    },
    "ECSDemoCluster": {
      "Type": "AWS::ECS::Cluster",
      "Properties": {}
    },
    "ECSDemoService": {
      "Type": "AWS::ECS::Service",
      "Properties": {
        "Cluster": {"Ref": "ECSDemoCluster"},
        "DesiredCount": 1,
        "DeploymentController": {
          "Type": "EXTERNAL"
        }
      }
    },
    "BlueTaskSet": {
```

```
      "Type": "AWS::ECS::TaskSet",
      "Properties": {
        "Cluster": {"Ref": "ECSDemoCluster"},
        "LaunchType": "FARGATE",
        "NetworkConfiguration": {
          "AwsVpcConfiguration": {
            "AssignPublicIp": "ENABLED",
            "SecurityGroups": [
              {"Ref": "ExampleSecurityGroup"}
            ],
            "Subnets": [
              {"Ref": "Subnet1"},
              {"Ref": "Subnet2"}
            ]
          }
        },
        "PlatformVersion": "1.4.0",
        "Scale": {
          "Unit": "PERCENT",
          "Value": 100
        },
        "Service": {"Ref": "ECSDemoService"},
        "TaskDefinition": {"Ref": "BlueTaskDefinition"},
        "LoadBalancers": [
          {
            "ContainerName": "DemoApp",
            "ContainerPort": 80,
            "TargetGroupArn": {"Ref": "ALBTargetGroupBlue"}
          }
        ]
      }
    },
    "PrimaryTaskSet": {
      "Type": "AWS::ECS::PrimaryTaskSet",
      "Properties": {
        "Cluster": {"Ref": "ECSDemoCluster"},
        "Service": {"Ref": "ECSDemoService"},
        "TaskSetId": {"Fn::GetAtt": ["BlueTaskSet", "Id"]}
      }
    }
  }
}
```

移行手順

CodeDeploy 固有のリソースを削除する

次のプロパティが不要になりました。

- AWS::CodeDeployBlueGreen 変換
- CodeDeployBlueGreenHook フック
- GreenTaskDefinition および GreenTaskSet リソース (Amazon ECS によって管理される)
- PrimaryTaskSet リソース (Amazon ECS によって内部でタスクセットが管理される)

ロードバランサーリスナーを再設定する

ALBListenerProdTraffic リソースを変更して、2 つのターゲットグループを持つ転送アクションを使用します。

```
{
  "DefaultActions": [
    {
      "Type": "forward",
      "ForwardConfig": {
        "TargetGroups": [
          {
            "TargetGroupArn": {"Ref": "ALBTargetGroupBlue"},
            "Weight": 1
          },
          {
            "TargetGroupArn": {"Ref": "ALBTargetGroupGreen"},
            "Weight": 0
          }
        ]
      }
    }
  ]
}
```

デプロイプロパティを更新します

次の内容を更新して追加します。

- DeploymentController プロパティを EXTERNAL から ECS に更新します。

- Strategy プロパティを追加して、BLUE_GREEN に設定します。
- BakeTimeInMinutes プロパティを追加します。

```
{
  "DeploymentConfiguration": {
    "MaximumPercent": 200,
    "MinimumHealthyPercent": 100,
    "DeploymentCircuitBreaker": {
      "Enable": true,
      "Rollback": true
    },
    "BakeTimeInMinutes": 5,
    "Strategy": "BLUE_GREEN"
  }
}
```

- ロードバランサー設定をサービスに追加します。

```
{
  "LoadBalancers": [
    {
      "ContainerName": "DemoApp",
      "ContainerPort": 80,
      "TargetGroupArn": {"Ref": "ALBTargetGroupBlue"},
      "AdvancedConfiguration": {
        "AlternateTargetGroupArn": {"Ref": "ALBTargetGroupGreen"},
        "ProductionListenerRule": {"Ref": "ALBListenerProdRule"},
        "RoleArn": {"Fn::GetAtt": ["ECSInfrastructureRoleForLoadBalancers", "Arn"]}
      }
    }
  ]
}
```

- サービスにタスク定義リファレンスを追加します。

```
{
  "TaskDefinition": {"Ref": "BlueTaskDefinition"}
}
```

AmazonECSInfrastructureRolePolicyForLoadBalancers ロールを作成します

Amazon ECS にロードバランサーリソースの管理を可能にする新しい IAM ロールを追加します。詳細については、the section called “ロードバランサー用のインフラストラクチャの IAM ロール”を参照してください。

レコメンデーションのテスト

1. 移行したテンプレートを非本番環境にデプロイします。
2. サービスが初期設定で正しくデプロイされることを確認してください。
3. タスク定義を更新してブルー/グリーンデプロイプロセスを観察することで、デプロイをテストします。
4. ブルーおよびグリーンのデプロイの間でトラフィックが正しく移行されることを確認してください。
5. デプロイの失敗を強制し、ロールバック機能をテストします。

移行後のテンプレート

最終テンプレート

これは、Amazon ECS ブルー/グリーンデプロイを使用する完全な CloudFormation テンプレートです。

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Parameters": {
    "Vpc": {
      "Type": "AWS::EC2::VPC::Id"
    },
    "Subnet1": {
      "Type": "AWS::EC2::Subnet::Id"
    },
    "Subnet2": {
      "Type": "AWS::EC2::Subnet::Id"
    }
  },
  "Resources": {
    "ExampleSecurityGroup": {
      "Type": "AWS::EC2::SecurityGroup",
      "Properties": {
        "GroupDescription": "Security group for ec2 access",
```

```
        "VpcId": {"Ref": "Vpc"},
        "SecurityGroupIngress": [
          {
            "IpProtocol": "tcp",
            "FromPort": 80,
            "ToPort": 80,
            "CidrIp": "0.0.0.0/0"
          },
          {
            "IpProtocol": "tcp",
            "FromPort": 8080,
            "ToPort": 8080,
            "CidrIp": "0.0.0.0/0"
          },
          {
            "IpProtocol": "tcp",
            "FromPort": 22,
            "ToPort": 22,
            "CidrIp": "0.0.0.0/0"
          }
        ]
      }
    },
    "ALBTargetGroupBlue": {
      "Type": "AWS::ElasticLoadBalancingV2::TargetGroup",
      "Properties": {
        "HealthCheckIntervalSeconds": 5,
        "HealthCheckPath": "/",
        "HealthCheckPort": "80",
        "HealthCheckProtocol": "HTTP",
        "HealthCheckTimeoutSeconds": 2,
        "HealthyThresholdCount": 2,
        "Matcher": {
          "HttpCode": "200"
        },
        "Port": 80,
        "Protocol": "HTTP",
        "Tags": [
          {
            "Key": "Group",
            "Value": "Example"
          }
        ],
        "TargetType": "ip",
```

```
        "UnhealthyThresholdCount": 4,
        "VpcId": {"Ref": "Vpc"}
      }
    },
    "ALBTargetGroupGreen": {
      "Type": "AWS::ElasticLoadBalancingV2::TargetGroup",
      "Properties": {
        "HealthCheckIntervalSeconds": 5,
        "HealthCheckPath": "/",
        "HealthCheckPort": "80",
        "HealthCheckProtocol": "HTTP",
        "HealthCheckTimeoutSeconds": 2,
        "HealthyThresholdCount": 2,
        "Matcher": {
          "HttpCode": "200"
        },
        "Port": 80,
        "Protocol": "HTTP",
        "Tags": [
          {
            "Key": "Group",
            "Value": "Example"
          }
        ],
        "TargetType": "ip",
        "UnhealthyThresholdCount": 4,
        "VpcId": {"Ref": "Vpc"}
      }
    },
    "ExampleALB": {
      "Type": "AWS::ElasticLoadBalancingV2::LoadBalancer",
      "Properties": {
        "Scheme": "internet-facing",
        "SecurityGroups": [
          {"Ref": "ExampleSecurityGroup"}
        ],
        "Subnets": [
          {"Ref": "Subnet1"},
          {"Ref": "Subnet2"}
        ],
        "Tags": [
          {
            "Key": "Group",
            "Value": "Example"
```

```
        }
      ],
      "Type": "application",
      "IpAddressType": "ipv4"
    }
  },
  "ALBListenerProdTraffic": {
    "Type": "AWS::ElasticLoadBalancingV2::Listener",
    "Properties": {
      "DefaultActions": [
        {
          "Type": "forward",
          "ForwardConfig": {
            "TargetGroups": [
              {
                "TargetGroupArn": {"Ref": "ALBTargetGroupBlue"},
                "Weight": 1
              },
              {
                "TargetGroupArn": {"Ref": "ALBTargetGroupGreen"},
                "Weight": 0
              }
            ]
          }
        }
      ],
      "LoadBalancerArn": {"Ref": "ExampleALB"},
      "Port": 80,
      "Protocol": "HTTP"
    }
  },
  "ALBListenerProdRule": {
    "Type": "AWS::ElasticLoadBalancingV2::ListenerRule",
    "Properties": {
      "Actions": [
        {
          "Type": "forward",
          "ForwardConfig": {
            "TargetGroups": [
              {
                "TargetGroupArn": {"Ref": "ALBTargetGroupBlue"},
                "Weight": 1
              },
              {
```

```
                "TargetGroupArn": {"Ref": "ALBTargetGroupGreen"},
                "Weight": 0
              }
            ]
          }
        }
      ],
      "Conditions": [
        {
          "Field": "http-header",
          "HttpHeaderConfig": {
            "HttpHeaderName": "User-Agent",
            "Values": [
              "Mozilla"
            ]
          }
        }
      ],
      "ListenerArn": {"Ref": "ALBListenerProdTraffic"},
      "Priority": 1
    }
  },
  "ECSTaskExecutionRole": {
    "Type": "AWS::IAM::Role",
    "Properties": {
      "AssumeRolePolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
              "Service": "ecs-tasks.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
          }
        ]
      },
      "ManagedPolicyArns": [
        "arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy"
      ]
    }
  },
  "ECSInfrastructureRoleForLoadBalancers": {
```

```
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Sid": "AllowAccessToECSForInfrastructureManagement",
              "Effect": "Allow",
              "Principal": {
                "Service": "ecs.amazonaws.com"
              },
              "Action": "sts:AssumeRole"
            }
          ]
        },
        "ManagedPolicyArns": [
          "arn:aws:iam::aws:policy/AmazonECSInfrastructureRolePolicyForLoadBalancers"
        ]
      }
    },
    "BlueTaskDefinition": {
      "Type": "AWS::ECS::TaskDefinition",
      "Properties": {
        "ExecutionRoleArn": {"Fn::GetAtt": ["ECSTaskExecutionRole", "Arn"]},
        "ContainerDefinitions": [
          {
            "Name": "DemoApp",
            "Image": "nginxdemos/hello:latest",
            "Essential": true,
            "PortMappings": [
              {
                "HostPort": 80,
                "Protocol": "tcp",
                "ContainerPort": 80
              }
            ]
          }
        ],
        "RequiresCompatibilities": [
          "FARGATE"
        ],
        "NetworkMode": "awsvpc",
        "Cpu": "256",
        "Memory": "512",
```

```
        "Family": "ecs-demo"
      }
    },
    "ECSDemoCluster": {
      "Type": "AWS::ECS::Cluster",
      "Properties": {}
    },
    "ECSDemoService": {
      "Type": "AWS::ECS::Service",
      "Properties": {
        "Cluster": {"Ref": "ECSDemoCluster"},
        "DesiredCount": 1,
        "DeploymentController": {
          "Type": "ECS"
        },
        "DeploymentConfiguration": {
          "MaximumPercent": 200,
          "MinimumHealthyPercent": 100,
          "DeploymentCircuitBreaker": {
            "Enable": true,
            "Rollback": true
          },
          "BakeTimeInMinutes": 5,
          "Strategy": "BLUE_GREEN"
        },
        "NetworkConfiguration": {
          "AwsvpcConfiguration": {
            "AssignPublicIp": "ENABLED",
            "SecurityGroups": [
              {"Ref": "ExampleSecurityGroup"}
            ],
            "Subnets": [
              {"Ref": "Subnet1"},
              {"Ref": "Subnet2"}
            ]
          }
        },
        "LaunchType": "FARGATE",
        "PlatformVersion": "1.4.0",
        "TaskDefinition": {"Ref": "BlueTaskDefinition"},
        "LoadBalancers": [
          {
            "ContainerName": "DemoApp",
            "ContainerPort": 80,
```

```
            "TargetGroupArn": {"Ref": "ALBTargetGroupBlue"},
            "AdvancedConfiguration": {
              "AlternateTargetGroupArn": {"Ref": "ALBTargetGroupGreen"},
              "ProductionListenerRule": {"Ref": "ALBListenerProdRule"},
              "RoleArn": {"Fn::GetAtt": ["ECSInfrastructureRoleForLoadBalancers",
 "Arn"]}
            }
          }
        ]
      }
    }
  }
}
```

CodeDeploy ブルー/グリーンから Amazon ECS ローリング更新サービスのデプロイへの移行

サービスデプロイを CodeDeploy ブルー/グリーンデプロイから Amazon ECS ローリング更新デプロイに移行できます。CodeDeploy の依存関係から統合デプロイの使用に移行します。

Amazon ECS サービススケジューラは、現在実行中のタスクを新しいタスクに置き換えます。ローリング更新中にサービスに対して Amazon ECS が追加または削除するタスク数は、サービスデプロイ設定により制御されます。

前提条件

ブルー/グリーンデプロイを開始する前に、次の操作を実行します。

1. Amazon ECS CodeDeploy IAM ロールが不要になりました。
2. CodeDeploy オートメーションをオフにします。詳細については、「CodeDeploy ユーザーガイド」の「Working with deployment groups in CodeDeploy」を参照してください。
3. サービスに継続中のサービスデプロイがないことを確認します。詳細については、「Amazon ECS サービスデプロイを使用してサービス履歴を表示する」を参照してください。

サービスのデプロイコントローラーの更新の詳細については、「Amazon ECS サービスのパラメータの更新」を参照してください。

手順

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
2. [クラスター] ページで、クラスターを選択します。

クラスター詳細のページが表示されます。

3. [サービス] タブでサービスを選択します。

   [サービスの詳細] ページが表示されます。

4. バナーで [移行] を選択します。

   [デプロイ設定の更新] ページが表示されます。

5. [デプロイオプション] を展開し、次のパラメータを指定します。

   a. [デプロイコントローラータイプ] では [ECS] を選択します。

   b. [デプロイ戦略] には、[ローリング更新] を選択します。

   c. [Min running tasks] (実行中のタスクの最小化) の場合、デプロイ時に RUNNING の状態に保つ必要のあるサービス内のタスクの下限数をタスクの必要数のパーセント値 (最も近い整数に切り上げ) で入力します。詳細については、[Deployment configuration] (デプロイ設定) を参照してください。

   d. [Max running tasks] (実行中のタスクの最大化)) には、デプロイ時に RUNNING または PENDING 状態にできるサービスのタスクの上限数を必要数のタスクのパーセント値 (最も近い整数に切り下げ) で入力します。

6. [ロードバランシング] を展開し、次の内容で設定します。

   a. [ロール] には、ブルー/グリーンのアクセス許可を持つ前提条件で作成したロールを選択します。

      詳細については、「Amazon ECS ブルー/グリーンデプロイの Lambda 関数に必要となるアクセス許可」を参照してください。

   b. [リスナー] には、CodeDeploy ブルー/グリーンデプロイから本番リスナーを選択します。

   c. [ターゲットグループ] には、CodeDeploy ブルー/グリーンデプロイから本番ターゲットグループを選択します。

7. [更新] を選択します。

次のステップ

変更を有効にするには、サービスを更新する必要があります。詳細については、「Amazon ECS サービスを更新する」を参照してください。

Amazon ECS デプロイ戦略を更新します

Amazon ECS は、サービスを更新するために複数のデプロイ戦略をサポートします。アプリケーションの要件に基づいて、これらの戦略の間で移行できます。このトピックでは、ローリングデプロイおよびブルー/グリーンデプロイの間で移行する方法について説明します。

Amazon ECS デプロイ戦略の概要

デプロイ戦略の間で移行する前に、各戦略の仕組みおよび主な違いを理解することが重要です。

ローリングデプロイ

ローリングデプロイでは、Amazon ECS は現在実行中のアプリケーションのバージョンを新しいバージョンに置き換えます。サービススケジューラは最小および最大の正常率パラメータを使用し、デプロイ戦略を判断します。

ローリングデプロイはセットアップがより簡単ですが、デプロイプロセスおよびトラフィックルーティングの制御性は低下します。

ブルー/グリーンデプロイ

ブルー/グリーンデプロイでは、既存のバージョン (ブルー) とともに、Amazon ECS はサービスの新しいバージョン (グリーン) を作成します。新しいバージョンに本番トラフィックをルーティングする前に、検証できるようになります。

ブルー/グリーンデプロイでは、デプロイプロセスを詳細に制御できます (トラフィックの移行、テスト、ロールバック機能など)。

ベストプラクティス

デプロイ戦略の間で移行する際、次のベストプラクティスに従ってください。

- 非本番環境でテスト: 本番サービスに変更を適用する前に、常に非本番環境で更新をテストしてください。
- ロールバックの計画: 更新が期待どおりに機能しない場合に備えて、ロールバック計画を立てます。
- 移行中のモニタリング: 移行中および移行後にサービスを注意深くモニタリングし、正しく動作し続けることを確認します。
- ドキュメントの更新: デプロイドキュメントを更新して、新しいデプロイ戦略を反映します。

- トラフィックへの影響の考慮: 更新がサービスへのトラフィックに与える影響を理解し、影響内容に応じて計画します。

ローリング更新から Amazon ECS ブルー/グリーンへのデプロイ戦略の更新

本番環境で実装する前にサービスを変更してテストする際、ローリング更新デプロイから Amazon ECS ブルー/グリーンデプロイに移行できます。

前提条件

サービスをローリングデプロイからブルー/グリーンデプロイに移行する前に、次の内容があることを確認してください。

- 現在のデプロイがすべて完了するまで待ちます。
- ローリングデプロイ戦略を使用する既存の Amazon ECS サービス。
- トラフィックを処理する複数のサービスリビジョンがある場合、Amazon ECS は移行中にトラフィックを 1 つのリビジョンに統合しようとします。これが失敗した場合、移行前に 1 つのリビジョンを使用するように、手動でサービスの更新が必要な場合があります。
- 適切なアクセス許可を設定します。
  - Elastic Load Balancing のアクセス許可の詳細については、「ロードバランサー用の Amazon ECS インフラストラクチャの IAM ロール」を参照してください。
  - Lambda のアクセス許可の詳細については、「Amazon ECS ブルー/グリーンデプロイの Lambda 関数に必要となるアクセス許可」を参照してください。
- 設定内容に応じて、次のいずれかを実行する必要があります。
  - サービスが Elastic Load Balancing を使用している場合、サービスを新しい「advancedConfiguration」で更新してローリングデプロイを開始します。
  - サービスが Service Connect を使用している場合、サービスを更新してローリングデプロイを開始します。
  - サービスが Elastic Load Balancing および Service Connect の両方を使用している場合、上記のステップを両方とも実行します (1 つの UpdateService リクエストを使用できます)。
  - サービスがこれらのいずれも使用していない場合、追加のオペレーションは不要です。
- Amazon ECS ブルー/グリーンデプロイでは、サービスが次のいずれかの機能を使用することが必須です。適切なリソースを設定します。
  - Application Load Balancer – 詳細については、「ブルー/グリーンデプロイ、リニアデプロイおよびカナリアデプロイ用の Application Load Balancer リソース」を参照してください。

- Network Load Balancer – 詳細については、「Amazon ECS ブルー/グリーン、リニアおよびカナリアデプロイの Network Load Balancer リソース」を参照してください。
- Service Connect – 詳細については、「Amazon ECS ブルー/グリーンデプロイ、リニアデプロイおよびカナリアデプロイ用の Service Connect リソース」を参照してください。

手順

1. https://console.aws.amazon.com/ecs/v2 で Amazon ECS コンソールを開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。
3. [クラスター] ページで、移行するサービスを含むクラスターを選択します。

   クラスター詳細のページが表示されます。
4. [クラスターの詳細] ページで、[サービス] タブを選択します。
5. サービスを選択したら、[更新] を選択します。

   更新のサービスページが表示されます
6. [デプロイオプション] を展開し、次の操作を行います。
7. [デプロイ戦略] では、[ブルー/グリーン] を選択します。
8. ブルー/グリーンデプロイ設定を構成します。

   a. ベイク時間には、ブルーリビジョンが終了する前に、ブルーおよびグリーンのサービスリビジョンの両方が同時に実行される分数を入力します。

      これにより、検証およびテストの時間を確保できます。

   b. (オプション) デプロイの特定段階で実行されるように、Lambda 関数を設定します。[デプロイライフサイクルフック] で、次のステージのために Lambda 関数を設定します。

      - スケールアップ前: グリーンサービスリビジョンをスケールアップする前に実行されます
      - スケールアップ後: グリーンサービスリビジョンをスケールアップした後に実行されます
      - テストトラフィックの移行: グリーンサービスリビジョンへのテストトラフィックのルーティング中に実行されます
      - テストトラフィック後の移行: テストトラフィックがグリーンサービスリビジョンにルーティングされた後に実行されます
      - 本番トラフィックの移行: グリーンサービスリビジョンへの本番トラフィックルーティング中に実行されます

- 本番トラフィック後の移行: 本番トラフィックがグリーンサービスリビジョンにルーティングされた後に実行されます

ライフサイクルフックを追加する方法

a. [Add] (追加) を選択します。

b. [Lambda 関数] には、ARN の関数名を入力します。

c. [ロール] には、Lambda 関数を呼び出すアクセス許可を持つ IAM ロールを選択します。

d. [ライフサイクルステージ] では、Lambda 関数を実行するステージを選択します。

e. オプション: [フックの詳細] には、キーおよび値のペアを入力してフックに追加情報を提示します。

9. ロードバランサーを設定します。

a. [ロードバランシング] で、サービスがロードバランサーを使用するように設定されていることを確認します。

b. [ターゲットグループ] では、本番 (青) 環境のプライマリターゲットグループを選択します。

c. [代替ターゲットグループ] では、テスト (緑) 環境のターゲットグループを選択します。

d. 本番のリスナールールでは、本番トラフィックをルーティングするリスナールールを選択します。

e. オプション: [テストのリスナールール]では、テストトラフィックをグリーン環境にルーティングするリスナールールを選択します。

f. [ロール] では、Amazon ECS にロードバランサーの管理を可能にする IAM ロールを選択します。

10. 変更内容を確認し、[更新] を選択します。

次のステップ

- サービスを更新してデプロイを開始します。詳細については、「Amazon ECS サービスを更新する」を参照してください。
- デプロイプロセスをモニタリングし、ブルー/グリーンのパターンに従っていることを確認します。
  - グリーンサービスリビジョンが作成され、スケールアップされる
  - テストトラフィックがグリーンリビジョンにルーティングされている (設定されている場合)

- 本番トラフィックがグリーンリビジョンに移行されている
- ベイク時間が過ぎたら、ブルーリビジョンが終了する

デプロイ戦略を Amazon ECS ブルー/グリーンからローリング更新への更新

ブルー/グリーンデプロイをローリング更新デプロイに移行できます。

ローリングデプロイに移行する際には、次の点を考慮してください。

- トラフィック処理: ローリングデプロイを使用すると、新しいタスクがヘルスチェックに合格したら、すぐにトラフィックの受信が開始されます。ブルー/グリーンデプロイと同様に、個別のテストフェーズはありません。
- リソース効率: 通常、ローリングデプロイはブルー/グリーンデプロイよりも少ないリソースを使用します。これは、ローリングデプロイは完全に重複する環境を作成するのではなく、タスクを段階的に置き換えるからです。
- ロールバックの複雑さ: ローリングデプロイでは、ブルー/グリーンデプロイと比較してロールバックが複雑になります。ロールバックする必要がある場合、前のタスク定義で新しいデプロイを開始する必要があります。
- デプロイの速度: 特にタスクが多いサービスの場合、ローリングデプロイはブルー/グリーンデプロイよりも完了に時間がかかることがあります。
- ロードバランサー設定: 既存のロードバランサー設定はローリングデプロイでも引き続き機能しますが、トラフィックの移行動作は異なります。

前提条件

ブルー/グリーンからローリングデプロイにサービスを移行する前に、以下があることを確認してください。

- ブルー/グリーンデプロイ戦略を使用する既存の Amazon ECS サービス
- 継続中のサービスのデプロイがないこと (現在のデプロイが完了するまで待つ)
- ローリングデプロイでサービスが動作する方法を明確に理解する

> **Note**
>
> デプロイが進行中の場合、サービスをローリングデプロイに移行することはできません。続行する前に、現在のデプロイが完了するまで待ちます。

移行手順

Amazon ECS サービスをブルー/グリーンからローリングデプロイに移行するには、次の手順に従います。

1. https://console.aws.amazon.com/ecs/v2 で Amazon ECS コンソールを開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。
3. [クラスター] ページで、移行するサービスを含むクラスターを選択します。
4. [クラスターの詳細] ページで、[サービス] タブを選択します。
5. 移行するサービスを選択したら、[更新] を選択します。
6. [サービスの更新] ページで、[デプロイオプション] セクションに移動し、必要に応じて展開します。
7. [デプロイ戦略] には、[ローリング更新] を選択します。
8. ローリングデプロイ設定を構成します。
   a. [最小正常率] には、デプロイ中に RUNNING 状態を維持する必要があるタスクの最小パーセント値を入力します。この値は、サービスに望ましいタスク数に対するパーセンテージとして指定されます。
   b. [最大パーセント] には、デプロイ中に RUNNING または PENDING の状態で許可されるタスクの最大パーセント値を入力します。この値は、サービスに望ましいタスク数に対するパーセンテージとして指定されます。
9. オプション: [デプロイ失敗の検出] で、Amazon ECS がデプロイ失敗を検出して処理する方法を設定します。
   a. 展開サーキットブレーカーを有効にするには、[展開サーキットブレーカー] を選択します。
   b. 失敗したデプロイを自動的にロールバックするには、[失敗時にロールバック] を選択します。
10. 設定の変更を確認したら、[更新] を選択して変更を保存し、サービスをローリングデプロイに移行します。

Amazon ECS はサービス設定を更新し、ローリングデプロイ戦略を使用します。次回サービスを更新すると、ローリングデプロイプロセスが使用されます。

> **Note**
>
> ブルー/グリーンからローリングデプロイに移行すると、Amazon ECS は次の方法で移行を処理します。
>
> 1. トラフィックを処理している現在のアクティブなサービスリビジョンを特定します。
> 2. 既存のロードバランサー設定を維持しながら、新しいデプロイを処理する方法を変更します。
> 3. 将来のローリングデプロイに備えてサービスを準備します。

次のステップ

- サービスを更新してデプロイを開始します。詳細については、「Amazon ECS サービスを更新する」を参照してください。

Amazon ECS デプロイ戦略の更新に関するトラブルシューティング

このセクションでは、デプロイ戦略の移行時に発生する可能性のあるよくある問題に対する解決策を紹介します。

複数のサービスリビジョンまたはタスクセット

次の問題は、デプロイに複数のサービスリビジョンを持つことに関連しています。

ECS デプロイコントローラーを更新するときに複数のタスクセットがある

エラーメッセージ: `Updating the deployment controller is not supported when there are multiple tasksets in the service. Please ensure your service has only one taskset and try again.`

解決策: このエラーは、複数のアクティブなタスクセットを持つサービスのデプロイコントローラータイプを変更しようとすると発生します。`CODE_DEPLOY` または `EXTERNAL` デプロイコントローラーでこの問題を解決する方法:

1. 現在のタスクセットを確認します。

```
aws ecs describe-services --cluster your-cluster-name --services your-service-name --query "services[0].taskSets"
```

2. 進行中のデプロイが完了するまで待ちます。
3. 新しいデプロイがタスクセットをクリーンアップするように強制します。

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --force-new-deployment
```

4. 必要に応じて、他のタスクセットを手動で削除します。

```
aws ecs delete-task-set --cluster your-cluster-name --service your-service-name
 --task-set task-set-id
```

5. 残るタスクセットが 1 つだけになったら、デプロイコントローラーの更新を再試行してください。

詳細については、「the section called “サービスデプロイコントローラーと戦略”」を参照してください。

ECS デプロイコントローラーの更新時にプライマリタスクセットが不足している

エラーメッセージ: `Updating the deployment controller requires a primary taskset in the service. Please ensure your service has a primary taskset and try again.`

解決策: このエラーは、プライマリタスクが設定されていないサービスのデプロイコントローラータイプを変更しようとすると発生します。この問題を解決するには。

1. サービスのステータスとタスクセットを確認します。)。サービスにタスクセットが存在する場は、ACTIVE とマークする必要があります。

```
aws ecs describe-services --cluster your-cluster-name --services your-service-name --query "services[0].taskSets[*].[status,id]
```

ACTIVE 状態のタスクセットがない場合、デプロイを移行します。詳細については、「移行の方法」を参照してください。

2. サービスに実行中のタスクがない場合、サービスを更新して少なくとも 1 つのタスクをデプロイしてください。

```
aws ecs update-service-primary-task-set --cluster your-cluster-name --
service your-service-name --primary-task-set your-taskset-id
```

サービス内の (以前は ACTIVE) タスクセットが PRIMARY ステータスとしてマークされます。

3. タスクが安定した実行状態になるまで待ちます。ステータスは次の方法で確認できます。

```
aws ecs describe-services --cluster your-cluster-name --services your-service-
name --query "services[0].deployments"
```

4. 実行中のタスクがサービスのプライマリタスクに設定されたら、デプロイコントローラーの更新を再試行してください。

詳細については、「the section called “サービスデプロイコントローラーと戦略”」を参照してください。

デプロイ障害の検出タイプおよびデプロイコントローラーの不一致

次の問題は、デプロイ障害の検出タイプおよびデプロイコントローラーの不一致に関連しています。

非 ECS コントローラーを持ったデプロイサーキットブレーカー

エラーメッセージ: `Deployment circuit breaker feature is only supported with ECS deployment controller. Update to ECS deployment controller and try again.`

解決策: このエラーは、ECS デプロイコントローラーを使用していないサービスに対して、デプロイサーキットブレーカー機能を有効にしようとすると発生します。デプロイサーキットブレーカーは、ECS デプロイコントローラーとのみ互換性があります。

1. サービスの現在のデプロイコントローラーを確認してください。

```
aws ecs describe-services --cluster your-cluster-name --services your-service-
name --query "services[0].deploymentController"
```

2. ECS デプロイコントローラーを使用するように、サービスを更新します。

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --deployment-controller type=ECS
```

3. サービスが ECS デプロイコントローラーを使用したら、デプロイサーキットブレーカーを有効にします。

```
aws ecs update-service --cluster your-cluster-name --
service your-service-name --deployment-configuration
 "deploymentCircuitBreaker={enable=true,rollback=true}"
```

詳細については、「the section called “ デプロイサーキットブレーカーが障害を検出する方法”」を参照してください。

非 ECS コントローラーのアラームベースのロールバック

エラーメッセージ: `Alarm based rollback feature is only supported with ECS deployment controller. Update to ECS deployment controller and try again.`

解決策: このエラーは、ECS デプロイコントローラーを使用していないサービスでアラームベースのロールバックを設定しようとすると発生します。アラームベースのロールバック機能は、ECS デプロイコントローラーとのみ互換性があります。

1. サービスの現在のデプロイコントローラーを確認してください。

```
aws ecs describe-services --cluster your-cluster-name --services your-service-
name --query "services[0].deploymentController"
```

2. ECS デプロイコントローラーを使用するように、サービスを更新します。

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --deployment-controller type=ECS
```

3. サービスが ECS デプロイコントローラーを使用したら、アラームベースのロールバックを設定してください。

```
aws ecs update-service --cluster your-cluster-name --services your-
service-name --deployment-configuration "alarms={alarmNames=[your-alarm-
name],enable=true,rollback=true}"
```

詳細については、「the section called “CloudWatch アラームがデプロイ障害を検出する方法”」を参照してください。

Service Connect およびデプロイコントローラーの不一致

次の問題は、Service Connect およびデプロイコントローラーの不一致に関連しています。

Service Connect を使用した EXTERNAL コントローラー

エラーメッセージ: `The EXTERNAL deployment controller type is not supported for services using Service Connect.`

解決策: このエラーは、Service Connect が有効になっているサービスで EXTERNAL デプロイコントローラーを使用しようとすると発生します。EXTERNAL コントローラーは Service Connect と互換性がありません。

1. サービスで Service Connect が有効になっているかどうかを確認してください。

```
aws ecs describe-services --cluster your-cluster-name --services your-service-name --query "services[0].serviceConnectConfiguration"
```

2. EXTERNAL デプロイコントローラーを使用する必要がある場合、サービスを更新して Service Connect を無効にします。

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --service-connect-configuration "{}"
```

3. または、Service Connect を使用する必要がある場合は、代わりに ECS デプロイコントローラーを使用します。

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --deployment-controller type=ECS
```

詳細については、「the section called “サービスデプロイコントローラーと戦略”」を参照してください。

非 ECS コントローラーを使用した Service Connect

エラーメッセージ: `Service Connect feature is only supported with ECS (rolling update) deployment controller. Update to ECS deployment controller and try again.`

解決策: このエラーは、ECS デプロイコントローラーを使用していない Service Connect を有効にしようとすると発生します。Service Connect 機能は、ECS デプロイコントローラーとのみ互換性があります。

1. サービスの現在のデプロイコントローラーを確認してください。

```
aws ecs describe-services --cluster your-cluster-name --services your-service-name --query "services[0].deploymentController"
```

2. ECS デプロイコントローラーを使用するように、サービスを更新してください。

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --deployment-controller type=ECS
```

3. サービスが ECS デプロイコントローラーを使用したら、Service Connect を有効にしてください。

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --service-connect-configuration "enabled=true,namespace=your-namespace"
```

詳細については、「the section called “サービスデプロイコントローラーと戦略”」を参照してください。

コントローラータイプおよびスケジューリング戦略の不一致

次の問題は、コントローラータイプおよびスケジューリング戦略の不一致に関連しています。

DAEMON スケジューリング戦略を使用する CODE_DEPLOY コントローラー

エラーメッセージ: `The CODE_DEPLOY deployment controller type is not supported for services using the DAEMON scheduling strategy.`

解決策: このエラーは、DAEMON スケジューリング戦略を使用するサービスで CODE_DEPLOY デプロイコントローラーを使用しようとすると発生します。CODE_DEPLOY コントローラーは、REPLICA スケジューリング戦略とのみ互換性があります。

1. サービスの現在のスケジューリング戦略を確認してください。

```
aws ecs describe-services --cluster your-cluster-name --services your-service-name --query "services[0].schedulingStrategy"
```

2. ブルー/グリーンデプロイが必要な場合、REPLICA スケジューリング戦略を使用するようにサービスを変更します。

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --scheduling-strategy REPLICA
```

3. または、DAEMON スケジューリング戦略を使用する必要がある場合は、代わりに ECS デプロイコントローラーを使用します。

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --deployment-controller type=ECS
```

詳細については、「the section called “サービスデプロイコントローラーと戦略”」を参照してください。

デーモンスケジューリング戦略を使用した EXTERNAL コントローラー

エラーメッセージ: `The EXTERNAL deployment controller type is not supported for services using the DAEMON scheduling strategy.`

解決策: このエラーは、デーモンスケジュール戦略を使用する ECS サービスで EXTERNAL デプロイコントローラーを使用しようとすると発生します。EXTERNAL コントローラーは、REPLICA スケジューリング戦略とのみ互換性があります。

1. サービスの現在のスケジューリング戦略を確認してください。

```
aws ecs describe-services --cluster your-cluster-name --services your-service-name --query "services[0].schedulingStrategy"
```

2. EXTERNAL デプロイコントローラーを使用する必要がある場合は、REPLICA スケジューリング戦略を使用するようにサービスを変更します。

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --scheduling-strategy REPLICA
```

3. または、DAEMON スケジューリング戦略を使用する必要がある場合は、代わりに ECS デプロイコントローラーを使用します。

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --deployment-controller type=ECS
```

詳細については、「the section called “サービスデプロイコントローラーと戦略”」を参照してください。

外部起動タイプを使用したサービスレジストリ

エラーメッセージ: `Service registries are not supported for external launch type.`

解決策: このエラーは、EXTERNAL 起動タイプを使用するサービスにサービス検出 (サービスレジストリ) を設定しようとすると発生します。サービス検出は、EXTERNAL 起動タイプと互換性がありません。

1. サービスの現在の起動タイプを確認してください。

```
aws ecs describe-services --cluster your-cluster-name --services your-service-
name --query "services[0].launchType"
```

2. サービス検出が必要な場合は、EC2 または FARGATE の起動タイプのいずれかを使用するようにサービスを変更します。

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --launch-type FARGATE
```

3. または、EXTERNAL 起動タイプを使用する必要がある場合は、サービスレジストリ設定を削除します。

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --service-registries "[]"
```

詳細については、「the section called “サービスデプロイコントローラーと戦略”」を参照してください。

デプロイコントローラーの更新を元に戻します

前のデプロイコントローラーに戻す場合、次のいずれかを実行できます。

- CloudFormation を使用した場合、前のテンプレートを使用して新しいスタックを作成できます。詳細については、「CloudFormation ユーザーガイド」の「スタック作成」を参照してください。
- Amazon ECS コンソールまたは AWS CLI を使用した場合、サービスを更新できます。詳細については、「the section called “サービスの更新”」を参照してください。

  update-service コマンドを使用する場合、`--deployment-controller` オプションを使用して前のデプロイコントローラーに設定します。

Amazon ECS サービスデプロイを使用してサービス履歴を表示する

サービスデプロイは、デプロイの包括的なビューを提供します。サービスデプロイでは、サービスに関する次の情報が提供されます。

- 現在デプロイされているワークロード設定 (ソースサービスリビジョン)
- デプロイ中のワークロード設定 (ターゲットサービスリビジョン)
- デプロイステータス
- サーキットブレークが検出された失敗したタスク数
- アラーム状態になっている CloudWatch アラーム
- サービスデプロイが開始および完了した日時
- ロールバックの詳細 (発生した場合)

サービスデプロイプロパティの詳細については、「Amazon ECS サービスデプロイに含まれるプロパティ」を参照してください。

サービスデプロイは読み取り専用で、それぞれに一意の ID があります。

サービスデプロイには 3 つのステージがあります。

ステージ	定義	関連付けられた状態
保留中	サービスデプロイが作成されたが、開始されていない	保留中
継続的	サービスデプロイが進行中	• IN_PROGRESS • STOP_REQUESTED • ROLLBACK_REQUESTED (ロールバックのリクエスト済み) • ROLLBACK_IN_PROGRESS
完了	サービスデプロイが完了 (成功または失敗)	• SUCCESSFUL • 停止 • ROLLBACK_SUCCESSFUL • ROLLBACK_FAILED

サービスのライフサイクルを理解し、実行が必要なアクションがあるかどうかを判断するため、サービスデプロイを使用します。例えば、ロールバックが発生した場合は、サービスデプロイを調査し、サービスイベントを確認する必要がある場合があります。

コンソール、API、および AWS CLI を使用して、2024 年 10 月 25 日以降に作成されたデプロイの最新の 90 日間の履歴を表示できます。

完了していないデプロイを停止できます。詳細については、「Amazon ECS サービスデプロイの停止」を参照してください。

デプロイが早期成功基準を使用している場合、`DescribeServiceDeployments` は設定された早期成功基準を返し、実行中のタスク数は、ライブカウントではなく、デプロイが完了したときのスナップショットです。`ListServiceDeployments` でフィルタリングする際にステータスが `SUCCESSFUL` の場合、早期に完了したデプロイが含まれます。詳細については、「早期成功基準を使用した Amazon ECS ローリングデプロイの早期完了」を参照してください。

サービスデプロイのライフサイクル

Amazon ECS は、次のいずれかのアクションが発生すると、新しいサービスデプロイを自動的に作成します。

- ユーザーがサービスを作成した。
- ユーザーがサービスを更新し、[新しいデプロイの強制] オプションを使用した。
- ユーザーが、デプロイを必要とする 1 つ以上のサービスプロパティを更新した。

Amazon ECS は、デプロイの実行時にサービスデプロイの進行状況を反映するために次のサービスデプロイプロパティを更新します。

- ステータス
- 実行中のタスク数

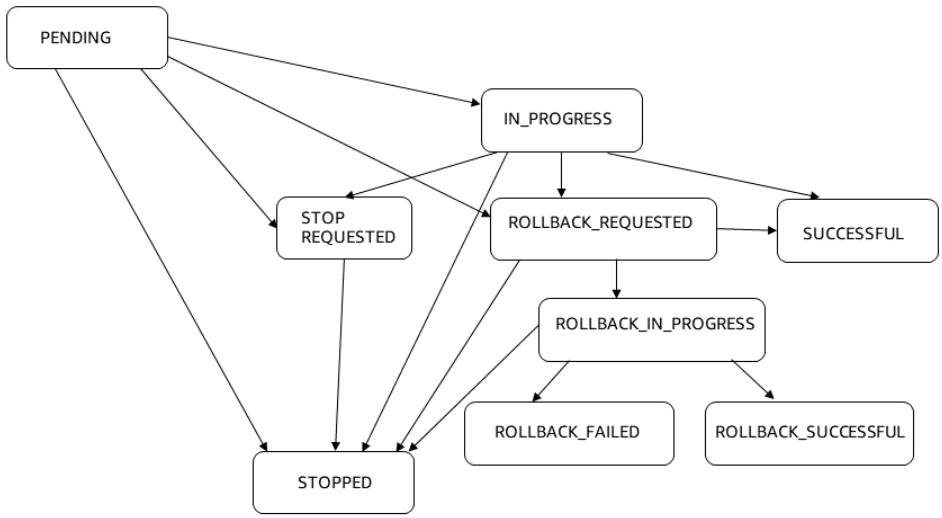
  サービスリビジョンに示されている実行中のタスク数が、実際に実行されているタスク数と等しくない場合があります。この数は、デプロイが完了したときに実行されているタスク数を表します。例えば、サービスデプロイとは関係なくタスクを起動した場合、それらのタスクはサービスリビジョンの実行タスク数に含まれません。
- サーキットブレーカーの障害検出:
  - 開始に失敗したタスク数
- CloudWatch アラームの障害検出
  - アクティブなアラーム
- ロールバック情報:
  - 開始時刻
  - ロールバックの理由
  - ロールバックに使用されるサービスリビジョンの ARN
- ステータスの理由

サービスを削除すると、Amazon ECS はサービスデプロイを削除します。

サービスデプロイの状態

サービスデプロイは `PENDING` 状態で開始されます。

次の図は、PENDING 状態の後に発生する可能性のあるサービスデプロイの状態を示します: IN_PROGRESS、ROLLBACK_REQUESTED、SUCCESSFUL、STOP_REQUESTED、ROLLBACK_IN_PROGRESS



以下の情報は、サービスデプロイの状態に関する詳細を提供します。

- PENDING - サービスデプロイは作成されましたが、開始されていません。

  状態は、そこから IN_PROGRESS、ROLLBACK_REQUESTED、STOP_REQUESTED、STOPPED のいずれかに移行します。

- IN_PROGRESS - サービスデプロイが進行中です。

  状態は、そこから SUCCESSFUL、STOP_REQUESTED、ROLLBACK_REQUESTED、ROLLBACK_IN_PROGRESS、STOPPED のいずれかに移行します。

- STOP_REQUESTED - 次のいずれかが発生すると、サービスデプロイの状態は STOP_REQUESTED に移行します。
  - ユーザーが新しいサービスデプロイを開始した。
  - ロールバックオプションが障害検出メカニズム (サーキットブレーカーまたはアラームベース) に使用されていないため、サービスが SUCCESSFUL 状態に達しない。

状態は STOPPED に移行します。

- ROLLBACK_REQUESTED – ユーザーがコンソール、API、または CLI を介してロールバックをリクエストすると、サービスのデプロイ状態は ROLLBACK_REQUESTED に移行します。

  状態は、そこから SUCCESSFUL、ROLLBACK_IN_PROGRESS、STOPPED のいずれかに移行します。

- SUCCESSFUL - サービスデプロイが正常に完了すると、サービスデプロイの状態は SUCCESSFUL に移行します。

- ROLLBACK_IN_PROGRESS - ロールバックオプションが障害検出メカニズム (サーキットブレーカーまたはアラームベース) に使用されていて、サービスが失敗すると、サービスデプロイの状態は ROLLBACK_IN_PROGRESS に移行します。

  状態は ROLLBACK_SUCCESSFUL または ROLLBACK_FAILED に移行します。

Amazon ECS サービスデプロイに含まれるプロパティ

サービスデプロイには、次のプロパティが含まれます。

プロパティ	説明	
サービスデプロイ ARN	サービスデプロイの ARN。	
サービス ARN	このサービスデプロイのサービスの ARN。	
クラスター ARN	サービスをホストするクラスターの ARN。	
サービスデプロイの作成時刻	サービスデプロイが作成された時刻。	
サービスデプロイの開始時刻	サービスデプロイが開始された時刻。	
サービスデプロイの終了時刻	サービスデプロイが終了した時刻。	

プロパティ	説明
サービスデプロイの停止時刻	サービスデプロイが停止した時刻。
サービスデプロイの更新時刻	サービスデプロイが最後に更新された時刻。
ソースサービスリビジョン	現在実行中のサービスリビジョン。 含まれるプロパティについては、「Amazon ECS サービスリビジョンに含まれるプロパティ」を参照してください。
デプロイ設定	サーキットブレーカー設定、決定するアラームを含むデプロイパラメータ。 • デプロイ中またはコンテナインスタンスのドレイン中に、サービスに対して実行する必要があるタスク数の下限 (サービスに必要なタスク数の割合)。 • デプロイ中またはコンテナインスタンスのドレイン中に、サービスに対して実行する必要があるタスク数の上限 (サービスに必要なタスク数の割合)。 • サーキットブレーカーの設定。 • 障害検出に使用されるアラーム。

プロパティ	説明
ターゲットサービスリビジョン	デプロイするサービスリビジョン。 デプロイが正常に完了すると、ターゲットサービスリビジョンは実行中のサービスリビジョンになります。
サービスデプロイの状態	サービスデプロイの状態。 有効な値は、PENDING、SUCCESSFUL、STOPPED、STOP_REQUESTED、STOP_IN_PROGRESS、IN_PROGRESS、ROLLBACK_IN_PROGRESS、ROLLBACK_SUCCESSFUL、および ROLLBACK_FAILED です。
サービスデプロイ状態の情報	サービスデプロイが現在の状態である理由に関する情報。例えば、サーキットブレーカーが障害を検出したなど。
ロールバック情報	デプロイが失敗したときにサービスデプロイが使用するロールバックオプション。
サービスデプロイサーキットブレーカーオプション	サービスデプロイが失敗したと判断するサーキットブレーカー。
サービスデプロイの CloudWatch アラーム	サービスデプロイがいつ失敗したかを判断する CloudWatch アラーム。

Amazon ECS サービスデプロイを表示するために必要なアクセス許可

最小権限のベストプラクティスに従うと、コンソールでサービスデプロイを表示するには、アクセス許可を追加する必要があります。

次のアクションへのアクセスが必要です。

- ListServiceDeployments
- DescribeServiceDeployments
- DescribeServiceRevisions

次のリソースへのアクセスが必要です。

- サービス
- サービスデプロイ
- サービスリビジョン

次のポリシー例には必要なアクセス許可が含まれており、アクションは指定されたサービスに限定されます。

`account`、`cluster-name`、および `service-name` を独自の値に置き換えます。

JSON

```
{
"Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "ecs:ListServiceDeployments",
            "ecs:DescribeServiceDeployments",
            "ecs:DescribeServiceRevisions"
        ],
        "Resource": [
            "arn:aws:ecs:us-east-1:123456789012:service/cluster-name/service-
name",
            "arn:aws:ecs:us-east-1:123456789012:service-deployment/cluster-name/
service-name/*",
```

```
            "arn:aws:ecs:us-east-1:123456789012:service-revision/cluster-name/
service-name/*"
            ]
        }
    ]
}
```

Amazon ECS サービスデプロイの表示

2024 年 10 月 25 日以降に作成されたデプロイの最新の 90 日間の履歴を確認できます。サービスデプロイは、次のいずれかの状態になります。

- 進行中
- 保留中
- 完了

この情報を使用して、サービスのデプロイ方法やサービスリビジョンを更新する必要があるかどうかを判断できます。含まれるプロパティについては、「Amazon ECS サービスデプロイに含まれるプロパティ」を参照してください。

開始する前に、サービスデプロイを表示するために必要なアクセス許可を設定します。詳細については、「Amazon ECS サービスデプロイを表示するために必要なアクセス許可」を参照してください。

Amazon ECS Console

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. [クラスター] ページで、クラスターを選択します。
3. [クラスターの詳細] ページの [サービス] セクションで、サービスを選択します。

   [サービスの詳細] ページが表示されます。
4. [サービスの詳細] ページで、[デプロイ] を選択します。
5. 表示するサービスデプロイを選択します。

このデプロイタイプのサービスデプロイを表示するには	この操作を行います
進行中のデプロイ	[進行中のデプロイ] で、[デプロイ ID] を選択します。
最後のデプロイ	[最後のデプロイ] で、[デプロイ ID] を選択します。
完了したデプロイ	[サービスデプロイ] で、[デプロイ ID] を選択します。

[サービスデプロイの詳細] ページが表示されます。

6. (オプション) サービスリビジョンを比較して、相違点を確認します。

   [サービスリビジョン] で、[リビジョンの比較] を選択し、比較するリビジョンを 2 つ選択します。

   サービスリビジョンは並行して表示され、相違点が強調表示されます。

AWS CLI

1. `list-service-deployments` を実行して、サービスデプロイ ARN を取得します。

   変数を実際の値に置き換えます。

   ```
   aws ecs list-service-deployments --cluster cluster-name --service service-name
   ```

   表示するデプロイの serviceDeploymentArn を書き留めます。

   ```
   {
       "serviceDeployments": [
           {
               "serviceDeploymentArn": "arn:aws:ecs:us-west-2:123456789012:service-
   deployment/example/sd-example/NCWGC2ZR-taawPAYrIaU5",
   ```

```
            "serviceArn": "arn:aws:ecs:us-west-2:123456789012:service/example/
sd-example",
            "clusterArn": "arn:aws:ecs:us-west-2:123456789012:cluster/example",
            "targetServiceRevisionArn": "arn:aws:ecs:us-
west-2:123456789012:service-revision/example/sd-example/4980306466373577095",
            "status": "SUCCESSFUL"
        }
    ]
}
```

2. describe-service-deployments を実行します。list-service-deployments から返された serviceDeploymentArn を使用します。

   変数を実際の値に置き換えます。

```
aws ecs describe-service-deployments --service-deployment-arns
 arn:aws:ecs:region:123456789012:service-deployment/cluster-name/service-
name/NCWGC2ZR-taawPAYrIaU5
```

次のステップ

デプロイのサービスリビジョンの詳細を表示できます。詳細については、Amazon ECS サービスリビジョンの詳細の表示を参照してください。

Amazon ECS サービスリビジョン

サービスリビジョンには、Amazon ECS がデプロイしようとしているワークロード設定のレコードが含まれています。サービスを作成またはデプロイするたびに、Amazon ECS はサービスリビジョンでデプロイしようとしている設定を自動的に作成してキャプチャします。

サービスリビジョンは読み取り専用で、一意の識別子があります。含まれるプロパティについては、「Amazon ECS サービスリビジョンに含まれるプロパティ」を参照してください。

サービスリビジョンには次の利点があります。

- サービスデプロイ中に、現在デプロイされているサービスリビジョン (ソース) とデプロイ中のサービスリビジョン (ターゲット) を比較できます。
- サービスデプロイにロールバックオプションを使用すると、Amazon ECS はサービスデプロイを最後に正常にデプロイされたサービスリビジョンに自動的にロールバックします。
- サービスリビジョンには、1 つのリソースにおけるワークロード設定のレコードが含まれます。

サービスリビジョンのライフサイクル

Amazon ECS は、サービスの作成時、またはデプロイを開始するサービスプロパティの更新時に、新しいサービスリビジョンを自動的に作成します。

Amazon ECS は、ロールバックオペレーションの新しいサービスリビジョンを作成しません。Amazon ECS は、最後に成功したサービスリビジョンをロールバックに使用します。

サービスリビジョンはイミュータブルです。

サービスを削除すると、Amazon ECS はサービスリビジョンを削除します。

2024 年 10 月 25 日以降に作成されたサービスリビジョンは、コンソール、API、および CLI を使用して表示できます。

Amazon ECS サービスリビジョンに含まれるプロパティ

サービスリビジョンには、次のプロパティが含まれます。

リソース	説明
サービス ARN	クライアントを識別する ARN。
クラスター ARN	サービスをホストするクラスターの ARN。
タスク定義 ARN	サービスタスクに使用されるタスク定義の ARN。
サービスレジストリ	サービス検出に使用されるサービスレジストリの詳細。 • コンテナ名 サービス検出サービスに使用されるコンテナ名の値。 • コンテナポート サービス検出サービスに使用されるポートの値。この

リソース	説明	
	値は、タスク定義でも指定されます。 • ポート サービス検出サービスが SRV レコードを指定した場合に使用されるポート値。 • レジストリ ARN サービスレジストリの Amazon リソースネーム (ARN)。	
キャパシティープロバイダー	キャパシティプロバイダー戦略の詳細。 • キャパシティプロバイダーの名前 サービスに使用されるキャパシティプロバイダー名。 • キャパシティプロバイダーの重み 指定されたキャパシティプロバイダーを使用する必要がある起動済みタスクの総数の相対的な割合。 • キャパシティプロバイダーベース 指定されたキャパシティプロバイダーで実行する最小タスク数。	

リソース	説明
コンテナイメージ	コンテナイメージに関する詳細。 • コンテナ名 • イメージダイジェスト • コンテナイメージ
ネットワーク	サービスのネットワーク設定。 • パブリック IP アドレスがあるかどうか • タスクが実行されるサブネット • サービストラフィックを制御するセキュリティグループ
起動タイプ	サービスに使用されるコンピューティングオプション。

Fargate 固有のプロパティ	Fargate を使用する場合について、これは Fargate バージョンに関する情報です。
	• プラットフォームバージョンとプラットフォームファミリー
	• エフェメラルストレージ
	  • デプロイに割り当てるエフェメラルストレージの量。
	  • ストレージの暗号化に使用する KMS キー。

リソース	説明	
デプロイ時に設定される Amazon EBS ボリューム	起動時に設定されるボリュームとしてタスク定義で指定されたボリュームの設定。 • 暗号化された ボリュームが暗号化されているかどうかを示します。 • ファイルシステムタイプ ボリュームの Linux ファイルシステムタイプ。 • IOPS 1 秒あたりの I/O 操作の数 (IOPS)。 • KMS キー • IAM ロール このボリュームに関連付けられている IAM ロールの ARN。 • サイズ ボリュームのサイズ (GiB 単位)。 • スナップショット ID ボリュームスナップショット ID。 • タグ仕様 ボリュームのタグ設定。 • スループット	

リソース	説明
	ボリュームにプロビジョニングされたスループット。 • ボリュームタイプ ボリュームタイプ。
Service Connect	Service Connect の設定。 • Service Connect が使用中かどうかの表示 • 名前空間 • 相互接続されたサービスのリスト • Service Connect のログ記録設定
サービスロードバランサー	サービストラフィックをルーティングするロードバランサー。 • 名前 • コンテナ名 • コンテナポート • ターゲットグループ ARN
Runtime Monitoring	Runtime Monitoring がオンになっているかどうかを示します。
作成日	サービスリビジョンが作成された日付。
VPC Lattice	サービスリビジョンの VPC Lattice 設定。

リソース	説明	
モニタリング	サービスのモニタリング設定。サービスレベル CPUUtilization と MemoryUtilization CloudWatch メトリクスの解決を定義します。 • メトリクス名 収集するメトリクス (CPUUtilization 、MemoryUtilization)。 • 解像度 (秒) メトリクス収集の解像度。有効な値は、20 および 60 です。	

Amazon ECS サービスリビジョンの詳細の表示

2024 年 10 月 25 日以降に作成された以下のサービスリビジョンタイプに関する情報を表示できます。

- ソース - 現在デプロイされているワークロード設定
- ターゲット - デプロイ中のワークロード設定

Amazon ECS Console

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
2. [クラスター] ページで、クラスターを選択します。
3. [クラスターの詳細] ページの [サービス] セクションで、サービスを選択します。

   [サービスの詳細] ページが表示されます。
4. [サービスの詳細] ページで、[デプロイ] を選択します。

5. 表示するサービスリビジョンを選択します。

このデプロイタイプのサービスリビジョンを表示するには	この操作を行います
進行中のデプロイ	[進行中のデプロイ] で、次の操作を行います。 • ソースサービスリビジョンを表示するには、[サービスリビジョン] で、ソース [リビジョンタイプ] のサービスリビジョン ID を選択します。 • ターゲットサービスリビジョンを表示するには、[サービスリビジョン] で、ターゲット [リビジョンタイプ] のサービスリビジョン ID を選択します。
最後のデプロイ	[最後のデプロイ] で、[ターゲットサービスリビジョン] を選択します。
完了したデプロイ	[サービスデプロイ] で、次の操作を行います。 • [ターゲットサービスリビジョン] で、ID を選択します。

AWS CLI

1. describe-service-deployments を実行して、サービスリビジョン ARN を取得します。

   変数を実際の値に置き換えます。

   ```
   aws ecs describe-service-deployments --service-deployment-arns
    arn:aws:ecs:region:account-id:service/cluster-name/service-name/NCWGC2ZR-
   taawPAYrIaU5
   ```

   sourceServiceRevisions または targetServiceRevisions の arn を書き留めます。

   ```
   {
       "serviceDeployments": [
           {
               "serviceDeploymentArn": "arn:aws:ecs:us-west-2:123456789012:service-
   deployment/example/sd-example/NCWGC2ZR-taawPAYrIaU5",
               "serviceArn": "arn:aws:ecs:us-west-2:123456789012:service/example/
   sd-example",
               "clusterArn": "arn:aws:ecs:us-west-2:123456789012:cluster/example",
               "updatedAt": "2024-09-10T16:49:35.572000+00:00",
               "sourceServiceRevision": {
                   "arn": "arn:aws:ecs:us-west-2:123456789012:service-revision/
   example/sd-example/4980306466373578954",
                   "requestedTaskCount": 0,
                   "runningTaskCount": 0,
                   "pendingTaskCount": 0
               },
               "targetServiceRevision": {
                   "arn": "arn:aws:ecs:us-west-2:123456789012:service-revision/
   example/sd-example/4980306466373577095",
                   "requestedTaskCount": 0,
                   "runningTaskCount": 0,
                   "pendingTaskCount": 0
               },
               "status": "IN_PROGRESS",
               "deploymentConfiguration": {
                   "deploymentCircuitBreaker": {
                       "enable": false,
                       "rollback": false
   ```

```
                },
                "maximumPercent": 200,
                "minimumHealthyPercent": 100
            }
        }
    ],
    "failures": []
}
```

2. describe-service-revisions を実行します。describe-service-deployments から返された arn を使用します。

   変数を実際の値に置き換えます。

```
aws ecs describe-service-revisions --service-revision-arns
 arn:aws:ecs:region:123456789012:service-revision/cluster-name/service-
name/4980306466373577095
```

アベイラビリティーゾーン間での Amazon ECS サービスの調整

2025 年 9 月 5 日以降、Amazon ECS は、この機能の対象となるすべてのサービスについてアベイラビリティーゾーンの再調整を有効にします。アベイラビリティーゾーンの分散が第一優先のタスク配置戦略である場合、または配置戦略がない場合、サービスは対象になります。

アプリケーションの高可用性を実現するために、マルチタスクサービスを複数のアベイラビリティーゾーンで実行するように設定することをお勧めします。最初の配置戦略をアベイラビリティーゾーンに分散するように指定するサービスの場合、AWS は可能な限り、使用可能なアベイラビリティーゾーン間でサービスタスクを均等に分散します。

ただし、アベイラビリティーゾーンの中断後など、あるアベイラビリティーゾーンで実行されているタスク数が、他のアベイラビリティーゾーンで実行されているタスク数と同じではない場合があります。このタスクの不均衡に対処するには、アベイラビリティーゾーンの再調整機能を有効にします。

アベイラビリティーゾーンの再調整により、Amazon ECS は各サービスのアベイラビリティーゾーン間のタスクの分散を継続的にモニタリングします。Amazon ECS は、不均等なタスク分散を検出すると、アベイラビリティーゾーン間でワークロードを再調整するアクションを自動的に実行します。これには、タスクが最も少ないアベイラビリティーゾーンで新しいタスクを起動し、オーバーロードされたアベイラビリティーゾーンでタスクを終了することが含まれます。

この再分散により、単一のアベイラビリティーゾーンが障害点になることがなくなり、コンテナ化されたアプリケーションの全体的な可用性を維持できます。自動再調整プロセスにより、手動による介入が不要になり、イベント後の復旧までの時間が短縮されます。

アベイラビリティーゾーンの再調整プロセスの概要を次に示します。

1. Amazon ECS は、サービスが定常状態に達した後にモニタリングを開始し、各アベイラビリティーゾーンで実行されているタスク数を調べます。
2. Amazon ECS は、各アベイラビリティーゾーンで実行されているタスク数の不均衡を検出すると、次のオペレーションを実行します。
   - アベイラビリティーゾーンの再調整が開始されていることを示すサービスイベントを送信します。
   - 実行中のタスク数が最も少ないアベイラビリティーゾーンのタスクを開始します
   - 実行中のタスク数が最も多いアベイラビリティーゾーンのタスクを停止します。
   - スケジューラは、新しく開始されたタスクが `HEALTHY` および `RUNNING` になるのを待ってから、オーバースケーリングされたアベイラビリティーゾーンのタスクを停止します。
   - アベイラビリティーゾーンの再調整結果を含むサービスイベントを送信します。

不均等なタスク分散を Amazon ECS が検出する方法

サービスの必要なタスク数を設定されたアベイラビリティーゾーンの数で割ることで、Amazon ECS は各アベイラビリティーゾーンで実行されているタスク数の不均衡を判断します。必要なタスク数を割り切れない場合、Amazon ECS は残りのタスクを設定されたアベイラビリティーゾーン間に均等に分散します。各アベイラビリティーゾーンには、少なくとも 1 つのタスクが必要です。

例えば、2 つのアベイラビリティーゾーンに対して設定されたタスクの必要数が 2 つの Amazon ECS サービスを考えてみましょう。このシナリオでは、必要なタスクの数が均等に分割されます。均等な分散では、アベイラビリティーゾーンあたり 1 つのタスクになります。アベイラビリティーゾーン 1 に 2 つのタスクがあり、アベイラビリティーゾーン 2 に 0 のタスクがある場合、Amazon ECS はアベイラビリティーゾーン 1 でタスクを停止する前に、アベイラビリティーゾーン 2 でタスクを開始して再調整を開始します。

ここで、2 つのアベイラビリティーゾーンに対して設定されたタスクの必要数が 3 つの Amazon ECS サービスを考えてみましょう。このシナリオでは、必要なタスクの数は均等に分割されません。各アベイラビリティーゾーンに少なくとも 1 つのタスクがあり、残りのタスクはアベイラビリティーゾーン 2 に配置されているため、アベイラビリティーゾーン 1 で 1 つのタスク、アベイラビリティーゾーン 2 で 2 つのタスクがバランスが取れた分散です。

3 つのアベイラビリティーゾーンに対して設定されたタスクの必要数が 5 つの Amazon ECS サービスを考えてみましょう。このシナリオでは、必要なタスクの数は均等に分割されません。バランスが取れた分散は、アベイラビリティーゾーン 1 で 1 つのタスク、アベイラビリティーゾーン 2 と 3 でそれぞれ 2 つのタスクになります。各アベイラビリティーゾーンに 1 つのタスクがあることを考慮した後、残り 2 つのタスクはアベイラビリティーゾーン全体に均等に分散されます。

アベイラビリティーゾーンの再調整の設定に関する考慮事項

アベイラビリティーゾーンの再調整を設定したい場合は、次の点を考慮してください。

- アベイラビリティーゾーンの再調整は、Fargate および EC2 のキャパシティプロバイダーをサポートし、Amazon ECS マネージドインスタンスでサポートされています。Fargate の場合、Amazon ECS はバランスを維持するために、使用可能なアベイラビリティーゾーン間でタスクを自動的に再分散します。EC2 キャパシティプロバイダーの場合、Amazon ECS は、定義された配置に関する戦略と制約を考慮して、ベストエフォートベースで既存のコンテナインスタンス間でタスクを再調整します。ただし、Amazon ECS は再調整プロセスの一環として使用率の低いアベイラビリティーゾーンで新しいインスタンスを起動することができないため、再調整は既存のコンテナインスタンスに制限されます。
- アベイラビリティーゾーンの再調整は、次の設定で機能します。
  - `Replica` 戦略を使用するサービス
  - アベイラビリティーゾーン分散を最初のタスク配置戦略として指定するサービス、または配置戦略を指定しないサービス。
- アベイラビリティーゾーンの再調整は、次のいずれかの条件を満たすサービスでは使用できません。
  - `Daemon` 戦略を使用する
  - `EXTERNAL` 起動タイプ (ECS Anywhere) を使用する
  - `maximumPercent` 値に 100% を使用する
  - Classic Load Balancer を使用する
  - `attribute:ecs.availability-zone` をタスク配置の制約として使用する

アベイラビリティーゾーンの再調整による配置戦略と配置制約

配置戦略により、Amazon ECS がタスク配置を終了するためのコンテナインスタンスとアベイラビリティーゾーンを選択する方法が決まります。タスク配置の制約は、タスクを特定のコンテナインスタンスで実行できるかどうかを決定するルールです。

EC2 では、アベイラビリティーゾーンの再調整と組み合わせて配置戦略と配置制約を使用できます。ただし、アベイラビリティーゾーンの再調整が機能するには、アベイラビリティーゾーン分散配置戦略が最初に指定される戦略である必要があります。

アベイラビリティーゾーンの再調整は、さまざまな配置戦略の組み合わせと互換性があります。例えば、最初にタスクをアベイラビリティーゾーン間で均等に分散し、次に各アベイラビリティーゾーン内のメモリに基づいてタスクをビンパックする戦略を作成できます。この場合、アベイラビリティーゾーンの分散戦略が最初に指定されているため、アベイラビリティーゾーンの再調整は機能します。

配置戦略配列の最初の戦略がアベイラビリティーゾーン分散コンポーネントではない場合、アベイラビリティーゾーンの再調整は機能しないことに注意してください。この要件により、タスク分散の主な焦点は、高可用性にとって重要なアベイラビリティーゾーン間のバランスを維持することです。

タスク配置の戦略と制約の詳細については、「Amazon ECS がタスクをコンテナインスタンスに配置する方法」を参照してください。

タスクの配置戦略および制約事項は、Fargate を使用しているタスクをサポートしていません。Fargate は、アクセス可能なアベイラビリティーゾーンにタスクを分散するよう最善を尽くします。キャパシティープロバイダーに Fargate と Fargate Spot の両方が含まれている場合、スプレッドの動作はキャパシティープロバイダーごとに異なります。

次の戦略例は、アベイラビリティーゾーン間でタスクを均等に分散し、次に各アベイラビリティーゾーン内でメモリに基づいてタスクをビンパックします。spread 戦略が最初に行われるため、アベイラビリティーゾーンの再調整はサービスと互換性があります。

```
"placementStrategy": [
    {
        "field": "attribute:ecs.availability-zone",
        "type": "spread"
    },
    {
        "field": "memory",
        "type": "binpack"
    }
]
```

アベイラビリティーゾーンの再調整を有効にする

新規および既存のサービスに対してアベイラビリティーゾーンの再調整を有効にする必要があります。

コンソール、API、AWS CLI、および CloudFormation を使用して、アベイラビリティーゾーンの再調整を有効または無効にできます。

AvailabilityZoneRebalancing のデフォルトの動作は、作成リクエストと更新リクエストで異なります。

- サービスリクエスト作成の場合、AvailabilityZoneRebalancing に値が指定されていない場合、Amazon ECS はデフォルトで値を ENABLED に設定します。
- サービスリクエスト更新の場合、AvailabilityZoneRebalancing に値が指定されていない場合、Amazon ECS はデフォルトでサービスのこれまでの AvailabilityZoneRebalancing の値に設定します。サービスに AvailabilityZoneRebalancing 値が設定されたことがない場合、Amazon ECS はこれを DISABLED として扱います。

サービスタイプ	API	コンソール	CLI
既存	UpdateService	Amazon ECS サービスを更新する	update-service
新	CreateService	Amazon ECS のローリング更新デプロイの作成	サービスの作成

次の例は、新しいサービスの作成時にサービスの再調整を有効にする方法を示しています。

```
aws ecs create-service \
    --cluster my-cluster \
    --service-name my-service \
    --task-definition my-task-definition:1 \
    --desired-count 6 \
    --availability-zone-rebalancing ENABLED
```

Amazon ECS アベイラビリティーゾーンの再調整を追跡する

アベイラビリティーゾーンの再調整がサービスに対して有効になっているかどうかは、コンソールで確認するか、describe-services を呼び出して確認できます。次の例を使用して、CLI でステータスを確認できます。

レスポンスは ENABLED または DISABLED のいずれかになります。

```
aws ecs describe-services \
    --services service-name \
    --cluster cluster-name \
    --query services[0].availabilityZoneRebalancing
```

サービスイベント

Amazon ECS は、アベイラビリティーゾーンの再調整ライフサイクルを理解するのに役立つサービスアクションイベントを送信します。

イベント	シナリオ	タイプ	詳細情報
SERVICE_REBALANCING_STARTED	Amazon ECS がアベイラビリティーゾーンの再調整オペレーションを開始	情報	サービス (*service-name*) は、*Availability Zone 1* の *number-tasks* タスク、*Availability Zone 2* の *number-tasks* タスク、*Availability Zone 3* の *number-tasks* タスクで AZ が調整されていません。AZ の再調整は進行中です。
SERVICE_REBALANCING_COMPLETED	アベイラビリティーゾーンの再調整オペレーションが完了	情報	サービス (*service-name*) は、*Availability Zone 1* の *number-tasks* タスク、*Availability Zone 2* の *number-tasks* タスク、*Availability Zone 3* の

イベント	シナリオ	タイプ	詳細情報
			number-tasks タスクで AZ が調整されています。
TASKS_STARTED	Amazon ECS がアベイラビリティーゾーンの再調整オペレーションの一環としてタスクを正常に開始する	情報	*service-name* は、AZ の再調整のために *Availability Zone* の *number-tasks* タスクを開始しました: *task-ids*。
TASKS_STOPPED	Amazon ECS がアベイラビリティーゾーンの再調整オペレーションの一環としてタスクを正常に停止	情報	*service-name* は、AZ の再調整のために *Availability Zone* の実行中の *number-tasks* タスクを停止しました: *task-ids*。

イベント	シナリオ	タイプ	詳細情報
SERVICE_TASK_PLACEMENT_FAILURE	Amazon ECS がアベイラビリティーゾーンの再調整オペレーションの一環としてタスクを開始することに失敗	エラー	EC2 については、「サービス (*service-name*) で、すべての要件を満たしたコンテナインスタンスがないため、*Availability Zone* にタスクを配置できません。」を参照してください。 Fargate については、「サービス (*service-name*) で、*Availability Zone* にタスクを配置できません。」を参照してください。
TASKSET_SCALE_IN_FAILURE_BY_TASK_PROTECTION	タスク保護が使用されているため、アベイラビリティーゾーンの再調整オペレーションはブロックされます。	情報	サービス (*service-name*) で、*task-set-name* が *reason* によりスケールインできなかったため、AZ を再調整できませんでした。
SERVICE_REBALANCING_STOPPED	アベイラビリティーゾーンの再調整オペレーションが停止。Amazon ECS は、詳細情報を提供する追加のイベントを送信します。	情報	サービス (*service-name*) が AZ の再調整を停止しました。

タスク状態変更イベント

Amazon ECS により、再調整プロセスの一環として開始されるタスクごとに、タスク状態の変更イベント (START) が送信されます。

Amazon ECS は、再調整プロセスの一環として停止するタスクごとに、タスク状態変更イベント (STOPPED) イベントを送信します。理由は `Availability-zone rebalancing initiated by (deployment ecs-svc/`*`deployment-id`*`)` に設定されています。

イベントの詳細については、「Amazon ECS タスク状態変更イベント」を参照してください。

サービス再調整のトラブルシューティング

サービス再調整で問題が発生した場合、次のトラブルシューティングステップを検討してください。

再調整が開始されない

以下を確認してください。

- サービスにサービス再調整が有効になっている
- サービスでサポートされている設定が使用される (「the section called “アベイラビリティーゾーンの再調整の設定に関する考慮事項”」を参照)
- サービスが定常状態になった

再調整中のタスク配置の失敗

`SERVICE_TASK_PLACEMENT_FAILURE` イベントが表示された場合

- EC2 の場合: ターゲットのアベイラビリティーゾーンで利用可能なコンテナインスタンスがあるかどうかを確認します。
- Fargate の場合: タスク配置を制限するリソース制約または Service Quotas があるかどうかを確認します。
- タスク配置の制約を確認し、適切なタスク分散が妨げられていないことを確認します。

予期せずに再調整が停止する

`SERVICE_REBALANCING_STOPPED` イベントが表示された場合

- オペレーションをブロックしている可能性のあるタスク保護を確認します
- 再調整を中断する可能性のある同時サービスデプロイを探します
- 再調整が停止した原因に関する追加情報については、サービスイベントを確認してください。

サービス再調整のベストプラクティス

サービス再調整を最大限に活用するには、次のベストプラクティスに従ってください。

- 再調整オペレーションのモニタリング – 問題を迅速に特定するため、CloudWatch アラームを設定して、再調整に関連するサービスイベントをモニタリングします。
- パフォーマンスへの影響の考慮 – 古いタスクが停止される前に新しいタスクが開始されると、再調整オペレーションによって一時的にリソース使用量が増加する可能性があるので注意してください。
- タスク保護の戦略的な使用 – 再調整中に終了すべきではない重要なタスクがある場合は、タスク保護の使用を検討してください。
- EC2 キャパシティの計画 – EC2 の場合は、効果的な再調整をサポートするため、すべてのアベイラビリティーゾーン間に十分なコンテナインスタンスがあることを確認します。
- 再調整動作のテスト – 本番で再調整に依存する前に、非本番環境で再調整オペレーション中にサービスがどのように動作するかをテストします。

ロードバランサーを使用して Amazon ECS サービストラフィックを分散する

サービスは、オプションで Elastic Load Balancing を使用して、サービスのタスク間でトラフィックを均等に分散するように設定できます。

> **Note**
>
> タスクセットを使用するとき、セット内のすべてのタスクが Elastic Load Balancing を使用するように設定、または Elastic Load Balancing を使用しないように設定する必要があります。

AWS Fargate でホストされている Amazon ECS サービスでは、Application Load Balancer、Network Load Balancer、および Gateway Load Balancer がサポートされています。次の表を使用して、使用するロードバランサーのタイプを確認してください。

ロードバランサーのタイプ	以下の場合に使用	
Application Load Balancer	HTTP/HTTPS (またはレイヤー 7) トラフィックをルーティングします。 Amazon Load Balancer は Amazon ECS サービスでの使用に便利な複数の機能を提供しています。 • 各サービスは、複数のロードバランサーからトラフィックを送信し、複数のターゲットグループを指定することにより複数のロードバランシングポートを公開できます。 • バインドマウントは、Fargate インスタンスと Amazon EC2 インスタンスの両方でホストされているタスクでサポートされています。 • Application Load Balancer により、コンテナが動的ホストポートマッピングを使用できるようになります (同じサービスから複数のタスクがコンテナインスタンスごとに許可するため)。 • Application Load Balancer では、パスベースのルーティングと優先ルールをサポートしています (複数のサービスが 1 つの Applicati	

ロードバランサーのタイプ	以下の場合に使用	
	on Load Balancer で同じリスナーポートを使用するため)。	
Network Load Balancer	TCP または UDP (またはレイヤー 4) トラフィックをルーティングします。	
Gateway Load Balancer	TCP または UDP (またはレイヤー 4) トラフィックをルーティングします。 ファイアウォール、侵入検知および防止システム、ディープパケットインスペクションシステムなどの仮想アプライアンスを使用します。	

サービスで Network Load Balancer または Gateway Load Balancer でのみ使用できる機能が必要な場合を除き、最新の機能を利用できるように、Amazon ECS サービスで Application Load Balancer を使用することをお勧めします。これらのロードバランサーの違いについては、「Elastic Load Balancing ユーザーガイド」の「Elastic Load Balancing とは」を参照してください。

ロードバランサーについては、お客様が利用された分のみのお支払いとなります。詳細については、Elastic Load Balancing の料金表を参照してください。

Amazon ECS のロードバランサーのヘルスチェックパラメータを最適化する

ロードバランサーは、ロードバランサーに対して有効になっているアベイラビリティーゾーンの正常なターゲットにのみ、リクエストをルーティングします。各ターゲットはターゲットグループに登録されます。ロードバランサーは、ターゲットグループのヘルスチェック設定を使用して、各ターゲットの状態を確認します。ターゲットは、登録後に正常と見なされるためには、1 つのヘルスチェックに合格する必要があります。Amazon ECS はロードバランサーをモニタリングします。ロードバランサーは Amazon ECS コンテナに定期的にヘルスチェックを送信します。Amazon ECS エージェントはモニタリングし、ロードバランサーがコンテナの状態を報告するのを待ちます。この処理は、コンテナが正常状態にあると見なされる前に行われます。

Elastic Load Balancing の次の 2 つのヘルスチェックパラメータがデプロイ速度に影響します。

- ヘルスチェック間隔: 個々のコンテナのヘルスチェック間のおおよその時間を秒単位で決定します。デフォルトでは、ロードバランサーは 30 秒ごとにチェックします。

  このパラメータは以下の名前です。

  - Elastic Load Balancing API の `HealthCheckIntervalSeconds`
  - Amazon EC2 コンソールでの [インターバル]
- 正常性しきい値: 異常なコンテナが正常であると判断されるまでに必要なヘルスチェックの連続成功回数を決定します。デフォルトでは、ロードバランサーはターゲットコンテナが正常であると報告する前に 5 回のヘルスチェックに合格する必要があります。

  このパラメータは以下の名前です。

  - Elastic Load Balancing API の `HealthyThresholdCount`
  - Amazon EC2 コンソールの [正常なしきい値]

重要: 新しく登録されたターゲットの場合、正常なしきい値数の設定に関係なく、ターゲットを正常とみなすために必要なヘルスチェックは 1 回だけです。正常なしきい値数は、ターゲットが異常な状態から正常な状態に戻っている場合にのみ適用されます。

デフォルト設定では、ターゲットが異常な状態になった後回復した場合、コンテナの状態を判断するための合計時間は 2 分 30 秒 (`30 seconds * 5 = 150 seconds`) です。

サービスが 10 秒以内に起動して安定すれば、ヘルスチェックプロセスをスピードアップできます。プロセスを高速にするには、ヘルスチェックの間隔と正常なしきい値の数を減らします。

- `HealthCheckIntervalSeconds` (Elastic Load Balancing API 名) または [インターバル] (Amazon EC2 コンソール名): 5
- `HealthyThresholdCount` (Elastic Load Balancing API 名) または [正常なしきい値] (Amazon EC2 コンソール名): 2

この設定では、ヘルスチェックの処理にデフォルトの 2 分 30 秒かかるのに対し、10 秒かかります。

Elastic Load Balancing ヘルスチェックパラメータの詳細については、「Elastic Load Balancing ユーザーガイド」の「ターゲットグループのヘルスチェック」を参照してください。

Amazon ECS のロードバランサー Connection Draining パラメータを最適化する

最適化を可能にするために、クライアントはコンテナサービスへのキープアライブ接続を維持します。これにより、そのクライアントからの後続のリクエストが既存の接続を再利用できるようにします。コンテナへのトラフィックを停止したいときは、ロードバランサーに通知します。ロードバランサーはクライアントがキープアライブ接続を閉じたかどうかを定期的に確認します。Amazon ECS はロードバランサーをモニタリングし、キープアライブ接続が閉じられた (ターゲットが UNUSED の状態にある) ことをロードバランサーが報告するのを待ちます。

ロードバランサーがターゲットを UNUSED 状態へと移行するまで待機する時間の長さが登録解除の遅延です。以下のロードバランサーパラメータを設定して、デプロイをスピードアップできます。

- `deregistration_delay.timeout_seconds`: 300 (デフォルト)

レスポンス時間が 1 秒未満のサービスの場合は、パラメータを次の値に設定して、クライアントとバックエンドサービスとの間の接続を切断するまでロードバランサーが 5 秒だけ待機するようにします。

- `deregistration_delay.timeout_seconds`: 5

> **Note**
>
> ファイルのアップロードやストリーミング接続が遅いなど、リクエストの有効期間が長いサービスの場合は、この値を 5 秒に設定しないでください。

SIGTERM 応答性

Amazon ECS は、まず停止シグナルをタスクに送信し、アプリケーションを終了してシャットダウンする必要があることを通知します。このシグナルは、STOPSIGNAL 命令を使用してコンテナイメージで定義でき、デフォルトで SIGTERM になります。次に、Amazon ECS は SIGKILL メッセージを送信します。アプリケーションが SIGTERM を無視する場合、Amazon ECS サービスはプロセスを終了する SIGKILL シグナルの送信を待つ必要があります。

Amazon ECS が SIGKILL メッセージの送信を待つ時間は、次の Amazon ECS エージェントオプションによって決まります。

- `ECS_CONTAINER_STOP_TIMEOUT`: 30 (デフォルト)

コンテナエージェントパラメータの詳細については、GitHub の「Amazon ECS コンテナエージェント」を参照してください。

待機時間を短縮するには、Amazon ECS エージェントパラメータを次の値に設定します。

- `ECS_CONTAINER_STOP_TIMEOUT: 2`

  アプリケーションに 1 秒以上かかる場合は、値に 2 を掛けて、その数値を値として使用します。

この場合、Amazon ECS はコンテナがシャットダウンされるまで 2 秒間待機し、アプリケーションが停止しなかったときに Amazon ECS は SIGKILL メッセージを送信します。

SIGTERM シグナルをトラップして反応するようにアプリケーションコードを変更することもできます。JavaScript の例を次に示します。

```
process.on('SIGTERM', function() {
  server.close();
})
```

このコードにより、HTTP サーバーは新しいリクエストのリッスンを停止し、処理中のリクエストへの応答を終了します。イベントループは何も行わないため Node.js プロセスが終了します。これを考慮すると、プロセスが実行中のリクエストを完了するのに 500 ミリ秒しかかからない場合、プロセスは停止タイムアウトを待って SIGKILL を送信する必要がなく、早期に終了します。

Amazon ECS 用の Application Load Balancer を使用する

Application Load Balancer では、アプリケーションレイヤー (HTTP/HTTPS) でルーティング決定を行い、パスベースのルーティングをサポートします。また、クラスター内の各コンテナインスタンスの 1 つまたは複数のポートにリクエストをルーティングできます。Application Load Balancer では、動的ホストポートマッピングをサポートしています。たとえば、タスクコンテナ定義で NGINX コンテナポートのポート 80、ホストポートのポート 0 を指定した場合、ホストポートはコンテナインスタンスの一時ポート範囲から動的に選択されます (最新の Amazon ECS に最適化された AMI の 32768 ~ 61000 など)。タスクの起動時に、NGINX コンテナは、インスタンス ID およびポートの組み合わせとして Application Load Balancer で登録されます。トラフィックは、コンテナに対応するインスタンス ID およびポートに分散されます。この動的なマッピングにより、同じコンテナインスタンスの単一のサービスから複数のタスクを持つことができます。詳細については、「Application Load Balancer ユーザーガイド」を参照してください。

デプロイを高速化するためのパラメータ設定のベストプラクティスについては、以下を参照してください。

- Amazon ECS のロードバランサーのヘルスチェックパラメータを最適化する
- Amazon ECS のロードバランサー Connection Draining パラメータを最適化する

Amazon ECS で Application Load Balancer を使用する場合は、次の点を考慮してください。

- Amazon ECS には、タスクの作成時および停止時に、ロードバランサーへのターゲットの登録および登録解除に必要なアクセス許可を提供するサービスリンク IAM ロールが必要です。詳細については、「Amazon ECS のサービスリンクロールの使用」を参照してください。
- IPv6 専用設定のサービスの場合は、Application Load Balancer のターゲットグループの IP アドレスタイプを `dualstack` または `dualstack-without-public-ipv4` に設定する必要があります。
- `awsvpc` ネットワークモードを使用するタスクを含むサービスの場合、サービスのターゲットグループを作成するときに、`ip` ではなく、`instance` をターゲットタイプとして選択する必要があります 。これは、`awsvpc` ネットワークモードを使用するタスクは、Amazon EC2 インスタンスではなく、Elastic Network Interface に関連付けられているためです。
- サービスが HTTP/HTTPS サービスのポート 80 やポート 443 など、複数の負荷分散されたポートにアクセスする必要がある場合は、2 つのリスナーを設定できます。1 つのリスナーは、リクエストをサービスに転送する HTTPS を担当し、別のリスナーは HTTP リクエストを適切な HTTPS ポートへのリダイレクトを担当します。詳細については、Application Load Balancer ユーザーガイドの「Application Load Balancer のアクセスログ」を参照してください。
- ロードバランサーのサブネット設定には、コンテナインスタンスが存在するアベイラビリティーゾーンすべてを含める必要があります。
- サービスの作成後にロードバランサーの設定を AWS マネジメントコンソール から変更することはできません。AWS Copilot、AWS CloudFormation、AWS CLI、SDK のいずれかを使用して、AWS CodeDeploy ブルー/グリーンまたは外部ではなく、ECS ローリングデプロイコントローラのみのロードバランサー設定を変更できます。ロードバランサー設定を追加、更新、削除すると、Amazon ECS は、更新された Elastic Load Balancing 設定で新しいデプロイを開始します。これにより、タスクがロードバランサーに登録およびロードバランサーから登録解除されます。Elastic Load Balancing 設定を更新する前に、テスト環境でこれを検証することをお勧めします。設定の変更方法については、「Amazon Elastic Containers Service API リファレンス」の「UpdateService」を参照してください。

- サービスタスクがロードバランサーのヘルスチェック条件を満たさない場合、タスクは停止され、再起動されます。このプロセスは、サービスが実行中のタスクの必要数に達するまで続行されます。
- ロードバランサーで有効にされているサービスに問題がある場合は、「Amazon ECS のサービスロードバランサーのトラブルシューティング」を参照してください。
- `instance` ターゲットタイプを使用する場合、タスクおよびロードバランサーは同じ VPC 内にある必要があります。`ip` ターゲットタイプを使用する場合、クロス VPC 接続がサポートされます。
- 各サービスに固有のターゲットグループを使用します。

  複数のサービスに同じターゲットグループを使用すると、サービスのデプロイ時に問題が発生する可能性があります。
- Application Load Balancer に関連付けられているターゲットグループを指定する必要があります。

Application Load Balancer の作成方法については、「Application Load Balancer」の「Application Load Balancer の作成」を参照してください。

Amazon ECS 用の Network Load Balancer を使用する

Network Load Balancer により、トランスポートレイヤー (TCP/SSL) でルーティングの決定が行われます。毎秒数百万のリクエストを処理できます。ロードバランサーは、接続を受信すると、フローハッシュルーティングアルゴリズムを使用してデフォルトルールのターゲットグループからターゲットを選択します。リスナー構成で指定されたポート上の選択したターゲットへの TCP 接続を開こうとします。ヘッダーを変更せずにリクエストを転送します。Network Load Balancer では、動的ホストポートマッピングをサポートしています。たとえば、タスクコンテナ定義で NGINX コンテナポートのポート 80、ホストポートのポート 0 を指定した場合、ホストポートはコンテナインスタンスの一時ポート範囲から動的に選択されます (最新の Amazon ECS に最適化された AMI の 32768 ~ 61000 など)。タスクの起動時に、NGINX コンテナは、インスタンス ID およびポートの組み合わせとして Network Load Balancer で登録されます。トラフィックは、コンテナに対応するインスタンス ID およびポートに分散されます。この動的なマッピングにより、同じコンテナインスタンスの単一のサービスから複数のタスクを持つことができます。Network Load Balancer の詳細については、「https://docs.aws.amazon.com/elasticloadbalancing/latest/network/ のユーザーガイド」を参照してください。

デプロイを高速化するためのパラメータ設定のベストプラクティスについては、以下を参照してください。

- Amazon ECS のロードバランサーのヘルスチェックパラメータを最適化する
- Amazon ECS のロードバランサー Connection Draining パラメータを最適化する

Amazon ECS で Network Load Balancer を使用する場合は、次の点を考慮してください。

- Amazon ECS には、タスクの作成時および停止時に、ロードバランサーへのターゲットの登録および登録解除に必要なアクセス許可を提供するサービスリンク IAM ロールが必要です。詳細については、「Amazon ECS のサービスリンクロールの使用」を参照してください。
- 1 つのサービスに 5 つ以上のターゲットグループをアタッチすることはできません。
- IPv6 専用設定のサービスの場合は、Network Load Balancer のターゲットグループの IP アドレスタイプを `dualstack` に設定する必要があります。
- `awsvpc` ネットワークモードを使用するタスクを含むサービスの場合、サービスのターゲットグループを作成するときに、`ip` ではなく、`instance` をターゲットタイプとして選択する必要があります 。これは、`awsvpc` ネットワークモードを使用するタスクは、Amazon EC2インスタンスではなく、Elastic Network Interface に関連付けられているためです。
- ロードバランサーのサブネット設定には、コンテナインスタンスが存在するアベイラビリティーゾーンすべてを含める必要があります。
- サービスの作成後にロードバランサーの設定を AWS マネジメントコンソール から変更することはできません。AWS Copilot、AWS CloudFormation、AWS CLI、SDK のいずれかを使用して、AWS CodeDeploy ブルー/グリーンまたは外部ではなく、ECS ローリングデプロイコントローラのみのロードバランサー設定を変更できます。ロードバランサー設定を追加、更新、削除すると、Amazon ECS は、更新された Elastic Load Balancing 設定で新しいデプロイを開始します。これにより、タスクがロードバランサーに登録およびロードバランサーから登録解除されます。Elastic Load Balancing 設定を更新する前に、テスト環境でこれを検証することをお勧めします。設定の変更方法については、「Amazon Elastic Containers Service API リファレンス」の「UpdateService」を参照してください。
- サービスタスクがロードバランサーのヘルスチェック条件を満たさない場合、タスクは停止され、再起動されます。このプロセスは、サービスが実行中のタスクの必要数に達するまで続行されます。
- IP アドレスをターゲットとして設定し、クライアント IP の保持をオフにした Gateway Load Balancer を使用する場合、リクエストは Gateway Load Balancer のプライベート IP アドレスから送信されたと見なされます。これは、ターゲットのセキュリティグループで受信リクエストとヘルスチェックを許可するとすぐに、Gateway Load Balancer の背後にあるサービスが世界中からアクセス可能になることを意味します。

- Fargate タスクでは、プラットフォームバージョン `1.4.0` (Linux) または `1.0.0` (Windows) を使用する必要があります。
- ロードバランサーで有効にされているサービスに問題がある場合は、「Amazon ECS のサービスロードバランサーのトラブルシューティング」を参照してください。
- `instance` ターゲットタイプを使用する場合、タスクおよびロードバランサーは同じ VPC 内にある必要があります。`ip` ターゲットタイプを使用する場合、クロス VPC 接続がサポートされます。
- Network Load Balancer のクライアント IP アドレスの保存は Fargate ターゲットと互換性があります。
- 各サービスに固有のターゲットグループを使用します。

  複数のサービスに同じターゲットグループを使用すると、サービスのデプロイ時に問題が発生する可能性があります。
- Network Load Balancer に関連付けられているターゲットグループを指定する必要があります。

Network Load Balancer を作成する方法については、「Network Load Balancers」の「Network Load Balancer の作成」を参照してください。

> **Important**
>
> サービスのタスク定義で、`awsvpc` ネットワークモード (Fargate の場合に必要) が使用されている場合は、`instance` ではなく、`ip` をターゲットタイプとして選択する必要があります。これは、`awsvpc` ネットワークモードを使用するタスクは、Amazon EC2インスタンスではなく、Elastic Network Interface に関連付けられているためです。
> インスタンス ID が
> C1、CC1、CC2、CG1、CG2、CR1、G1、G2、HI1、HS1、M1、M2、M3、および T1 のインスタンス ID でインスタンスを登録することはできません。IP アドレスで、これらの種類のインスタンスを登録することができます。

Amazon ECS 用の Gateway Load Balancer を使用する

ゲートウェイロードバランサーは、開放型システム間相互接続 (OSI) モデルの第 3 層 (ネットワークレイヤー) で機能します。すべてのポートですべての IP パケットをリッスンし、リスナールールで指定されたターゲットグループにトラフィックを転送します。5 タプル (TCP/UDP フローの場合) または 3 タプル (非 TCP/UDP フローの場合) を使用して、特定のターゲットアプライアン

スへのフローのスティッキ状態を維持します。たとえば、タスクコンテナ定義で NGINX コンテナポートのポート 80、ホストポートのポート 0 を指定した場合、ホストポートはコンテナインスタンスの一時ポート範囲から動的に選択されます (最新の Amazon ECS に最適化された AMI の 32768 ~ 61000 など)。タスクの起動時に、NGINX コンテナは、インスタンス ID およびポートの組み合わせとして Gateway Load Balancer で登録されます。トラフィックは、コンテナに対応するインスタンス ID およびポートに分散されます。この動的なマッピングにより、同じコンテナインスタンスの単一のサービスから複数のタスクを持つことができます。詳細については、「Gateway Load Balancers」の「What is a Gateway Load Balancer」を参照してください。

デプロイを高速化するためのパラメータ設定のベストプラクティスについては、以下を参照してください。

- Amazon ECS のロードバランサーのヘルスチェックパラメータを最適化する
- Amazon ECS のロードバランサー Connection Draining パラメータを最適化する

Amazon ECS で Gateway Load Balancer を使用する場合は、次の点を考慮してください。

- Amazon ECS には、タスクの作成時および停止時に、ロードバランサーへのターゲットの登録および登録解除に必要なアクセス許可を提供するサービスリンク IAM ロールが必要です。詳細については、「Amazon ECS のサービスリンクロールの使用」を参照してください。
- IPv6 専用設定のサービスの場合は、Gateway Load Balancer のターゲットグループの IP アドレスタイプを `dualstack` に設定する必要があります。
- `awsvpc` 以外のネットワークモードを使用するタスクがあるサービスの場合、Gateway Load Balancer はサポートされません。
- ロードバランサーのサブネット設定には、コンテナインスタンスが存在するアベイラビリティーゾーンすべてを含める必要があります。
- サービスの作成後にロードバランサーの設定を AWS マネジメントコンソール から変更することはできません。AWS Copilot、AWS CloudFormation、AWS CLI、SDK のいずれかを使用して、AWS CodeDeploy ブルー/グリーンまたは外部ではなく、ECS ローリングデプロイコントローラのみのロードバランサー設定を変更できます。ロードバランサー設定を追加、更新、削除すると、Amazon ECS は、更新された Elastic Load Balancing 設定で新しいデプロイを開始します。これにより、タスクがロードバランサーに登録およびロードバランサーから登録解除されます。Elastic Load Balancing 設定を更新する前に、テスト環境でこれを検証することをお勧めします。設定の変更方法については、「Amazon Elastic Containers Service API リファレンス」の「UpdateService」を参照してください。

- サービスタスクがロードバランサーのヘルスチェック条件を満たさない場合、タスクは停止され、再起動されます。このプロセスは、サービスが実行中のタスクの必要数に達するまで続行されます。
- IP アドレスをターゲットとして設定した Gateway Load Balancer を使用する場合、リクエストは Gateway Load Balancer のプライベート IP アドレスから送信されたと見なされます。これは、ターゲットのセキュリティグループで受信リクエストとヘルスチェックを許可するとすぐに、Gateway Load Balancer の背後にあるサービスが世界中からアクセス可能になることを意味します。
- Fargate タスクでは、プラットフォームバージョン `1.4.0` (Linux) または `1.0.0` (Windows) を使用する必要があります。
- ロードバランサーで有効にされているサービスに問題がある場合は、「Amazon ECS のサービスロードバランサーのトラブルシューティング」を参照してください。
- `instance` ターゲットタイプを使用する場合、タスクおよびロードバランサーは同じ VPC 内にある必要があります。`ip` ターゲットタイプを使用する場合、クロス VPC 接続がサポートされます。
- 各サービスに固有のターゲットグループを使用します。

  複数のサービスに同じターゲットグループを使用すると、サービスのデプロイ時に問題が発生する可能性があります。
- Gateway Load Balancer に関連付けられているターゲットグループを指定する必要があります。

Gateway Load Balancer を作成する方法については、「Gateway Load Balancers」の「Gateway Load Balancer の開始方法」を参照してください。

> **Important**
>
> サービスのタスク定義で、`awsvpc` ネットワークモード (Fargate の場合に必要) が使用されている場合は、`instance` ではなく、`ip` をターゲットタイプとして選択する必要があります。これは、`awsvpc` ネットワークモードを使用するタスクは、Amazon EC2インスタンスではなく、Elastic Network Interface に関連付けられているためです。
> インスタンス ID が
> C1、CC1、CC2、CG1、CG2、CR1、G1、G2、HI1、HS1、M1、M2、M3、および T1 のインスタンス ID でインスタンスを登録することはできません。IP アドレスで、これらの種類のインスタンスを登録することができます。

Amazon ECS サービスに複数のターゲットグループを登録する

サービス定義で複数のターゲットグループを指定すると、Amazon ECS サービスは複数のロードバランサーからのトラフィックを送信し、複数のロードバランサーポートを公開できます。

複数のターゲットグループを指定してサービスを作成するには、Amazon ECS API、SDK、AWS CLI、または CloudFormation テンプレートを使用してサービスを作成する必要があります。サービスの作成後、AWS マネジメントコンソール に登録されているサービスとターゲットグループを表示できます。既存サービスのロードバランサー設定を変更するには、UpdateService を使用する必要があります。

次の形式を使用して、複数のターゲットグループをサービス定義で指定できます。サービス定義の完全な構文については、「サービス定義テンプレート」を参照してください。

```
"loadBalancers":[
   {

 "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_1/1234567890123456",
      "containerName":"container_name",
      "containerPort":container_port
   },
   {

 "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_2/6543210987654321",
      "containerName":"container_name",
      "containerPort":container_port
   }
]
```

考慮事項

サービス定義で複数のターゲットグループを指定する際には、次の点を考慮する必要があります。

- Application Load Balancer または Network Load Balancer を使用するサービスの場合、6 個以上のターゲットグループはアタッチできません。
- サービス定義での複数のターゲットグループの指定は、次の条件でのみサポートされます。
  - サービスでは、Application Load Balancer またはNetwork Load Balancer を使用する必要があります。

  - サービスで (ECS) デプロイコントローラタイプを使用する必要があります。これは、Amazon ECS ネイティブ/ブルーグリーンデプロイ、またはローリング更新デプロイのいずれかです。
- Fargate と EC2 の両方の起動タイプを使用するタスクを含むサービスでは、複数のターゲットグループの指定がサポートされます。
- 複数のターゲットグループを指定するサービスを作成するときは、Amazon ECS サービスにリンクされたロールを作成する必要があります。ロールは、API リクエストの `role` パラメータを省略するか、`Role` の CloudFormation プロパティを省略することによって作成されます。詳細については、「Amazon ECS のサービスリンクロールの使用」を参照してください。

サービス定義の例

次に、サービス定義で複数のターゲットグループを指定するいくつかのユースケースの例を示します。サービス定義の完全な構文については、「サービス定義テンプレート」を参照してください。

内部トラフィックと外部トラフィックに別々のロードバランサーを使用する

次のユースケースでは、サービスは 2 つの別個のロードバランサーを使用します。1 つは内部トラフィック用、もう 1 つはインターネット向けトラフィック用で、同じコンテナとポートに対して使用します。

```
"loadBalancers":[
   //Internal ELB
   {

 "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_1/1234567890123456",
      "containerName":"nginx",
      "containerPort":8080
   },
   //Internet-facing ELB
   {

 "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_2/6543210987654321",
      "containerName":"nginx",
      "containerPort":8080
   }
]
```

同じコンテナの複数のポートを公開する

次のユースケースでは、サービスは 1 つのロードバランサーを使用しますが、同じコンテナから複数のポートを公開します。例えば、Jenkins コンテナは、Jenkins ウェブインターフェイス用にポート 8080 を、API 用にポート 50000 を公開する場合があります。

```
"loadBalancers":[
   {

 "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_1/1234567890123456",
      "containerName":"jenkins",
      "containerPort":8080
   },
   {

 "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_2/6543210987654321",
      "containerName":"jenkins",
      "containerPort":50000
   }
]
```

複数のコンテナのポートを公開する

次のユースケースでは、サービスは 1 つのロードバランサーと 2 つのターゲットグループを使用して、個別のコンテナからポートを公開します。

```
"loadBalancers":[
   {

 "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_1/1234567890123456",
      "containerName":"webserver",
      "containerPort":80
   },
   {

 "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_2/6543210987654321",
      "containerName":"database",
      "containerPort":3306
```

```
    }
]
```

Amazon ECS サービスを自動的にスケールする

オートスケーリング は、Amazon ECS サービスで必要なタスク数を自動的に増減する機能です。Amazon ECS は アプリケーション Auto Scaling サービスを活用してこの機能を提供します。詳細については、 ユーザーガイドの Application Auto Scaling リファレンスを参照してください。

Amazon ECS はご使用のサービスの CPU とメモリの平均使用量を含む CloudWatch メトリクスを発行します。詳細については、「Amazon ECS サービスの使用率メトリクス」を参照してください。これらおよびその他の CloudWatch メトリクスを使用して、ピーク時に高需要に対処するためにサービスをスケールアウトし (実行するタスクを増やし)、使用率の低い期間にコストを削減するためにサービスをスケールインする (実行するタスクを減らす) ことができます。

Amazon ECS Service Auto Scaling は、以下のタイプの自動スケーリングをサポートします。

- ターゲットメトリクスを使用して Amazon ECS サービスをスケールする - 特定のメトリクスのターゲット値に基づいて、サービスが実行するタスク数を増減させます。これはサーモスタットが家の温度を維持する方法に似ています。温度を選択すれば、後はサーモスタットがすべてを実行します。
- CloudWatch アラームに基づく定義済みの増分を使用して Amazon ECS サービスをスケールする - アラーム超過のサイズに応じて変動する一連のスケーリング調整値 (ステップ調整値) に基づいて、サービスが実行するタスク数を増減させます。
- スケジュールされたアクションを使用して Amazon ECS サービスをスケールする - 日付と時刻に基づいてサービスが実行するタスクの数を増減させます。
- 履歴パターンを使用して予測スケーリングで Amazon ECS サービスをスケールする — トラフィックフローの日次または週次のパターンを検出するための過去の負荷データ分析に基づいて、サービスが実行するタスク数を増減します。

考慮事項

スケーリングポリシーを使用する場合は、次の考慮事項に注意してください。

- Amazon ECSは、CloudWatch に 1 分間隔でメトリクスを送信します。クラスターとサービスが CloudWatch にメトリクスを送信するまで、メトリクスは使用できません。また、存在しないメトリクスに対して CloudWatch アラームを作成することはできません。

- スケーリングポリシーは、クールダウン期間をサポートします。これは、以前のスケーリングアクティビティが有効になるまで待機する秒数です。
  - スケールアウトイベントでは、スケールアウトが継続的に (ただし過剰になることなく) 行われます。スケーリングポリシーを使用してサービスの自動スケーリングが正常にスケールアウトすると、クールダウン時間の計算が開始されます。スケーリングポリシーは、より大きなスケールアウトが開始されるか、クールダウン期間が終了しない限り、必要な容量を再度増加させません。このスケールアウトクールダウン期間が有効な間は、スケールアウトアクティビティを開始することで追加された容量は、次のスケールアウトアクティビティに予定される容量の一部として繰り入れられます。
  - スケールインイベントでは、アプリケーションの可用性を保護するために控えめにスケールインされます。そのため、スケールインアクティビティはクールダウン期間が終了するまでブロックされます。ただし、スケールインクールダウン期間中に別のアラームがスケールアウトアクティビティを開始した場合、アプリケーションの自動スケーリングスケールによってターゲットが即座にスケールアウトされます。この場合、スケールインクールダウン期間は停止し、完了しません。
- サービススケジューラは常に必要数を優先しますが、サービスにアクティブなスケーリングポリシーとアラームがある限り、サービスの自動スケーリングはユーザーが手動で設定した必要数を変更できます。
- サービスの必要タスク数が容量最小値より小さく設定された状態で、アラームがスケールアウトアクティビティを開始したとき、サービスの自動スケーリングが必要タスク数を容量最小値までスケールアップします。その後もアラームに関連付けられたスケーリングポリシーに基づいて、必要に応じてスケーリングし続けます。ただし、必要数はすでにキャパシティーの最小値より小さいため、スケールインアクティビティでは調整されません。
- サービスの必要タスク数が容量最大値より大きく設定された状態で、アラームがスケールインアクティビティを開始したとき、Service Auto Scaling が必要タスク数を容量最大値までスケールアウトします。その後もアラームに関連付けられたスケーリングポリシーに基づいて、必要に応じてスケーリングし続けます。ただし、必要タスク数はすでに容量最大値より大きいため、スケールアウトアクティビティでは調整されません。
- スケーリングアクティビティ中、サービスで実際に実行されているタスクの数は、必要数ではなく、サービスの自動スケーリングが開始点として使用する値です。これが想定される処理能力です。これにより、例えば、追加タスクを配置するために十分なコンテナインスタンスリソースがない場合に、満たすことができない過剰な (ランナウェイ) スケーリングを防ぐことができます。後でコンテナインスタンスのキャパシティーを使用できるようになった場合、保留中の規模の拡大や縮小が続行され、クールダウン期間後にさらに規模の拡大や縮小を続行できることができます。

- 実行する作業がないときにタスク数をゼロにスケーリングするには、キャパシティーの最小値を 0 に設定します。ターゲット追跡スケーリングポリシーでは、実際の容量が 0 で、メトリクスがワークロードの需要があることを示している場合、サービスの自動スケーリングは 1 つのデータポイントの送信を待ってからスケールアウトします。この場合、開始点として可能な最小量だけスケールアウトしてから、実際の実行中のタスク数に基づいてスケーリングを再開します。
- Application Auto Scaling は、Amazon ECS デプロイの進行中にスケールインプロセスをオフにします。ただし、スケールアウトプロセスは、中断しない限り、デプロイ中に引き続き発生します。この動作は、外部デプロイコントローラーを使用した Amazon ECS サービスには適用されません。詳細については、「サービスの自動スケーリングとデプロイ」を参照してください。
- Amazon ECS タスクには、いくつかの Application Auto Scaling オプションがあります。ターゲットトラッキングは最も使いやすいモードです。これにより、CPU 平均使用率などのメトリクスの目標値を設定するだけです。次に、オートスケーラーは、その値を達成するために必要なタスクの数を自動的に管理します。ステップスケーリングを使用すると、スケーリングメトリクスの特定のしきい値と、しきい値を超えたときに追加または削除するタスクの数を定義できるため、需要の変化に迅速に対応できます。さらに重要なことは、しきい値アラームが超過する時間を最小限に抑えることで、需要の変化に非常に迅速に対応できることです。

サービスの自動スケーリングにおけるベストプラクティスの詳細については、「Amazon ECS サービスの自動スケーリングの最適化」を参照してください。

サービスの自動スケーリングとデプロイ

Application Auto Scaling は、Amazon ECS デプロイの進行中にスケールインプロセスをオフにします。ただし、スケールアウトプロセスは、中断しない限り、デプロイ中に引き続き発生します。この動作は、外部デプロイコントローラーを使用した Amazon ECS サービスには適用されません。デプロイの進行中にスケールアウトプロセスを中断する場合は、次の手順を実行します。

1. Application Auto Scaling のスケーラブルなターゲットに関連付けられたサービスのリソース ID (例: `service/default/sample-webapp`) を指定して describe-scalable-targets コマンドを呼び出します。出力を記録します。これは、次のコマンドを呼び出すときに必要になります。
2. リソース ID、名前空間、およびスケーラブルなディメンションを指定して register-scalable-target コマンドを呼び出します。`true` と`DynamicScalingInSuspended` の両方に `DynamicScalingOutSuspended` を指定します。
3. デプロイが完了したら、register-scalable-target コマンドを呼び出してスケーリングを再開できます。

詳細については、「Application Auto Scaling のスケーリングの中断と再開」を参照してください。

ターゲットメトリクスを使用して Amazon ECS サービスをスケールする

ターゲット追跡スケーリングポリシーで、メトリクスを選択してターゲット値を設定します。Amazon ECS サービス自動スケーリングは、スケーリングポリシーを制御する CloudWatch アラームを作成および管理し、メトリクスとターゲット値に基づいてスケーリング調整値を計算します。スケーリングポリシーは、指定されたターゲット値、またはそれに近い値にメトリクスを維持するため、必要に応じてサービスタスクを追加または削除します。ターゲットの追跡スケーリングポリシーは、メトリクスをターゲット値近くに維持することに加えて、負荷パターンの変動によるメトリクスの変動に合わせて調整し、サービスで実行されているタスク数の急速な変動を最小化します。

ターゲット追跡ポリシーにより、CloudWatch アラームとスケーリング調整を手動で定義する必要がなくなります。Amazon ECS は、設定したターゲットに基づいてこれを自動的に処理します。

ターゲット追跡ポリシーを使用する際は、以下について検討します。

- ターゲットの追跡スケーリングポリシーでは、指定されたメトリクスがターゲット値を超えている場合、スケールアウトする必要があると見なされます。指定されたメトリクスがターゲット値を下回っている場合、ターゲットの追跡スケーリングポリシーを使用してスケールアウトすることはできません。
- 指定されたメトリクスに十分なデータがない場合、ターゲットの追跡スケーリングポリシーによってスケールされません。不十分なデータは低い使用率として解釈されないため、スケールインされません。
- ターゲット値と実際のメトリクスデータポイント間にギャップが発生する場合があります。これは、Service Auto Scaling が追加または削除する容量を判断するときに、その数を切り上げまたは切り捨てることによって、常に控えめに動作するためです。これにより、不十分な容量を追加したり、必要以上に容量を削除することを防ぎます。
- アプリケーションの可用性を高めるために、サービスのスケールアウトはメトリクスに比例して可能な限り高速に行われますが、スケールインはより緩やかです。
- Application Auto Scaling は、Amazon ECS デプロイの進行中にスケールインプロセスをオフにします。ただし、スケールアウトプロセスは、中断しない限り、デプロイ中に引き続き発生します。この動作は、外部デプロイコントローラーを使用した Amazon ECS サービスには適用されません。詳細については、「サービスの自動スケーリングとデプロイ」を参照してください。
- Amazon ECS サービスに対して複数のターゲット追跡スケーリングポリシーを設定できます。ただし、各ポリシーがそれぞれ異なるメトリクスを使用している必要があります。サービスの自動スケーリングの目的は常に可用性を優先することであるため、その動作は、スケールアウトまたは

スケールインに対するターゲット追跡ポリシーの準備が整っているかどうかに応じて異なります。ターゲット追跡ポリシーのいずれかでスケールアウトする準備ができると、サービスがスケールアウトされますが、すべてのターゲット追跡ポリシー (スケールイン部分がオン) でスケールインする準備ができている場合にのみスケールインされます。

- ターゲット追跡スケーリングポリシーのためにサービスの自動スケーリングが管理する CloudWatch アラームを編集または削除しないでください。スケーリングポリシーを削除するときに、サービスの自動スケーリングはアラームを自動的に削除します。
- ブルー/グリーンデプロイタイプでは、ターゲット追跡スケーリングポリシーの `ALBRequestCountPerTarget` メトリクスはサポートされません。

ターゲット追跡スケーリングポリシーの詳細については、「Application Auto Scaling ユーザーガイド」の「ターゲット追跡スケーリングポリシー」を参照してください。

Amazon ECS サービスの自動スケーリングのターゲット追跡スケーリングポリシーを作成する

ターゲット追跡スケーリングポリシーを作成して、Amazon ECS がサービスで必要なタスク数を自動的に増減するようにします。ターゲット追跡は、ターゲットメトリクス値に基づいて機能します。

コンソール

1. サービスの作成や更新に使用する標準の IAM アクセス権限に加えて、追加のアクセス権限が必要です。詳細については、「Amazon ECS のサービス自動スケーリングに必要な IAM アクセス許可」を参照してください。
2. ポリシーに使用するメトリクスを決定します。以下のメトリクスが利用可能です。
   - [ECSServiceAverageCPUUtilization] — サービスが使用する平均 CPU 使用率。
   - [ECSServiceAverageMemoryUtilization] — サービスが使用する平均メモリ使用率。
   - [ALBRequestCountPerTarget] – タスクが受信する 1 分あたりの理想的な平均リクエスト数。
3. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
4. [クラスター] ページで、クラスターを選択します。
5. [クラスターの詳細] ページの [サービス] セクションで、サービスを選択します。

   [サービス詳細] ページが表示されます。
6. [タスク数を設定する] を選択します。
7. [Amazon ECS サービスのタスク数] で、[自動スケーリングを使用する] を選択します。

   [タスク数セクション] が表示されます。

a. [タスクの最小数] に、サービスの自動スケーリングで使用するタスクの下限数を入力します。必要な数がこの数を下回ることはありません。

b. [最大] に、サービスの自動スケーリングで使用するタスクの上限数を入力します。必要な数がこの数を超えることはありません。

c. [保存] を選択します。

[ポリシー] ページが表示されます。

8. [スケーリングポリシーを作成する] を選択します。

[ポリシーを作成する] ページが表示されます。

9. [スケーリングポリシータイプ] で [ターゲットの追跡] を選択します。

10. [Policy Name] (ポリシー名) にこのポリシーの名前を入力します。

11. [メトリクスのタイプ] で、オプションのリストからメトリクスを選択します。

12. [ターゲット使用率] には、Amazon ECS が維持するタスクの割合のターゲット値を入力します。サービスの自動スケーリングは、平均使用率が目標使用率になるまで、または指定した最大タスク数に達するまで、キャパシティをスケールアウトします。

13. [追加設定] で、以下を実行します。

a. [スケールインのクールダウン期間] に、スケールインアクティビティが完了してから別のスケールインアクティビティが開始されるまでの時間を秒単位で入力します。

b. [スケールアウトのクールダウン期間] には、前回のスケールアウトアクティビティが有効になるまで待機する時間を秒単位で入力します。

c. スケールアウトポリシーのみを作成するには、[スケールインを無効にする] を選択します。

14. [スケーリングポリシーを作成する] を選択します。

AWS CLI

1. register-scalable-target コマンドを使用して、スケーラブルなターゲットとして Amazon ECS サービスを登録します。

2. put-scaling-policy コマンドを使用して、スケーリングポリシーを作成します。

高解像度メトリクスによる高速自動スケーリング

デフォルトでは、Amazon ECS は CPUUtilization および MemoryUtilization メトリクスを 60 秒の解像度で発行します。20 秒の解像度を設定して、サービスの自動スケーリングを高速化できます。

ターゲット追跡スケーリングポリシーには、以下の高解像度の事前定義済みメトリクスが利用可能です。

PredefinedMetricType	説明
ECSServiceAverageCPUUtilizationHighResolution	20 秒の解像度での平均 CPU 使用率
ECSServiceAverageMemoryUtilizationHighResolution	20 秒の解像度での平均メモリ使用率

高解像度メトリクスを使用して新しいサービスを作成し、自動スケーリングを高速化する

コンソール

1. [クラスター] ページで、サービスを作成するクラスターを選択します。
2. [サービス] タブで、[作成] を選択します。
3. 「サービスの作成」フォームの「モニタリング設定」セクションで、「メトリクス構成の追加」を選択します。
4. メトリクス名 で、高解像度で収集するメトリクスを選択します。[CPUUtilization] または [MemoryUtilization] を選択します。
5. 解像度 (秒) で、20 を選択します。
6. (オプション) 両方のメトリクスを高解像度で追加するには、メトリクス設定を再度追加 を選択し、他のメトリクスに対して前の 2 つのステップを繰り返します。
7. サービスの自動スケーリング セクションで、サービスの自動スケーリングを使用する を選択します。
8. [タスクの最小数] に、サービスの自動スケーリングで使用するタスクの下限数を入力します。必要な数がこの数を下回ることはありません。
9. [タスクの最大数] に、サービスの自動スケーリングで使用するタスクの上限数を入力します。必要な数がこの数を超えることはありません。

10. [スケーリングポリシータイプ] で [ターゲットの追跡] を選択します。
11. メトリクスタイプ で、対応する高解像度メトリクスを選択します。たとえば、CPUUtilization を 20 秒の解像度で設定した場合は ECSServiceAverageCPUUtilizationHighResolution を選択し、MemoryUtilization を 20 秒の解像度で設定した場合は ECSServiceAverageMemoryUtilizationHighResolution を選択してください。
12. ターゲット値 には、ターゲット使用率を入力します。サービスの自動スケーリングは、平均使用率が目標値になるまで、または指定した最大タスク数に達するまで、キャパシティをスケールアウトします。
13. [追加設定] で、以下の項目を設定します。
    a. [スケールアウトのクールダウン期間] には、前回のスケールアウトアクティビティが有効になるまで待機する時間を秒単位で入力します。
    b. [スケールインのクールダウン期間] に、スケールインアクティビティが完了してから別のスケールインアクティビティが開始されるまでの時間を秒単位で入力します。
14. 作成 を選択して サービスを作成します。

AWS CLI

1. create-service コマンドを使用して、モニタリング設定でサービスを作成します。

```
aws ecs create-service --region us-east-1 \
    --cluster my-cluster \
    --service-name my-service \
    --task-definition my-app:1 \
    --desired-count 2 \
    --monitoring
 "metricConfigurations=[{metricNames=[CPUUtilization,MemoryUtilization],resolutionSeconds=2
```

2. register-scalable-target コマンドを使用して、スケーラブルなターゲットとしてサービスを登録します。

```
aws application-autoscaling register-scalable-target \
    --service-namespace ecs \
    --resource-id service/my-cluster/my-service \
    --scalable-dimension ecs:service:DesiredCount \
    --min-capacity 1 \
    --max-capacity 10 \
```

```
    --region us-east-1
```

3. put-scaling-policy コマンドを使用して、高解像度の事前定義されたメトリクスを使用してターゲット追跡スケーリングポリシーを作成します。

```
aws application-autoscaling put-scaling-policy \
    --service-namespace ecs \
    --resource-id service/my-cluster/my-service \
    --scalable-dimension ecs:service:DesiredCount \
    --policy-name my-highres-cpu-policy \
    --policy-type TargetTrackingScaling \
    --target-tracking-scaling-policy-configuration '{
        "TargetValue": 50.0,
        "PredefinedMetricSpecification": {
            "PredefinedMetricType": "ECSServiceAverageCPUUtilizationHighResolution"
        }
    }' \
    --region us-east-1
```

4. describe-scaling-policies コマンドを使用して、ポリシーが作成されたことを確認します。

```
aws application-autoscaling describe-scaling-policies \
    --service-namespace ecs \
    --resource-id service/my-cluster/my-service \
    --scalable-dimension ecs:service:DesiredCount \
    --region us-east-1
```

既存のサービスを更新して高解像度メトリクスを使用し、より高速な自動スケーリングを実現します

既存のサービスを更新して高解像度のメトリクスを使用し、サービスの自動スケーリングを高速化するプロセスは、2 つのステップで構成されます。まず、サービスデプロイをトリガーするモニタリング解像度を更新する必要があります。デプロイが正常に完了し、すべてのタスクが 20 秒の解像度でメトリクスを出力したら、高解像度のスケーリングポリシーを設定できます。

> **Important**
>
> 以前のサービスで 60 秒のメトリクス解像度を使用していた場合、スケーリングポリシー用の高解像度メトリクスは、サービスを 20 秒のメトリクス解像度を使用するように更新し、サービスのデプロイが完了した後にのみ、Amazon ECS コンソールで利用可能になります。

コンソール

ステップ 1: モニタリング解像度を更新する

1. [クラスター] ページで、更新するサービスが含まれているクラスターを選択します。
2. サービス セクションで、サービスを選択し、更新 を選択します。
3. 更新サービスフォームの Monitoring configuration セクションで、Add metric configuration を選択します。
4. メトリクス名 で、高解像度で収集するメトリクスを選択します。[CPUUtilization] または [MemoryUtilization] を選択します。
5. 解像度 (秒) で、20 を選択します。
6. (オプション) 両方のメトリクスを高解像度で追加するには、メトリクス設定を再度追加 を選択し、他のメトリクスに対して前の 2 つのステップを繰り返します。
7. [更新] を選択して変更を送信します。これにより、サービスのデプロイがトリガーされます。
8. デプロイが完了するまで待ちます。デプロイ タブのサービスの詳細ページで進行状況をモニタリングできます。

> **Note**
>
> サービスの自動スケーリングセクションでは、デプロイが成功し、すべてのタスクが 20 秒間隔でデータを出力するようになるまで、高解像度スケーリングメトリクスのオプションはグレー表示されます。

ステップ 2: 高解像度スケーリングポリシーを設定する

デプロイが完了したら、次のいずれかの方法を使用してスケーリングポリシーを設定します。

オプション A: サービスの詳細ページ

1. サービスの詳細ページで、[サービスの自動スケーリング] タブを選択します。
2. スケーリングポリシーの作成 (または既存のポリシーの編集) を選択します。
3. [スケーリングポリシータイプ] で [ターゲットの追跡] を選択します。
4. メトリクスタイプ で、高解像度メトリクスを選択します。例えば、ECSServiceAverageCPUUtilizationHighResolution、ECSServiceAverageMemoryU
です。

5. ターゲット値 には、ターゲット使用率を入力します。
6. [追加設定] で、以下の項目を設定します。
   a. [スケールアウトのクールダウン期間] には、前回のスケールアウトアクティビティが有効になるまで待機する時間を秒単位で入力します。
   b. [スケールインのクールダウン期間] に、スケールインアクティビティが完了してから別のスケールインアクティビティが開始されるまでの時間を秒単位で入力します。
7. [スケーリングポリシーを作成する] を選択します。

オプション B: サービスフォームの更新

1. [サービスの詳細] ページで、[更新] を選択します。
2. サービスの自動スケーリング セクションで、サービスの自動スケーリングを使用する (まだ有効になっていない場合) を選択します。
3. 最小タスク数 には、下限を入力します。
4. 最大タスク数 には、上限を入力します。
5. [スケーリングポリシータイプ] で [ターゲットの追跡] を選択します。
6. メトリクスタイプ で、高解像度メトリクスを選択します (デプロイの完了後に有効になりました) 。
7. ターゲット値 には、ターゲット使用率を入力します。
8. 追加設定で、スケールアウトクールダウン期間 と スケールインクールダウン期間 を設定します。
9. [更新] を選択して変更を送信します。

AWS CLI

ステップ 1: モニタリング解像度を更新する

1. update-service コマンドを使用して、モニタリング設定でサービスを更新します。

```
aws ecs update-service --region us-east-1 \
    --cluster my-cluster \
    --service my-service \
    --monitoring
 "metricConfigurations=[{metricNames=[CPUUtilization,MemoryUtilization],resolutionSeconds=2
```

2. デプロイが完了するまで待ちます。describe-services コマンドを使用し、デプロイのステータスを確認することで、進行状況をモニタリングできます。

```
aws ecs describe-services --region us-east-1 \
    --cluster my-cluster \
    --services my-service
```

3. デプロイが完了したら、サービスリビジョンを説明することで、監視設定が有効になっていることを確認します。describe-services 出力からアクティブなサービスリビジョン ARN を取得し、以下を実行します。

```
aws ecs describe-service-revisions --region us-east-1 \
    --service-revision-arns arn:aws:ecs:us-east-1:012345678910:service-revision/my-cluster/my-service/1234567890123456789
```

モニタリング設定がレスポンスに表示されることを確認します。

```
{
    "serviceRevisions": [
        {
            ...
            "monitoring": {
                "metricConfigurations": [
                    {
                        "metricNames": ["CPUUtilization", "MemoryUtilization"],
                        "resolutionSeconds": 20
                    }
                ]
            }
        }
    ]
}
```

ステップ 2: 高解像度スケーリングポリシーを設定する

1. まだ登録されていない場合はregister-scalable-target コマンドを使用して、サービスをスケーラブルターゲットとして登録します。

```
aws application-autoscaling register-scalable-target \
```

```
    --service-namespace ecs \
    --resource-id service/my-cluster/my-service \
    --scalable-dimension ecs:service:DesiredCount \
    --min-capacity 1 \
    --max-capacity 10 \
    --region us-east-1
```

2. put-scaling-policy コマンドを使用して、高解像度の事前定義されたメトリクスを使用してターゲット追跡スケーリングポリシーを作成します。以下の例では、50% の CPU 使用率を対象としています。

```
aws application-autoscaling put-scaling-policy \
    --service-namespace ecs \
    --resource-id service/my-cluster/my-service \
    --scalable-dimension ecs:service:DesiredCount \
    --policy-name my-highres-cpu-policy \
    --policy-type TargetTrackingScaling \
    --target-tracking-scaling-policy-configuration '{
        "TargetValue": 50.0,
        "PredefinedMetricSpecification": {
            "PredefinedMetricType": "ECSServiceAverageCPUUtilizationHighResolution"
        }
    }' \
    --region us-east-1
```

3. ポリシーが作成されたことを確認します。

```
aws application-autoscaling describe-scaling-policies \
    --service-namespace ecs \
    --resource-id service/my-cluster/my-service \
    --scalable-dimension ecs:service:DesiredCount \
    --region us-east-1
```

高速サービス自動スケーリングを検証する

パート 1: 高解像度メトリクスが公開されていることを確認する

1. [CloudWatch console] (CloudWatch のコンソール) に移動する。
2. ナビゲーションペインで、[メトリクス] 、[すべてのメトリクス] の順に選択します。
3. AWS/ECS 名前空間を選択します。

4. ClusterName, ServiceName を選択します。
5. サービスの CPUUtilization および MemoryUtilization メトリクスを見つけます。
6. 期間を 20 秒 に設定します。データポイントは 20 秒間隔で表示され、高解像度メトリクスがアクティブであることを確認します。

パート 2: スケーリングポリシーが高解像度メトリクスを使用していることを確認する

1. サービスの詳細ページで、[サービスの自動スケーリング] タブを選択します。
2. 既存のスケーリングポリシーを表示し、メトリクスが高解像度バリアント (ECSServiceAverageCPUUtilizationHighResolution または ECSServiceAverageMemoryUtilizationHighResolution) に設定されていることを確認します。
3. または、 CLI を使用して検証します。

```
aws application-autoscaling describe-scaling-policies \
    --service-namespace ecs \
    --resource-id service/my-cluster/my-service \
    --scalable-dimension ecs:service:DesiredCount \
    --region us-east-1
```

レスポンスの PredefinedMetricType が高解像度メトリクスに設定されていることを確認します。

考慮事項

- 設定変更をモニタリングすると、新しいサービスリビジョンが作成され、サービスのデプロイが必要になります。高解像度スケーリングメトリクスは、デプロイが完了し、すべてのタスクが 20 秒の解像度で出力されるまで使用できません。
- 高解像度メトリクスは、Application Load Balancer と Network Load Balancer を使用するサービスでサポートされています。ターゲットグループを使用しない Classic Load Balancer を使用するサービスはサポートされていません。
- 高解像度メトリクスは、CODE_DEPLOY または EXTERNAL デプロイコントローラータイプを使用するサービスではサポートされていません。
- DescribeServiceRevisions を使用して、サービスのモニタリング設定を表示できます。モニタリング設定は CreateService または UpdateService レスポンスでは返されません。

CloudWatch アラームに基づく定義済みの増分を使用して Amazon ECS サービスをスケールする

ステップスケーリングポリシーを使用して、スケーリングプロセスを呼び出す CloudWatch アラームを作成および管理します。アラームに違反すると、Amazon ECS はそのアラームに関連付けられたスケーリングポリシーを開始します。ステップスケーリングポリシーは、ステップ調整と呼ばれる一連の調整を使用してタスクをスケールします。調整値の規模は、アラーム違反の大きさに応じて異なります。

- 違反が最初のしきい値を超えた場合、Amazon ECS は最初のステップの調整を適用します。
- 違反が 2 番目のしきい値を超えた場合、Amazon ECS は 2 番目のステップの調整を適用するというように続きます。

ターゲット追跡スケーリング ポリシーを使用して、ターゲットごとの平均 CPU 使用率や平均リクエスト数などのメトリクスに基づいてスケールすることを強くお勧めします。キャパシティーが増加すると減少し、キャパシティーが減少すると増加するメトリクスを使用すると、ターゲット追跡を使用してインスタンス数を比例的にスケールアウトしたり、タスク数を増やすことができます。これにより、Amazon ECS がアプリケーションの需要曲線に厳密に従うことが保証されます。

Amazon ECS サービスの自動スケーリングのステップスケーリングポリシーを作成する

ステップスケーリングポリシーを作成して、Amazon ECS がサービスで必要なタスク数を自動的に増減させます。ステップスケーリングは、ステップ調整と呼ばれる、超過アラームのサイズによって異なる一連のスケーリング調整値に基づいて実行されます。

コンソール

1. サービスの作成や更新に使用する標準の IAM アクセス権限に加えて、追加のアクセス権限が必要です。詳細については、「Amazon ECS のサービス自動スケーリングに必要な IAM アクセス許可」を参照してください。
2. ポリシーに使用するメトリクスを決定します。以下のメトリクスが利用可能です。
   - [ECSServiceAverageCPUUtilization] — サービスが使用する平均 CPU 使用率。
   - [ECSServiceAverageMemoryUtilization] — サービスが使用する平均メモリ使用率。
   - [ALBRequestCountPerTarget] – タスクが受信する 1 分あたりの理想的な平均リクエスト数。
3. メトリクスの CloudWatch アラームを作成します。詳細については、Amazon CloudWatch ユーザーガイドの「静的しきい値に基づいて CloudWatch アラームを作成する」を参照してください。

4. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
5. [クラスター] ページで、クラスターを選択します。
6. [クラスターの詳細] ページの [サービス] セクションで、サービスを選択します。

   [サービス詳細] ページが表示されます。
7. [タスク数を設定する] を選択します。
8. [Amazon ECS サービスのタスク数] で、[自動スケーリングを使用する] を選択します。

   [タスク数セクション] が表示されます。

   a. [タスクの最小数] に、サービスの自動スケーリングで使用するタスクの下限数を入力します。必要な数がこの数を下回ることはありません。
   b. [最大] に、サービスの自動スケーリングで使用するタスクの上限数を入力します。必要な数がこの数を超えることはありません。
   c. [保存] を選択します。

      [ポリシー] ページが表示されます。
9. [スケーリングポリシーを作成する] を選択します。

   [ポリシーを作成する] ページが表示されます。
10. [スケーリングポリシータイプ] で [ステップスケーリング] を選択します。
11. スケールアウトプロパティを設定します。[タスクを追加するステップ] で、次を実行します。

    a. [Policy Name] (ポリシー名) にこのポリシーの名前を入力します。
    b. [CloudWatch アラーム名] で、CloudWatch アラームを選択します。
    c. [メトリクスの集計タイプ] で、選択したメトリクスを定義されたしきい値と比較する方法を選択します。
    d. [調整タイプ] で、調整がタスク数の変化に基づくか、タスクの割合の変化に基づくかを選択します。
    e. [実行するアクション] で、実行するアクションの値を入力します。

       [ステップを追加] を選択して、追加のアクションを追加します。
12. スケールインプロパティを設定します。[タスクを削除するステップ] で、次を実行します。

    a. [Policy Name] (ポリシー名) にこのポリシーの名前を入力します。
    b. [CloudWatch アラーム名] で、CloudWatch アラームを選択します。

c. [メトリクスの集計タイプ] で、選択したメトリクスを定義されたしきい値と比較する方法を選択します。

d. [調整タイプ] で、調整がタスク数の変化に基づくか、タスクの割合の変化に基づくかを選択します。

e. [実行するアクション] で、実行するアクションの値を入力します。

[ステップを追加] を選択して、追加のアクションを追加します。

13. [クールダウン期間] には、前回のスケーリングアクティビティが有効になるまで待機する時間を秒単位で入力します。追加ポリシーの場合、スケールアウトアクティビティの終了後、スケーリングポリシーによってスケールインアクティビティがブロックされ、一度にスケールアウトできるタスクの数が制限されます。削除ポリシーの場合、これはスケールインアクティビティの後、別のスケールインアクティビティが開始されるまでに経過する必要がある時間です。

14. [スケーリングポリシーを作成する] を選択します。

AWS CLI

1. register-scalable-target コマンドを使用して、スケーラブルなターゲットとして Amazon ECS サービスを登録します。

2. put-scaling-policy コマンドを使用して、スケーリングポリシーを作成します。

スケジュールされたアクションを使用して Amazon ECS サービスをスケールする

スケジュールされたスケーリングでは、特定の時間にタスク数を増減するスケジュールアクションを作成することで、予測可能な負荷の変化に基づいてアプリケーションの自動スケーリングを設定できます。これにより、予測可能な負荷の変化に合わせてアプリケーションを事前対応的にスケーリングできます。

これらのスケジュールされたスケーリングアクションにより、コストとパフォーマンスを最適化できます。アプリケーションには、週半ばのトラフィックのピークを処理するのに十分な数のタスクがありますが、それ以外の時間帯にタスクを過剰にプロビジョニングすることはありません。

スケジュールされたスケーリングとスケーリングポリシーを併用して、スケーリングに事前対応型アプローチと即応型アプローチの両方のメリットを得ることができます。スケジュールされたスケーリングアクションの実行後、スケーリングポリシーはタスクをさらにスケールするかどうかの判断を引き続き行うことができます。これは、アプリケーションの負荷を処理するために十分なタスク数を確保する上で役立ちます。アプリケーションは需要に合わせてスケールしますが、現行のキャパシティ

は、スケジュールされたアクションによって設定された最小タスク数と最大タスク数の範囲内に収まる必要があります。

スケジュールスケーリングは AWS CLI を使用して設定できます。スケジュールに基づくスケーリングの詳細については、「Application Auto Scaling ユーザーガイド」の「スケジュールに基づくスケーリング」を参照してください。

Amazon ECS サービスの自動スケーリングのスケジュールされたアクションを作成する

スケジュールされたアクションを作成して、Amazon ECS でサービスが実行するタスク数を日付と時刻に基づいて増減させます。

コンソール

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
2. [クラスター] ページで、クラスターを選択します。
3. [クラスターの詳細] ページの [サービス] セクションで、サービスを選択します。

   [サービス詳細] ページが表示されます。
4. [サービスの自動スケーリング] を選択します。

   [サービスの自動スケーリング] ページが表示されます。
5. [サービスの自動スケーリング] を設定していない場合は、[タスク数の設定] を選択します。

   Amazon ECS サービスタスク数 のセクションが表示されます。

   Amazon ECS サービスタスク数 で、[サービスの自動スケーリングを使用してサービスの必要なタスク数を調整する] を選択します。

   [タスク数セクション] が表示されます。

   a. [タスクの最小数] に、サービスの自動スケーリングで使用するタスクの下限数を入力します。必要な数がこの数を下回ることはありません。
   b. [最大] に、サービスの自動スケーリングで使用するタスクの上限数を入力します。必要な数がこの数を超えることはありません。
   c. [保存] を選択します。

      [ポリシー] ページが表示されます。
6. [スケジュールされたアクション] を選択し、[作成] を選択します。

[スケジュールされたアクションを作成] ページが表示されます。

7. バケット名に、一意の名前を入力します。
8. [Time zone (タイムゾーン)] でタイムゾーンを選択。

   リストされているすべてのタイムゾーンは、IANA タイムゾーンデータベースから取得されます。詳細については、「List of tz database time zones」を参照してください。
9. [開始時刻] で、アクションが開始される [日付] と [時刻] を入力します。

   定期的なスケジュールを選択した場合、開始時間によって、定期的なシリーズの最初のスケジュールされたアクションが実行されるタイミングが定義されます。
10. [Recurrence (反復)] で、使用可能なオプションの 1 つを選択します。

    - 反復スケジュールに基づいてスケールするには、Amazon ECS がスケジュールされたアクションを実行する頻度を選択します。
      - [レート] で始まるオプションを選択した場合、cron 式が作成されます。
      - [Cron] を選択した場合は、いつアクションを実行するかを Cron 式を入力します。
    - 1 回だけスケールするには、[1 回] を選択します。
11. [タスク調整] で、次の操作を行います。

    - [最小] で、サービスが実行する最小タスク数を入力します。
    - [最大] で、サービスが実行する最大タスク数を入力します。
12. [スケジュールされたアクションの作成] を選択してください。

CLI

次のように AWS CLI を使用して、サービスのスケジュールされたスケーリングポリシーを設定します。各###############を独自の情報に置き換えます。

例: 1 回のみスケールするには

次の put-scheduled-action コマンドを、`--start-time "YYYY-MM-DDThh:mm:ssZ"` と `--MinCapacity` および `--MaxCapacity` オプションのいずれかまたは両方と併用します。

```
aws application-autoscaling put-scheduled-action --service-namespace ecs \
  --resource-id service/my-cluster/my-service \
  --scheduled-action-name my-one-time-schedule \
  --start-time 2021-01-30T12:00:00 \
```

```
  --scalable-target-action MinCapacity=3,MaxCapacity=10
```

例: 定期的なスケーリングをスケジュールするには

次の put-scheduled-action コマンドを使用します。*user-input* を独自の値に置き換えます。

```
aws application-autoscaling put-scheduled-action --service-namespace ecs \
  --resource-id service/my-cluster/my-service \
  --scheduled-action-name my-recurring-action \
  --schedule "rate(5 hours)" \
  --start-time 2021-01-30T12:00:00 \
  --end-time 2021-01-31T22:00:00 \
  --scalable-target-action MinCapacity=3,MaxCapacity=10
```

指定された繰り返しスケジュールは、UTC タイムゾーンに基づいて実行されます。別のタイムゾーンを指定するには、次の例のように、--time-zone オプションと IANA タイムゾーンの名前を含めます。

```
--time-zone "America/New_York"
```

詳細については、「List of tz database time zones」を参照してください。

履歴パターンを使用して予測スケーリングで Amazon ECS サービスをスケールする

予測スケーリングは、トラフィックフローからの過去の負荷データを調べて、日次または週次のパターンを分析します。次に、この分析を使用して将来のニーズを予測し、必要に応じてサービスのタスクをプロアクティブに増やします。

予測自動スケーリングは、以下の状況で最も役立ちます。

- 周期的なトラフィック - 通常の営業時間にリソースの使用が増加し、夜間や週末にはリソースの使用が減少する。
- オンとオフを繰り返すワークロードパターン - バッチ処理、テスト、定期的なデータ分析など。
- 初期化に時間がかかるアプリケーション - スケールアウトイベント中のアプリケーションのパフォーマンスに影響を及ぼし、顕著なレイテンシーを引き起こす可能性がある。

アプリケーションの初期化に時間がかかり、トラフィックが規則的なパターンで増加する場合は、予測スケーリングの使用を検討する必要があります。ターゲット追跡やステップスケーリングなどの動的スケーリングポリシーを単独で使用するのではなく、予測される負荷に応じてタスク数を事前に増

やすことで、より迅速にスケールできます。予測スケーリングにより、タスク数が過剰にプロビジョニングされる可能性を回避できるため、コストを削減できる場合もあります。

例えば、営業時間中の使用率が高く、夜間の使用量が少ないアプリケーションを考えてみましょう。各営業日の開始時に、予測スケーリングにより、トラフィックが最初に流入する前にタスクをスケールアウトできます。これにより、使用率の低い期間から高い使用率の期間に移行するときに、アプリケーションの高可用性とパフォーマンスを維持するのに役立ちます。トラフィックの変化に動的スケーリングが反応するのを待つ必要はありません。また、アプリケーションの負荷パターンを確認し、スケジュールされたスケーリングを使用して適切な量のタスクをスケジュールするために時間を費やす必要もありません。

予測スケーリングは、基盤となるコンピューティングキャパシティ (EC2 や Fargate など) のスケーリングとは別にサービスのタスクをスケーリングするサービスレベルの機能です。Fargate の場合、AWS はタスク要件に基づいて基盤となる容量を管理して動的にスケーリングします。EC2 キャパシティーの場合、Auto Scaling グループキャパシティープロバイダーを使用して、タスクのスケーリング要件に基づいて基盤となる EC2 インスタンスを自動的にスケーリングできます。

内容

- Amazon ECS における予測スケーリングの仕組み
- Amazon ECS サービスの自動スケーリングの予測スケーリングポリシーを作成する
- Amazon ECS の予測スケーリングポリシーの評価
- スケジュールされたアクションを使用して Amazon ECS の予測値を上書きする
- Amazon ECS のカスタムメトリクスを使用した高度な予測スケーリングポリシー

Amazon ECS における予測スケーリングの仕組み

ここでは、予測スケーリングの使用に関する考慮事項、その仕組み、制限事項について説明します。

予測スケーリングを使用する際の考慮事項

- 予測スケーリングがワークロードに適していることを確認する必要があります。これを確認するには、予測専用モードでスケーリングポリシーを設定し、コンソールが推奨する内容を確認します。予測スケーリングの使用を開始する前に、予測とレコメンデーションを評価する必要があります。
- 予測スケーリングが予測を開始するには、少なくとも 24 時間以上の履歴データが必要です。使用可能な履歴データが多いほど、予測がより効果的になり、2 週間分の履歴データがあると理想的です。また、Amazon ECS サービスを削除して新しいサービスを作成する場合、予測スケーリングによって新しい予測が生成されるまで 24 時間待つ必要があります。これを高速化する 1 つの方法

は、カスタムメトリクスを使用して、古い Amazon ECS サービスと新しい Amazon ECS サービス全体のメトリクスを集約することです。

- アプリケーションのすべての負荷を正確に表し、スケーリングが最も重要なアプリケーションの側面である負荷メトリクスを選択します。
- 予測スケーリングを使用した動的スケーリングにより、アプリケーションの需要に迅速に対応でき、需要が少ないときにはスケールインし、予期しないトラフィックの増加時にはスケールアウトすることができます。複数のスケーリングポリシーがアクティブな場合、各ポリシーによって希望するタスク数が個別に決定され、希望するタスク数はそれらの最大値に設定されます。
- ターゲット追跡やステップスケーリングなどの動的スケーリングポリシーと共に予測スケーリングを使用すると、リアルタイムパターンと履歴パターンの両方に基づいてアプリケーションをスケーリングできます。それ自体では、予測スケーリングはタスクをスケールインしません。
- `register-scalable-target` API を呼び出すときにカスタムロールを使用すると、予測スケーリングポリシーが SLR を有効にした場合にのみ機能することを示す エラーが表示されることがあります。この場合、role-arn なしで再度 `register-scalable-target` を呼び出す必要があります。スケーラブルターゲットを登録する際に SLR を使用し、`put-scaling-policy` API を呼び出します。

予測スケーリングの仕組み

予測スケーリングを使用するには、モニタリングおよび分析する CloudWatch メトリクスを指定する予測スケーリングポリシーを作成します。予測スケーリングが将来の値の予測を開始するには、このメトリクスに 24 時間以上のデータが必要です。

ポリシーを作成すると、予測スケーリングは最長過去 14 日間のメトリクスデータの分析を開始し、パターンを特定します。この分析は、今後 48 時間の必要量の時間ごとの予測を生成するために使用されます。最新の CloudWatch データを使用して、6 時間ごとに予測が更新されます。新しいデータを取得すると、予測スケーリングは将来の予測の正確性を継続的に向上させます。

最初に予測スケーリングを有効にしたときは、予測のみモードで実行されます。このモードでは予測を生成しますが、それらの予測に基づいて Amazon ECS サービスをスケールすることはありません。これにより、予測の正確性と適合性を評価できます。`GetPredictiveScalingForecast` API オペレーションまたは AWS マネジメントコンソールを使用して、予測データを表示します。

予測スケーリングの使用を開始する場合は、スケーリングポリシーを予測とスケーリングモードに切り替えます。このモードでは、次のことが発生します。

Amazon ECS サービスは、デフォルトでは、その時間の予測に基づいて各時間の開始時にスケールされます。`PutScalingPolicy` API オペレーションで `SchedulingBufferTime` プロパティを使

用して、早く開始することを選択できます。これにより、新しいタスクは予測される需要よりも早く起動し、トラフィックを処理する準備が整うまでの時間を確保できます。

最大タスク制限

スケーリングのために Amazon ECS サービスを登録する場合、サービスごとに起動できる最大タスク数を定義します。デフォルトでは、スケーリングポリシーが設定されている場合、タスク数をその最大制限を超えて増やすことはできません。

ただし、予測が Amazon ECS サービスの最大タスク数に近づいた場合、または超えた場合に、サービスの最大タスク数を自動的に増やすことを許可できます。

> **Warning**
>
> 最大タスク数を自動的に増やす場合は注意してください。これにより、増加した最大タスク数がモニタリングおよび管理されていない場合、意図したよりも多くのタスクが起動される可能性があります。増加した最大タスク数は、手動で更新するまで Amazon ECS サービスの新しい通常の最大タスク数になります。最大タスク数は、自動的に元の最大タスク数まで減少しません。

サポート対象のリージョン

- 米国東部 (バージニア北部)
- 米国東部 (オハイオ)
- 米国西部 (北カリフォルニア)
- 米国西部 (オレゴン)
- アフリカ (ケープタウン)
- アジアパシフィック (香港)
- アジアパシフィック (ジャカルタ)
- アジアパシフィック (ムンバイ)
- アジアパシフィック (大阪)
- アジアパシフィック (ソウル)
- アジアパシフィック (シンガポール)
- アジアパシフィック (シドニー)

- アジアパシフィック (東京)
- カナダ (中部)
- 中国 (北京)
- 中国 (寧夏)
- 欧州 (フランクフルト)
- 欧州 (アイルランド)
- 欧州 (ロンドン)
- 欧州 (ミラノ)
- 欧州 (パリ)
- 欧州 (ストックホルム)
- 中東 (バーレーン)
- 南米 (サンパウロ)
- AWS GovCloud (米国東部)
- AWS GovCloud (米国西部)

Amazon ECS サービスの自動スケーリングの予測スケーリングポリシーを作成する

予測スケーリングポリシーを作成して、Amazon ECS でサービスが実行するタスク数を履歴データに基づいて増減させます。

> **Note**
>
> 新しいサービスは、予測を生成する前に 24 時間以上のデータを提供する必要があります。

コンソール

1. サービスの作成や更新に使用する標準の IAM アクセス権限に加えて、追加のアクセス権限が必要です。詳細については、「Amazon ECS のサービス自動スケーリングに必要な IAM アクセス許可」を参照してください。
2. ポリシーに使用するメトリクスを決定します。以下のメトリクスが利用可能です。
   - [ECSServiceAverageCPUUtilization] — サービスが使用する平均 CPU 使用率。
   - [ECSServiceAverageMemoryUtilization] — サービスが使用する平均メモリ使用率。

- [ALBRequestCountPerTarget] – タスクが受信する 1 分あたりの理想的な平均リクエスト数。

または、カスタムメトリクスを使用することもできます。次の値を定義する必要があります。

- 負荷 - アプリケーションのすべての負荷を正確に表し、スケーリングが最も重要なアプリケーションの側面であるメトリクス。

- スケーリングメトリクス - アプリケーションにとって理想的な使用率を予測するために最適な予測子。

3. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
4. [クラスター] ページで、クラスターを選択します。
5. [クラスターの詳細] ページの [サービス] セクションで、サービスを選択します。

   [サービス詳細] ページが表示されます。
6. [サービスの自動スケーリング] を選択し、[タスクの数を設定] を選択します。
7. [Amazon ECS サービスのタスク数] で、[自動スケーリングを使用する] を選択します。

   [タスク数セクション] が表示されます。

   a. [タスクの最小数] に、サービスの自動スケーリングで使用するタスクの下限数を入力します。必要な数がこの数を下回ることはありません。
   b. [最大] に、サービスの自動スケーリングで使用するタスクの上限数を入力します。必要な数がこの数を超えることはありません。
   c. [保存] を選択します。

      [ポリシー] ページが表示されます。
8. [スケーリングポリシーを作成する] を選択します。

   [ポリシーを作成する] ページが表示されます。
9. [スケーリングポリシータイプ] で [予測スケーリング] を選択します。
10. [Policy Name] (ポリシー名) にこのポリシーの名前を入力します。
11. [メトリクスペア] で、オプションのリストからメトリクスを選択します。

    [Application Load Balancer request count per target] (ターゲットあたりの Application Load Balancer リクエスト数) を選択した場合、[Target group] (ターゲットグループ) のターゲットグループを選択します。[ターゲットあたりの Application Load Balancer リクエスト数]

は、Application Load Balancer ターゲットグループをサービスにアタッチしている場合にのみサポートされます。

[カスタムメトリクスペア] を選択した場合、[負荷のメトリクス] と [スケーリングのメトリクス] のリストから個々のメトリクスを選択します。

12. [ターゲット使用率] には、Amazon ECS が維持するタスクの割合のターゲット値を入力します。サービスの自動スケーリングは、平均使用率が目標使用率になるまで、または指定した最大タスク数に達するまで、キャパシティをスケールアウトします。
13. [スケーリングポリシーを作成する] を選択します。

AWS CLI

次のように AWS CLI を使用して、Amazon ECS サービスの予測スケーリングポリシーを設定します。各###############を独自の情報に置き換えます。

指定できる CloudWatch メトリクスの詳細については、「Amazon EC2 Auto Scaling API リファレンス」の「PredictiveScalingMetricSpecification」を参照してください。

例 1: メモリが事前定義された予測スケーリングポリシー。

以下は、メモリ設定が事前定義されたポリシーの例です。

```
cat policy.json
{
    "MetricSpecifications": [
        {
            "TargetValue": 40,
            "PredefinedMetricPairSpecification": {
                "PredefinedMetricType": "ECSServiceMemoryUtilization"
            }
        }
    ],
    "SchedulingBufferTime": 3600,
    "MaxCapacityBreachBehavior": "HonorMaxCapacity",
    "Mode": "ForecastOnly"
}
```

以下の例は、設定ファイルを指定し put-scaling-policy コマンドを実行して、ポリシーを作成する方法を示しています。

```
aws application-autoscaling put-scaling-policy \
```

```
--service-namespace ecs \
--region us-east-1 \
--policy-name predictive-scaling-policy-example \
--resource-id service/MyCluster/test \
--policy-type PredictiveScaling \
--scalable-dimension ecs:service:DesiredCount \
--predictive-scaling-policy-configuration file://policy.json
```

成功した場合、このコマンドはポリシーの ARN を返します。

```
{
    "PolicyARN": "arn:aws:autoscaling:us-
east-1:012345678912:scalingPolicy:d1d72dfe-5fd3-464f-83cf-824f16cb88b7:resource/ecs/
service/MyCluster/test:policyName/predictive-scaling-policy-example",
    "Alarms": []
}
```

例 2: CPU が事前定義された予測スケーリングポリシー。

以下は、CPU 設定が事前定義されたポリシーの例です。

```
cat policy.json
{
    "MetricSpecifications": [
        {
            "TargetValue": 0.00000004,
            "PredefinedMetricPairSpecification": {
                "PredefinedMetricType": "ECSServiceCPUUtilization"
            }
        }
    ],
    "SchedulingBufferTime": 3600,
    "MaxCapacityBreachBehavior": "HonorMaxCapacity",
    "Mode": "ForecastOnly"
}
```

以下の例は、設定ファイルを指定し put-scaling-policy コマンドを実行して、ポリシーを作成する方法を示しています。

```
aws aas put-scaling-policy \
--service-namespace ecs \
--region us-east-1 \
```

```
--policy-name predictive-scaling-policy-example \
--resource-id service/MyCluster/test \
--policy-type PredictiveScaling \
--scalable-dimension ecs:service:DesiredCount \
--predictive-scaling-policy-configuration file://policy.json
```

成功した場合、このコマンドはポリシーの ARN を返します。

```
{
    "PolicyARN": "arn:aws:autoscaling:us-
east-1:012345678912:scalingPolicy:d1d72dfe-5fd3-464f-83cf-824f16cb88b7:resource/ecs/
service/MyCluster/test:policyName/predictive-scaling-policy-example",
    "Alarms": []
}
```

Amazon ECS の予測スケーリングポリシーの評価

予測スケーリングポリシーを使用してサービスをスケールする前に、Amazon ECS コンソールでポリシーの推奨事項とその他のデータを確認します。予測が正確であることがわかるまで、予測スケーリングポリシーが実際のキャパシティをスケーリングすることが好ましくないため、これは重要です。

サービスが新しい場合は、最初の予測の作成に 24 時間かかります。

AWS が予測を作成するとき、履歴データを使用します。サービスに最新の履歴データがまだ十分にない場合、予測スケーリングは現在使用可能な履歴集計から作成された集計で予測を一時的にバックフィルすることがあります。予測は、ポリシーの作成日より最大 2 週間前にバックフィルされます。

予測スケーリングの推奨事項の表示

分析を効果的に行うには、サービスの自動スケーリングに比較対象となる予測スケーリングポリシーが少なくとも 2 つ必要です。(ただし、単一ポリシーの結果を確認することはできます) 複数のポリシーを作成するとき、1 つのメトリクスを使用するポリシーを異なるメトリクスを使用するポリシーと比較して評価できます。異なる目標値およびメトリクスの組み合わせによる影響を評価することもできます。予測スケーリングポリシーが作成された後、Amazon ECS は、どのポリシーがグループのスケーリングに適しているかについて、すぐに評価を開始します。

Amazon ECS コンソールで推奨事項を表示するには

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。

2. [クラスター] ページで、クラスターを選択します。
3. [クラスターの詳細] ページの [サービス] セクションで、サービスを選択します。

   [サービス詳細] ページが表示されます。
4. [サービスの自動スケーリング] を選択します。
5. 予測スケーリングポリシーを選択し、[アクション]、[予測スケーリング]、[レコメンデーションの表示] を選択します。

   ポリシーの詳細と推奨事項を表示できます。推奨事項は、予測スケーリングポリシーを使用しない場合よりも優れた性能を発揮するかどうかについて説明します。

   予測スケーリングポリシーがグループに適切かどうかが不明な場合、[可用性への影響] 列および [コストへの影響] 列を確認し、適切なポリシーを選択してください。各列の情報は、ポリシーの影響について説明します。

   - [可用性への影響]: ポリシーを使用しない場合と比較し、ワークロードを処理するために十分なタスクをプロビジョニングすることにより、ポリシーが可用性への悪影響を回避できるかどうかについて説明します。
   - [コストへの影響]: ポリシーを使用しない場合と比較し、タスクを過剰にプロビジョニングしないことにより、ポリシーがコストへの悪影響を回避できるかどうかについて説明します。過剰にプロビジョニングしすぎると、サービスが十分に活用されない、またはアイドル状態になり、コストへの影響が増す一方です。

   複数のポリシーがある場合、より低コストで最も可用性のメリットが高いポリシーの名前の横に [最良予測] タグが表示されます。可用性への影響がより重視されます。
6. (オプション) 推奨結果の必要な期間を選択するには、[評価期間] のドロップダウンから [2 日]、[1 週間]、または [2 週間] のいずれか希望する値を選択します。デフォルトでは、評価期間は過去の 2 週間です。評価期間が長いほど、推奨結果のデータポイントが増えます。ただし、需要が非常に高い時期の後など、負荷パターンが変化した場合、データポイントを追加しても結果が改善されない可能性があります。この場合、最新のデータを見ることでより焦点を絞った推奨事項を得ることができます。

> **Note**
>
> 推奨事項は [予測のみ] モードのポリシーに対してのみ生成されます。推奨機能は、評価期間中にポリシーが [予測のみ] モードのときにより効果的に機能します。ポリシーを [予測とス

ケーリング] モードで開始し、後で [予測のみ] モードに切り替える場合、そのポリシーの結果に偏りが生じる可能性があります。これは、ポリシーが既に実際のキャパシティに関与しているためです。

予測スケーリングのモニタリンググラフの確認

コンソールでは、以前の日、週間、月の予測を確認し、時間の経過と共にポリシーがどの程度機能しているか視覚化できます。また、ポリシーが実際のタスク数をスケールするかどうかを決定するとき、この情報を使用して予測の精度を評価できます。

Amazon ECS コンソールで予測スケーリングのモニタリンググラフを表示するには

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
2. [クラスター] ページで、クラスターを選択します。
3. [クラスターの詳細] ページの [サービス] セクションで、サービスを選択します。

   [サービス詳細] ページが表示されます。
4. [サービスの自動スケーリング] を選択します。
5. 予測スケーリングポリシーを選択し、[アクション]、[予測スケーリング]、[グラフの表示] を選択します。
6. [モニタリング] セクションでは、ポリシーの負荷およびキャパシティに関する過去および今後の予測を実際の値と比較できます。[負荷] グラフには、選択した負荷メトリクスの負荷予測および実際の値が表示されます。[キャパシティ] グラフには、ポリシーによって予測されたタスク数が表示されます。実際に起動されたタスク数も含まれます。縦線は履歴の値と今後の予測を区切っています。これらのグラフは、ポリシーの作成後にすぐ利用できます。
7. (オプション) グラフに表示される履歴データの量を変更するには、[評価期間] のドロップダウンから希望する値を選択します。評価期間はこのページのデータをはいかなる方法で変換することはありません。表示される履歴データの量のみを変更します。

[負荷] グラフのデータを比較

各水平線は、1 時間間隔で報告される異なる一連のデータポイントを表しています。

1. [実際に観測された負荷] は、選択した負荷メトリクスの SUM 統計を使用し、過去の 1 時間ごとの合計負荷を表示します。

2. [ポリシーによって予測される負荷] は、1 時間ごとの負荷予測を表示します。この予測は過去 2 週間分の実際の負荷観測に基づいています。

[キャパシティ] グラフのデータの比較

各水平線は、1 時間間隔で報告される異なる一連のデータポイントを表しています。

1. [実際に観測されたタスク数] には、Amazon ECS サービスの過去の実際のキャパシティが表示されます。これは、他のスケーリングポリシーや、選択した期間に有効な最小グループサイズによって異なります。
2. [ポリシーによって予測されるキャパシティ] には、ポリシーが [予測とスケーリング] モードになっているときに各時間の開始時に予想されるベースラインキャパシティが表示されます。
3. [推定される必要タスク数] には、スケーリングメトリクスを選択したターゲット値に維持するためのサービス内の理想的なタスク数が表示されます。
4. [最小タスク数] には、サービス内の最小タスク数が表示されます。
5. [最大キャパシティ] には、サービス内の最大タスク数が表示されます。

推定される必要キャパシティを計算するため、最初は各タスクが指定された目標値で均等に使用されていると仮定します。実際には、タスク数は均等に使用されません。ただし、使用率がタスク間で均等に分散されていると仮定することにより、必要なキャパシティの量を推定できます。次に、タスク数の要件は、予測スケーリングポリシーに使用したスケーリングメトリクスに反比例するように計算されます。つまり、タスク数が増加するにつれ、スケーリングメトリクスは同じ割合で減少します。例えば、タスク数が 2 倍になった場合、スケーリングメトリクスは半減します。

推定された必要キャパシティの計算式は、次のとおりです。

```
sum of (actualServiceUnits*scalingMetricValue)/(targetUtilization)
```

例えば、特定の時間の `actualServiceUnits` (`10`) および `scalingMetricValue` (`30`) を算出します。その後、予測スケーリングポリシー (`60`) で指定した `targetUtilization` を使用し、同じ時間に推定される必要キャパシティを計算します。これは `5` の値を返します。これは、スケーリングメトリクスの目標値とは正反比例し、キャパシティを維持するために必要なキャパシティの推定量は 5 であることを意味します。

> **Note**
>
> コスト削減およびアプリケーションの可用性を調整および改善するため、さまざまな手段が用意されています。

- ベースラインキャパシティに予測スケーリングを使用し、追加のキャパシティに動的スケーリングを使用します。動的スケーリングは予測スケーリングとは独立して動作し、現在の使用率に基づいてスケールインおよびスケールアウトを行います。まず、Amazon ECS は、スケジュールされていないスケーリングポリシーごとに推奨されるタスク数を計算します。次に、最も多くのタスクを提供するポリシーに基づいてスケーリングします。
- 負荷が減少したときにスケールインできるようにするには、サービスに、スケールイン部分を有効にした動的スケーリングポリシーが常に少なくとも 1 つ必要です。
- 最小キャパシティおよび最大キャパシティを制限しすぎないようにすることにより、スケーリングパフォーマンスを向上させることができます。推奨されるタスク数が最小キャパシティおよび最大キャパシティの範囲に収まらないポリシーは、スケールインおよびスケールアウトができなくなります。

CloudWatch による Amazon ECS の予測スケーリングメトリクスをモニタリングする

Amazon CloudWatch を使用して、予測スケーリング用にデータをモニタリングできます。予測スケーリングポリシーは、今後の負荷を予測するために使用されるデータを収集します。収集されたデータは、定期的に CloudWatch に自動的に保存され、ポリシーが時間の経過と共にどの程度機能しているかを視覚化するために使用できます。また、CloudWatch アラームを作成して、パフォーマンス指標が定義した制限を超えて変化したときに通知させることもできます。

履歴予測データの視覚化

予測スケーリングポリシーの負荷予測データは CloudWatch で表示でき、他の CloudWatch メトリクスに対する予測を 1 つのグラフで視覚化する場合に役立ちます。また、より広い時間範囲を表示することで、時間の経過に伴う傾向を確認することもできます。最大 15 か月間の履歴メトリクスにアクセスして、ポリシーの動作をより的確に把握できます。

CloudWatch コンソールを使用して履歴予測データを表示する方法

1. CloudWatch コンソールの https://console.aws.amazon.com/cloudwatch/ を開いてください。
2. ナビゲーションペインで、[Metrics] (メトリクス)、[All metrics] (すべてのメトリクス) の順に選択します。
3. [アプリケーションの自動スケーリング] メトリクス名前空間を選択します。
4. [予測スケーリングの負荷予測] を選択します。
5. 検索フィールドに、予測スケーリングポリシー名または Amazon ECS サービスグループ名を入力し、Enter キーを押して結果をフィルタリングします。

6. メトリクスをグラフ表示するには、メトリクスの横にあるチェックボックスを選択します。グラフの名前を変更するには、鉛筆アイコンを選択します。時間範囲を変更するには、事前定義済みの値を選択するか、[custom] を選択します。詳細については、「Amazon CloudWatch ユーザーガイド」の「メトリクスをグラフ化する」を参照してください。
7. 統計を変更するには、[Graphed metrics] タブを選択します。列見出しまたは個々の値を選択し、続いて各種統計を選択します。各メトリクスの任意の統計を選択できますが、すべての統計が [PredictiveScalingLoadForecast] メトリクスに有用なわけではありません。例えば、平均、最小、最大の各統計は有用ですが、合計の統計は有用ではありません。
8. グラフに別のメトリクスを追加するには、[Browse] (参照) で [All] (すべて) を選択し、追加したいメトリクスを見つけて、その横にあるチェックボックスをオンにします。最大 10 個のメトリクスを追加できます。
9. (オプション) このグラフを CloudWatch ダッシュボードに追加するには、[Actions] (アクション)、[Add to dashboard] (ダッシュボードに追加) の順に選択します。

Metric Math を使用して精度メトリクスを作成する

Metric Math により、複数の CloudWatch メトリクスをクエリし、数式を使用して、これらのメトリクスに基づく新しい時系列を作成できます。作成された時系列を CloudWatch コンソールで可視化でき、ダッシュボードに追加できます。メトリクス演算の詳細については、「Amazon CloudWatch ユーザーガイド」の「CloudWatch メトリクスでの数式の使用」を参照してください。

Metric Math を使用して、サービス自動スケーリングが予測スケーリングのために生成するデータを各種の方法でグラフ化できます。これにより、ポリシーのパフォーマンスを経時的にモニタリングし、メトリクスの組み合わせを改善できるかどうかを把握することができます。

例えば、Metric Math 式を使用して、平均絶対パーセント誤差 (MAPE) をモニタリングできます。MAPE メトリクスは、予測値と、特定の予測期間中に観測された実際の値の差をモニタリングするのに役立ちます。MAPE の値の変化は、アプリケーションの性質が変化するにつれて、ポリシーのパフォーマンスが経時的に低下しているかどうかを示します。MAPE の増加は、予測値と実際の値の差が大きいことを示します。

例: Metric Math 式

このタイプのグラフを使用するには、次の例に示すような Metric Math 式を作成します。

単一のメトリクスではなく、`MetricDataQueries` 用のメトリクスデータクエリ構造の配列があります。`MetricDataQueries` の各項目は、メトリクスを取得するか、数式を実行します。最初の項

目は、数式である e1 です。指定された式は、ReturnData パラメータを true に設定し、最終的に単一の時系列を生成します。他のすべてのメトリクスで、ReturnData 値は false です。

この例では、指定された式は実際の値と予測値を入力値として使用し、新しいメトリクス (MAPE) を返します。m1 は、実際の負荷値を含む CloudWatch メトリクスです (CPU 使用率が、my-predictive-scaling-policy という名前のポリシーに対して最初に指定された負荷メトリクスであると仮定)。m2 は、予測負荷値を含む CloudWatch メトリクスです。MAPE メトリクスの計算構文は次のとおりです。

(絶対値 ((実際の値 - 予測値)/(実際の値))) の平均

精度メトリクスを視覚化してアラームを設定する

精度メトリクスデータを視覚化するには、CloudWatch コンソールの [Metrics] (メトリクス) タブをクリックします。そこからデータをグラフ化できます。詳細については、「Amazon CloudWatch ユーザーガイド」の「CloudWatch グラフに数式を追加する」を参照してください。

[Metrics] (メトリクス) セクションから、モニタリングしているメトリクスにアラームを設定することもできます。[Graphed metrics] (グラフ化したメトリクス) タブで、[Actions] (アクション) 列にある [Create alarm] (アラームを作成) アイコンをクリックします。[Create alarm] (アラームを作成) アイコンは小さなベルです。詳細および通知オプションについては、「Amazon CloudWatch ユーザーガイド」の「メトリクス数式に基づく CloudWatch アラームの作成」と「アラームの変更をユーザーに通知する」を参照してください。

GetMetricData および PutMetricAlarm を使用して、Metric Math によって計算し、その出力に基づいてアラームを作成することもできます。

スケジュールされたアクションを使用して Amazon ECS の予測値を上書きする

予測計算では考慮できない将来のアプリケーション要件に関する追加情報がある場合があります。例えば、予測の計算では、今後のマーケティングイベントに必要なタスクが過小評価される可能性があります。スケジュールされたアクションを使用して、将来の期間中の予測を一時的に上書きできます。スケジュールされたアクションは、繰り返し実行することも、1 回限りの需要変動がある特定の日時に実行することもできます。

例えば、予測される数よりも多いタスク数でスケジュールされたアクションを作成できます。実行時に、Amazon ECS はサービス内の最小タスク数を更新します。予測スケーリングはタスクの数に合わせて最適化するので、予測値を超える最小タスクの数でスケジュールされたアクションが尊重されます。これにより、タスクの数が予想を下回らないようになります。予測の上書きを停止するには、2 番目にスケジュールされたアクションを使用して、最小タスク数を元の設定に戻します。

次の手順では、将来の期間中の予測を上書きするステップを示します。

トピック

- ステップ 1: (オプション) 時系列データを分析する
- ステップ 2: 2 つのスケジュールされたアクションを作成する

> **Important**
>
> このトピックでは、予測を上書きして、予測よりも大きなキャパシティにスケールしようとしていることを前提としています。予測スケーリングポリシーの干渉なしに一時的にタスク数を減らす必要がある場合は、代わりに 予測のみ モードを使用します。予測のみモードでは、予測スケーリングは予測を生成し続けますが、自動的にタスク数を増やすことはありません。その後、リソース使用率をモニタリングし、必要に応じてタスク数を手動で減らすことができます。

ステップ 1: (オプション) 時系列データを分析する

まず、予測時系列データを分析します。これはオプションのステップですが、予測の詳細を理解したい場合に役立ちます。

1. 予測を取得する

   予測が作成されたら、予測の特定の期間をクエリできます。このクエリの目的は、特定の期間の時系列データの完全なビューを取得することです。

   クエリには、将来の予測データを最大 2 日間含めることができます。予測スケーリングをしばらく使用している場合は、過去の予測データにアクセスすることもできます。ただし、開始時刻と終了時刻の間の最大期間は 30 日間です。

   get-predictive-scaling-forecast AWS CLI コマンドを使用して予測を取得するには、コマンドに次のパラメータを指定します。

   - `resource-id` パラメータにクラスター名を入力します。
   - ポリシーの名前を `--policy-name` パラメータに入力します。
   - 開始時刻を `--start-time` パラメータに入力して、指定した時刻以降の予測データのみが返されるようにします。

- 終了時刻を --end-time パラメータに入力して、指定された時刻より前の予測データのみが返されるようにします。

```
aws application-autoscaling get-predictive-scaling-forecast \
    --service-namespace ecs \
    --resource-id service/MyCluster/test \
    --policy-name cpu40-predictive-scaling-policy \
    --scalable-dimension ecs:service:DesiredCount \
    --start-time "2021-05-19T17:00:00Z" \
    --end-time "2021-05-19T23:00:00Z"
```

成功すると、コマンドは次の例のようなデータを返します。

```
{
    "LoadForecast": [
        {
            "Timestamps": [
                "2021-05-19T17:00:00+00:00",
                "2021-05-19T18:00:00+00:00",
                "2021-05-19T19:00:00+00:00",
                "2021-05-19T20:00:00+00:00",
                "2021-05-19T21:00:00+00:00",
                "2021-05-19T22:00:00+00:00",
                "2021-05-19T23:00:00+00:00"
            ],
            "Values": [
                153.0655799339254,
                128.8288551285919,
                107.1179447150675,
                197.3601844551528,
                626.4039934516954,
                596.9441277518481,
                677.9675713779869
            ],
            "MetricSpecification": {
                "TargetValue": 40.0,
                "PredefinedMetricPairSpecification": {
                    "PredefinedMetricType": "ASGCPUUtilization"
                }
            }
        }
```

```
    ],
    "CapacityForecast": {
        "Timestamps": [
            "2021-05-19T17:00:00+00:00",
            "2021-05-19T18:00:00+00:00",
            "2021-05-19T19:00:00+00:00",
            "2021-05-19T20:00:00+00:00",
            "2021-05-19T21:00:00+00:00",
            "2021-05-19T22:00:00+00:00",
            "2021-05-19T23:00:00+00:00"
        ],
        "Values": [
            2.0,
            2.0,
            2.0,
            2.0,
            4.0,
            4.0,
            4.0
        ]
    },
    "UpdateTime": "2021-05-19T01:52:50.118000+00:00"
}
```

応答には LoadForecast と CapacityForecast の 2 つの予測が含まれています。LoadForecast は、時間ごとの負荷予測を示します。CapacityForecast は、40.0 の TargetValue (平均 CPU 使用率 40%) を維持しながら予測された負荷を処理するために時間単位で必要なキャパシティーの予測値を示します。

2. ターゲット期間を特定する

   1 回限りの需要変動が発生する時間または時間範囲を特定します。予測に表示される日付と時刻は UTC であることに注意してください。

ステップ 2: 2 つのスケジュールされたアクションを作成する

次に、アプリケーションの負荷が予測を上回る特定の期間に、2 つのスケジュールされたアクションを作成します。例えば、マーケティングイベントで一時的にトラフィックがサイトに流入する場合は、1 回限りのアクションをスケジュールして、開始時に最小キャパシティーを更新できます。次に、イベント終了時に最小キャパシティーを元の設定に戻す別のアクションをスケジュールします。

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。

2. [クラスター] ページで、クラスターを選択します。
3. [クラスターの詳細] ページの [サービス] セクションで、サービスを選択します。

   [サービス詳細] ページが表示されます。
4. [サービスの自動スケーリング] を選択します。

   [ポリシー] ページが表示されます。
5. [スケジュールされたアクション] を選択し、[作成] を選択します。

   [スケジュールアクションの作成] ページが表示されます。
6. バケット名に、一意の名前を入力します。
7. [Time zone (タイムゾーン)] でタイムゾーンを選択。

   リストされているすべてのタイムゾーンは、IANA タイムゾーンデータベースから取得されます。詳細については、「List of tz database time zones」を参照してください。
8. [開始時刻] で、アクションが開始される [日付] と [時刻] を入力します。
9. [繰り返し] で、[1 回] を選択してください。
10. [タスク調整] の [最小] には、タスクの最大数以下の値を入力します。
11. [スケジュールされたアクションの作成] を選択してください。

    [ポリシー] ページが表示されます。
12. イベントの終了時に、最小タスク数を元の設定に戻すように、2 番目にスケジュールされたアクションを設定します。予測スケーリングでは、[最小] に設定した値が予測値未満の場合のみ、タスク数をスケーリングできます。

1 回限りのイベントに対して 2 つのスケジュールされたアクションを作成するには (AWS CLI)

AWS CLI を使用してスケジュールされたアクションを作成するには、put-scheduled-update-group-action コマンドを使用します。

例えば、5 月 19 日の午後 5 時から 8 時間、最小キャパシティーを 3 インスタンスに維持するスケジュールを定義しましょう。以下のコマンドは、このシナリオを実装する方法を示しています。

最初の put-scheduled-update-group-action コマンドは、2021 年 5 月 19 日の午後 5 時 (UTC) に指定された Auto Scaling グループの最小キャパシティーを更新するように Amazon EC2 Auto Scaling に指示します。

```
aws autoscaling put-scheduled-update-group-action --scheduled-action-name my-event-
start \
  --auto-scaling-group-name my-asg --start-time "2021-05-19T17:00:00Z" --minimum-
capacity 3
```

2 番目のコマンドは、2021 年 5 月 20 日の午前 1 時 (UTC) にグループの最小キャパシティーを 1 に設定するように Amazon EC2 Auto Scaling に指示します。

```
aws autoscaling put-scheduled-update-group-action --scheduled-action-name my-event-end
 \
  --auto-scaling-group-name my-asg --start-time "2021-05-20T01:00:00Z" --minimum-
capacity 1
```

これらのスケジュールされたアクションを Auto Scaling グループに追加すると、Amazon EC2 Auto Scaling は次の処理を実行します。

- 2021 年 5 月 19 日の午後 5 時 (UTC) に、最初にスケジュールされたアクションが実行されます。グループのインスタンスが 3 未満である場合、グループは 3 インスタンスにスケールアウトされます。この時刻以降の 8 時間の間、予測キャパシティーが実際のキャパシティーよりも大きい場合、または動的スケーリングポリシーが有効な場合、Amazon EC2 Auto Scaling は引き続きスケールアウトできます。
- 2021 年 5 月 20 日の午前 1 時 (UTC) に、2 番目のスケジュールされたアクションが実行されます。これにより、イベントの終了時に最小キャパシティーが元の設定に戻ります。

繰り返し起こるスケジュールに基づくスケーリング

毎週同じ期間の予測を上書きするには、2 つのスケジュールされたアクションを作成し、cron 式を使用して日時のロジックを指定します。

この cron 式のフォーマットは、スペースで区切られた 5 つのフィールド ([分] [時間] [日] [月] [曜日]) で構成されます。フィールドには、特殊文字を含む任意の許容される値を含めることができます。

例えば、次の cron 式は、毎週火曜日の午前 6:30 にアクションを実行します。アスタリスクは、フィールドのすべての値を照合するワイルドカードとして使用されます。

```
30 6 * * 2
```

関連情報

スケジュールされたアクションを管理する方法の詳細については、「スケジュールされたアクションを使用して Amazon ECS サービスをスケールする」を参照してください。

Amazon ECS のカスタムメトリクスを使用した高度な予測スケーリングポリシー

予測スケーリングポリシーで、事前定義されたメトリクスまたはカスタムメトリクスを使用できます。カスタムメトリクスは、CPU、メモリなどの事前定義されたメトリクスが、アプリケーションの負荷を十分に記述するには不十分である場合に有用です。

カスタムメトリクスを使用して予測スケーリングポリシーを作成するときは、AWS が提供するその他の CloudWatch メトリクスを指定できます。または、自分で定義して公開するメトリクスを指定することもできます。メトリクス計算を使用して既存のメトリクスを集計し、AWS が自動的に追跡しない新しい時系列に変換することもできます。例としては、新しい合計や平均を計算してデータ内の値を結合すること (集計と呼ばれます) が挙げられます。結果のデータは集計と言います。

以下のセクションには、ポリシー用の JSON 構造を構築する方法のベストプラクティスと例が記載されています。

前提条件

予測スケーリングポリシーにカスタムメトリクスを追加するには、`cloudwatch:GetMetricData` 許可が必要です。

AWS が提供するメトリクスの代わりに独自のメトリクスを指定するには、まずそのメトリクスを CloudWatch に発行する必要があります。詳細については、「Amazon CloudWatch ユーザーガイド」の「カスタムメトリクスの発行」を参照してください。

独自のメトリクスを発行するときは、少なくとも 5 分間隔の頻度でデータポイントを発行するようにしてください。データポイントは、必要な期間の長さに基づいて CloudWatch から取得されます。例えば、負荷メトリクスの指定では、時間単位のメトリクスを使用してアプリケーションの負荷を測定します。CloudWatch は、発行されたメトリクスデータを使用して、各 1 時間の期間内にタイムスタンプがあるすべてのデータポイントを集計することにより、各 1 時間の期間に対して単一のデータ値を提供します。

ベストプラクティス

次のベストプラクティスは、カスタムメトリクスをより効果的に使用するのに役立ちます。

- 負荷メトリクス仕様で最も有用なメトリクスは、Auto Scaling グループ全体の負荷を表すメトリクスです。

- スケーリングメトリクス仕様のスケールに最も有用なメトリクスは、タスクメトリクスごとの平均スループットまたは使用率です。
- ターゲット使用率は、スケーリングメトリクスのタイプと一致する必要があります。例えば、CPU 使用率を使用するポリシー設定の場合、これはパーセンテージのターゲットです。
- これらの推奨事項に従わない場合、予測される将来の時系列の値は、多くの場合、誤りになります。データが正しいことを確認するために、コンソールで予測値を表示できます。または、予測スケーリングポリシーを作成した後、GetPredictiveScalingForecast API を呼び出して返された `LoadForecast` オブジェクトを検査します。
- 予測スケーリングがアクティブスケーリングを開始する前に予測を評価できるように、予測のみモードで予測スケーリングを設定することを強くお勧めします。

制限事項

- 1 つのメトリクス指定で最大 10 個のメトリクスのデータポイントをクエリできます。
- この制限に関しては、1 つの式は 1 つのメトリクスとしてカウントされます。

カスタムメトリクスを使用した予測スケーリングポリシーのトラブルシューティング

カスタムメトリクスの使用中に問題が発生した場合は、次の操作を実行することをお勧めします。

- 検索式の使用中にブルー/グリーンデプロイで問題が発生した場合は、完全一致ではなく部分一致を検索する検索式を作成してください。また、クエリが特定のアプリケーションで実行されている Auto Scaling グループのみを検索していることも確認する必要があります。検索式の構文の詳細については、「Amazon CloudWatch ユーザーガイド」の「CloudWatch 検索式の構文」を参照してください。
- put-scaling-policy コマンドは、スケーリングポリシーの作成時に式を検証します。ただし、このコマンドでは、検出されたエラーの正確な原因を特定できない可能性があります。問題を解決するには、get-metric-data コマンドへのリクエストからの応答で受け取ったエラーをトラブルシューティングします。CloudWatch コンソールから式をトラブルシューティングすることもできます。
- `MetricDataQueries` で SUM() のような数学関数を使用せずに、独自の SEARCH() 関数を指定する場合、`ReturnData` に `false` を指定する必要があります。これは、検索式が複数の時系列を返す可能性がある一方、数式に基づくメトリクス指定は 1 つの時系列しか返すことができないためです。
- 検索式に含まれるすべてのメトリクスは、同じ解像度である必要があります。

Amazon ECS を使用した予測スケーリングカスタムメトリクス用の JSON の構築

以下のセクションには、CloudWatch からのデータをクエリするための予測スケーリングを設定する方法の例が記載されています。このオプションの設定には 2 つの異なる手法あり、予測スケーリングポリシーの JSON を構築するために使用する形式は、選択される手法の影響を受けます。メトリクス計算を使用する場合は、実行されるメトリクス計算に基づいて JSON の形式がさらに多様化します。

1. AWS が提供するその他の CloudWatch メトリクス、またはユーザーが CloudWatch に発行するメトリクスから直接データを取得するポリシーを作成するには、「AWS CLI を使用したカスタム負荷メトリクスとスケーリングメトリクスを用いた予測スケーリングポリシーの例」を参照してください。

AWS CLI を使用したカスタム負荷メトリクスとスケーリングメトリクスを用いた予測スケーリングポリシーの例

AWS CLI でカスタムロードメトリクスとスケーリングメトリクスを使用する予測スケーリングポリシーを作成するには、`config.json` という名前の JSON ファイルに `--predictive-scaling-configuration` の引数を保存します。

カスタムメトリクスの追加は、以下の例にある置き換え可能な値を独自のメトリクスとターゲット使用率に置き換えることによって開始します。

```
{
  "MetricSpecifications": [
    {
      "TargetValue": 50,
      "CustomizedScalingMetricSpecification": {
        "MetricDataQueries": [
          {
            "Id": "scaling_metric",
            "MetricStat": {
              "Metric": {
                "MetricName": "MyUtilizationMetric",
                "Namespace": "MyNameSpace",
                "Dimensions": [
                  {
                    "Name": "MyOptionalMetricDimensionName",
                    "Value": "MyOptionalMetricDimensionValue"
                  }
                ]
```

```
              },
              "Stat": "Average"
            }
          }
        ]
      },
      "CustomizedLoadMetricSpecification": {
        "MetricDataQueries": [
          {
            "Id": "load_metric",
            "MetricStat": {
              "Metric": {
                "MetricName": "MyLoadMetric",
                "Namespace": "MyNameSpace",
                "Dimensions": [
                  {
                    "Name": "MyOptionalMetricDimensionName",
                    "Value": "MyOptionalMetricDimensionValue"
                  }
                ]
              },
              "Stat": "Sum"
            }
          }
        ]
      }
    }
  ]
}
```

詳細については、「Amazon EC2 Auto Scaling API Reference」の「MetricDataQuery」を参照してください。

> **Note**
>
> 以下は、CloudWatch メトリクスのメトリクス名、名前空間、ディメンション、および統計を見つけるために役立つ追加のリソースです。
>
> - AWS のサービスで使用可能なメトリクスの詳細については、「Amazon CloudWatch ユーザーガイド」の「CloudWatch メトリクスを発行する AWS のサービス」を参照してください。

- CloudWatch メトリクスの正確なメトリクス名、名前空間、ディメンション (該当する場合) を AWS CLI で取得するには、「list-metrics」を参照してください。

このポリシーを作成するには、以下の例にあるように、JSON ファイルを入力として使用して put-scaling-policy コマンドを実行します。

```
aws application-autoscaling put-scaling-policy --policy-name my-predictive-scaling-policy \
  --auto-scaling-group-name my-asg --policy-type PredictiveScaling \
  --predictive-scaling-configuration file://config.json
```

成功した場合、このコマンドはポリシーの Amazon リソースネーム (ARN) を返します。

```
{
  "PolicyARN": "arn:aws:autoscaling:region:account-id:scalingPolicy:2f4f5048-d8a8-4d14-b13a-d1905620f345:autoScalingGroupName/my-asg:policyName/my-predictive-scaling-policy",
  "Alarms": []
}
```

Metric Math 式を使用する

以下のセクションでは、ポリシー内の予測スケーリングポリシーで Metric Math を使用する方法について説明します。

Metric Math について

既存のメトリクスデータの集計だけを行いたい場合は、CloudWatch Metric Math により、別のメトリクスを CloudWatch に発行する手間とコストを節約できます。AWS が提供するメトリクスはすべて使用できます。また、アプリケーションの一部として定義したメトリクスを使用することもできます。

詳細については、「Amazon CloudWatch ユーザーガイド」の「Amazon CloudWatch メトリクス数学の使用」を参照してください。

予測スケーリングポリシーで Metric Math の数式を使用する場合は、次の点を考慮してください。

- Metric Math 演算では、メトリクスのメトリクス名、名前空間、ディメンションのキーと値のペアの一意の組み合わせのデータポイントを使用します。
- 任意の算術演算子 (+ - * / ^)、統計関数 (AVG や SUM など)、または CloudWatch がサポートするその他の関数を使用できます。

- 数式の関係式では、メトリクスと他の数式の結果の両方を使用できます。
- Metric Math の数式は、さまざまな集計で構成できます。ただし、最終的な集計結果として、Average をスケーリングメトリクスに使用し、Sum を負荷メトリクスに使用するのがベストプラクティスです。
- メトリクスの指定で使用される数式はすべて、最終的に単一の時系列を返す必要があります。

Metric Math を使用するには、次の操作を実行します。

- 1 つまたは複数の CloudWatch メトリクスを選択します。次に、数式を作成します。詳細については、「Amazon CloudWatch ユーザーガイド」の「Amazon CloudWatch メトリクス数学の使用」を参照してください。
- CloudWatch コンソールまたは CloudWatch GetMetricData API を使用して、Metric Math の数式が有効であることを確認します。

メトリクス計算を使用してメトリクスを組み合わせる予測スケーリングポリシーの例 (AWS CLI)

場合によっては、メトリクスを直接指定するのではなく、まず何らかの方法でそのデータを処理する必要がある場合があります。例えば、Amazon SQS キューから作業を取り出すアプリケーションがあり、キュー内の項目数を予測スケーリングの基準として使用したいとします。キューにあるメッセージの数だけでは、必要なインスタンスの数は定義されません。インスタンスごとのバックログを計算するために使用できるメトリクスを作成するには、さらに多くの作業が必要です。

このシナリオの予測スケーリングポリシー例を次に示します。Amazon SQS ApproximateNumberOfMessagesVisible メトリクス (キューから取得可能なメッセージの数) に基づくスケーリングメトリクスおよび負荷メトリクスを指定します。Amazon EC2 Auto Scaling GroupInServiceInstances メトリクスと、スケーリングメトリクスのインスタンスごとのバックログを計算するための数式も使用します。

```
aws application-autoscaling put-scaling-policy --policy-name my-sqs-custom-metrics-
policy \
  --policy-type PredictiveScaling \
  --predictive-scaling-configuration file://config.json
  --service-namespace ecs \
  --resource-id service/MyCluster/test \
  "MetricSpecifications": [
    {
      "TargetValue": 100,
      "CustomizedScalingMetricSpecification": {
```

```
        "MetricDataQueries": [
          {
            "Label": "Get the queue size (the number of messages waiting to be
processed)",
            "Id": "queue_size",
            "MetricStat": {
              "Metric": {
                "MetricName": "ApproximateNumberOfMessagesVisible",
                "Namespace": "AWS/SQS",
                "Dimensions": [
                  {
                    "Name": "QueueName",
                    "Value": "my-queue"
                  }
                ]
              },
              "Stat": "Sum"
            },
            "ReturnData": false
          },
          {
            "Label": "Get the group size (the number of running instances)",
            "Id": "running_capacity",
            "MetricStat": {
              "Metric": {
                "MetricName": "GroupInServiceInstances",
                "Namespace": "AWS/AutoScaling",
                "Dimensions": [
                  {
                    "Name": "AutoScalingGroupName",
                    "Value": "my-asg"
                  }
                ]
              },
              "Stat": "Sum"
            },
            "ReturnData": false
          },
          {
            "Label": "Calculate the backlog per instance",
            "Id": "scaling_metric",
            "Expression": "queue_size / running_capacity",
            "ReturnData": true
          }
```

```
        ]
      },
      "CustomizedLoadMetricSpecification": {
        "MetricDataQueries": [
          {
            "Id": "load_metric",
            "MetricStat": {
              "Metric": {
                "MetricName": "ApproximateNumberOfMessagesVisible",
                "Namespace": "AWS/SQS",
                "Dimensions": [
                  {
                    "Name": "QueueName",
                    "Value": "my-queue"
                  }
                ],
              },
              "Stat": "Sum"
            },
            "ReturnData": true
          }
        ]
      }
    }
  ]
}
```

この例では、ポリシーの ARN が返されます。

```
{
  "PolicyARN": "arn:aws:autoscaling:region:account-id:scalingPolicy:2f4f5048-d8a8-4d14-
b13a-d1905620f345:autoScalingGroupName/my-asg:policyName/my-sqs-custom-metrics-policy",
  "Alarms": []
}
```

Amazon ECS サービスを相互接続する

Amazon ECS タスク内で実行されるアプリケーションは、多くの場合はインターネットからの接続を受信するか、Amazon ECS サービス内で実行される他のアプリケーションに接続する必要があります。インターネットからの外部接続が必要な場合は、Elastic Load Balancing の使用をお勧めします。統合負荷分散の詳細については、「the section called “ロードバランサーを使用してサービストラフィックを分散する”」を参照してください。

Amazon ECS サービス内で実行される他のアプリケーションにアプリケーションを接続する必要がある場合、Amazon ECS にはロードバランサーを使用せずに、次の方法で行うことができます。

- Amazon ECS Service Connect

  サービス検出、接続、トラフィックモニタリング用の Amazon ECS 設定を提供する Service Connect をお勧めします。Service Connect を使用すると、アプリケーションは短縮名と標準ポートを使用して、同じクラスターや他のクラスター (同じ AWS リージョン の VPC を含む) 内の Amazon ECS サービスに接続できます。

  Service Connect を使用すると、Amazon ECS がサービス検出のすべての部分を管理します。つまり、検出可能な名前の作成、タスクの開始と終了時に各タスクのエントリを動的に管理すること、名前を検出するように設定された各タスクでのエージェントの実行などです。アプリケーションは DNS 名の標準機能を使用して接続することで名前を検索できます。アプリケーションがすでにこれを実行している場合、Service Connect を使用する際に、アプリケーションを変更する必要はありません。

  各サービスとタスク定義内で完全な設定を行います。Amazon ECS は、デプロイ内のすべてのタスクが同じように動作するように、サービスデプロイごとにこの設定の変更を管理します。たとえば、サービス検出の DNS でよくある問題は、移行の制御です。新しい IP アドレスを指すように DNS 名を変更すると、すべてのクライアントが新しいサービスの使用を開始するまでに最大 TTL 時間がかかることがあります。Service Connect では、クライアントデプロイメントがクライアントタスクを置き換えて設定を更新します。他のデプロイと同様に、Service Connect の変更に影響するように、デプロイサーキットブレーカーやその他のデプロイ設定を設定できます。

  詳細については、「Service Connect を使用して Amazon ECS サービスを短縮名で接続する」を参照してください。

- Amazon ECS サービス検出

  サービス間の通信のもう 1 つのアプローチは、サービス検出を使用する直接的な通信です。このアプローチでは、Amazon ECS との AWS Cloud Map サービス検出の統合を使用できます。Amazon ECS はサービス検出を使用して、起動されたタスクのリストを AWS Cloud Map に同期します。このホスト名は特定のサービスの 1 つ以上のタスクの内部 IP アドレスに解決される DNS ホスト名を保持します。Amazon VPC の他のサービスは、この DNS ホスト名を使用して、内部 IP アドレスを使用してトラフィックを別のコンテナに直接送信できます。

  このサービス間通信のアプローチでは、レイテンシが低くなります。コンテナ間には余分なコンポーネントはありません。トラフィックは 1 つのコンテナから別のコンテナに直接移動します。

この方法は、各タスクに固有の IP アドレスが割り当てられる awsvpc ネットワークモードを使用する場合に適しています。ほとんどのソフトウェアは、IP アドレスに直接変換される DNS A レコードの使用のみをサポートしています。awsvpc ネットワークモードを使用する場合、各タスクの IP アドレスは A レコードになります。ただし、bridge ネットワークモードを使用している場合は、複数のコンテナが同じ IP アドレスを共有している可能性があります。さらに、動的ポートマッピングでは、その 1 つの IP アドレスのポート番号がコンテナにランダムに割り当てられます。この時点では、A レコードだけではサービス検出には不十分です。SRV レコードも使用する必要があります。このタイプのレコードは IP アドレスとポート番号の両方を記録できますが、アプリケーションを適切に設定する必要があります。使用するビルド済みアプリケーションの中には、SRV レコードをサポートしていないものもあります。

awsvpc ネットワークモードのもう 1 つの利点は、サービスごとに固有のセキュリティグループがあることです。このセキュリティ グループを設定して、そのサービスと通信する必要がある特定のアップストリームサービスからの受信接続のみを許可することができます。

サービス検出を使用するサービス間直接的な通信の主な欠点は、再試行や接続障害に対処するためのロジックを追加する必要があることです。DNS レコードには、キャッシュされる時間を制御する有効期限 (TTL) があります。DNS レコードが更新されてキャッシュが期限切れになり、アプリケーションが最新バージョンの DNS レコードを取得できるようになるまでには、ある程度の時間がかかります。そのため、アプリケーションが DNS レコードを解決して、もう存在しない別のコンテナを指すようになってしまう可能性があります。アプリケーションには再試行を処理し、不正なバックエンドを無視するロジックが必要です。

詳細については、サービス検出を使用して Amazon ECS サービスを DNS 名で接続するを参照してください。

- Amazon VPC Lattice

  Amazon VPC Lattice は、Amazon ECS のお客様がコードを変更することなく、AWS コンピューティングサービス、VPC、アカウント全体で構築されたアプリケーションを観察、保護、モニタリングするために使用するマネージドアプリケーションネットワーキングサービスです。

  VPC Lattice は、コンピューティングリソースのコレクションであるターゲットグループを使用します。これらのターゲットは、アプリケーションまたはサービスを実行するものであり、Amazon EC2 インスタンス、IP アドレス、Lambda 関数、Application Load Balancer を含みます。Amazon ECS サービスを VPC Lattice ターゲットグループに関連付けることで、お客様は Amazon ECS タスクを VPC Lattice の IP ターゲットとして有効にできるようになりました。Amazon ECS は、登

録されたサービスのタスクが起動されると、VPC Lattice ターゲットグループにタスクを自動的に登録します。

詳細については、「Amazon VPC Lattice を使用して Amazon ECS サービスを接続、監視、保護する」を参照してください。

ネットワークモード互換表

次の表は、これらのオプションとタスクネットワークモードの互換性を示すものです。この表の「クライアント」とは、Amazon ECS タスク内から接続を行うアプリケーションを指します。

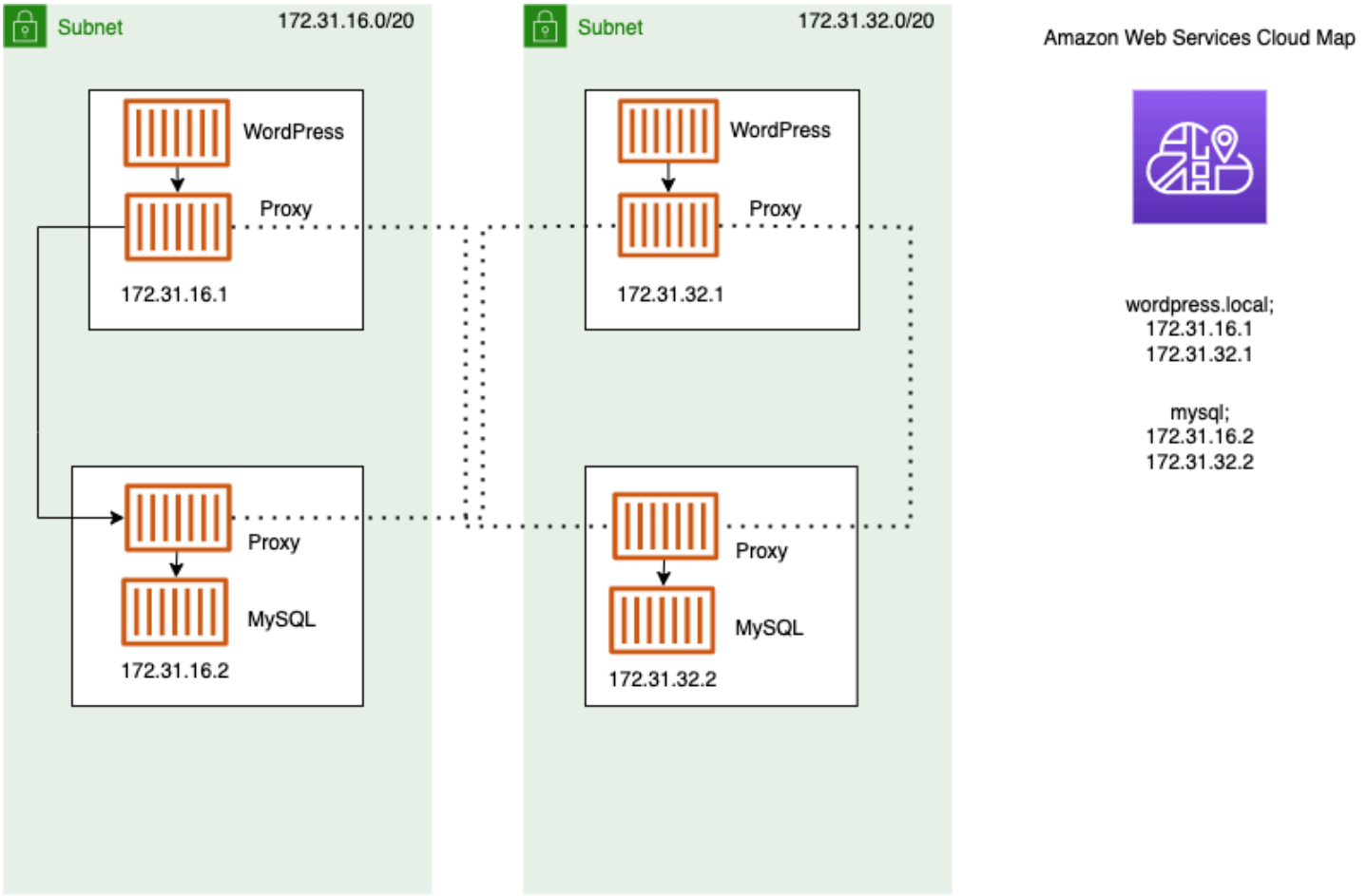
相互接続オプション	ブリッジ	**awsvpc**	ホスト
サービス検出	はい。ただし、クライアントは `hostPort` を使用せず DNS の SRV レコードを認識する必要があります。	はい	はい。ただし、クライアントは `hostPort` を使用せず DNS の SRV レコードを認識する必要があります。
Service Connect	はい	はい	なし
VPC Lattice	はい	はい	はい

Service Connect を使用して Amazon ECS サービスを短縮名で接続する

Amazon ECS Service Connect では、Amazon ECS 設定としてサービス間通信を管理できます。Amazon ECS でサービス検出とサービスメッシュの両方を構築します。これにより、サービスのデプロイごとに管理する各サービス内の完全な設定、VPC DNS 設定に依存しない名前空間内のサービスを参照する統一された方法、すべてのアプリケーションを監視するための標準化されたメトリクスとログが提供されます。Service Connect は、サービスのみを相互接続します。

次の図は、VPC に 2 つのサブネットと 2 つのサービスを含む Service Connect ネットワークの例を示しています。各サブネットにつき 1 つのタスクで WordPress を実行するクライアントサービス。各サブネットに 1 つのタスクで MySQL を実行するサーバーサービス。どちらのサービスも、2 つのサブネットに分散された複数のタスクを実行するため、可用性が高く、タスクやアベイラビリティー

ゾーンの問題に対する耐性があります。実線の矢印は、WordPress から MySQL への接続を示しています。例えば、IP アドレス 172.31.16.1 が設定されたタスク内の WordPress コンテナ内から実行される mysql --host=mysql CLI コマンドなどです。このコマンドは、MySQL のデフォルトポートの短縮名 mysql を使用します。この名前とポートは、同じタスク内で Service Connect プロキシに接続します。WordPress タスクのプロキシは、ラウンドロビン負荷分散と異常値検出に対する以前の障害情報を使用して、接続する MySQL タスクを選択します。図の実線の矢印で示されているように、プロキシは IP アドレス 172.31.16.2 を使用して MySQL タスクの 2 番目のプロキシに接続します。2 番目のプロキシは、同じタスクでローカル MySQL サーバーに接続します。どちらのプロキシも接続パフォーマンスをレポートし、Amazon ECS と Amazon CloudWatch コンソールのグラフに表示されるため、あらゆる種類のアプリケーションのパフォーマンスメトリクスを同じ方法で取得できます。



Service Connect では次の用語が使用されます。

ポート名

特定のポートマッピングに名前を割り当てる Amazon ECS タスク定義設定です。この設定は Amazon ECS Service Connect でのみ使用されます。

クライアントエイリアス

エンドポイントで使用されるポート番号を割り当てる Amazon ECS サービス設定です。さらに、クライアントエイリアスはエンドポイントの DNS 名を割り当て、検出名を上書きすることができます。Amazon ECS サービスで検出名が提供されていない場合、クライアントエイリアス名がエンドポイント名としてポート名に上書きされます。エンドポイントの例については、エンドポイントの定義を参照してください。1 つの Amazon ECS サービスに複数のクライアントエイリアスを割り当てることができます。この設定は Amazon ECS Service Connect でのみ使用されます。

検出名

タスク定義から指定されたポートにオプションとして作成できる中間名。この名前は AWS Cloud Map サービスの作成に使用されます。この名前が指定されていない場合は、タスク定義からのポート名が使用されます。特定のポートおよび Amazon ECS サービスに複数の検出名を割り当てることができます。この設定は Amazon ECS Service Connect でのみ使用されます。

AWS Cloud Map サービス名は、名前空間内で一意である必要があります。この制限により、各名前空間の特定のタスク定義に対し、検出名のない Service Connect 設定は 1 つしか使用できません。

エンドポイント

API またはウェブサイトに接続するための URL です。URL にはプロトコル、DNS 名、ポートが含まれます。エンドポイント全般の詳細については、「Amazon Web Services 全般のリファレンス」の「AWS 用語集」の「エンドポイント」を参照してください。

Service Connect は、Amazon ECS サービスに接続するエンドポイントを作成すると共に、Amazon ECS サービスのタスクがエンドポイントに接続するための設定を行います。URL にはプロトコル、DNS 名、ポートが含まれます。ポートはコンテナイメージ内のアプリケーションと一致する必要があるため、プロトコルとポート名はタスク定義で選択します。サービス内では、各ポートを名前で選択し、DNS 名を割り当てることができます。Amazon ECS サービス設定で DNS 名を指定しない場合、デフォルトではタスク定義からのポート名が使用されます。例えば、Service Connect エンドポイントは `http://blog:80`、`grpc://checkout:8080`、`http://_db.production.internal:99` のいずれかになります。

Service Connect サービス

Amazon ECS サービス内の 1 つのエンドポイントの設定です。これは Service Connect 設定の一部で、コンソールの [Service Connect and discovery name configuration] (Service Connect および検出名の設定) の中の 1 行、または Amazon ECS サービスにおける JSON 設定の `services`

リストの中の 1 つのオブジェクトで構成されます。この設定は Amazon ECS Service Connect でのみ使用されます。

詳細については、「Amazon Elastic Container Service API リファレンス」の「ServiceConnectService」を参照してください。

名前空間

Service Connect で使用する AWS Cloud Map 名前空間の短縮名または完全な Amazon リソースネーム (ARN)。名前空間は、Amazon ECS サービスおよびクラスターと同じ AWS リージョンにある必要があります。AWS Cloud Map 内の名前空間のタイプは Service Connect に影響しません。名前空間は、AWS RAM で使用できる AWS リージョンの AWS Resource Access Manager (AWS RAM) を使用して AWS アカウントと共有できます。共有名前空間の詳細については、「AWS Cloud Map デベロッパーガイド」の「クロスアカウント AWS Cloud Map 名前空間共有」を参照してください。

Service Connect は、相互に対話する Amazon ECS タスクの論理グループとして AWS Cloud Map 名前空間を使用します。各 Amazon ECS サービスは 1 つの名前空間のみに属することができます。名前空間内のサービスは、同じ AWS リージョン内の異なる Amazon ECS クラスターに分散できます。名前空間が共有名前空間である場合、サービスは名前空間所有者と名前空間コンシューマー AWS アカウントに分散できます。サービスは、任意の基準で自由に整理できます。

クライアントサービス

ネットワーククライアントアプリケーションを実行するサービスです。このサービスには名前空間が設定されている必要があります。サービス内の各タスクは、Service Connect プロキシコンテナを介して、名前空間のすべてのエンドポイントを検出して接続できます。

タスク内のいずれかのコンテナが名前空間内のサービスからエンドポイントに接続する必要がある場合は、クライアントサービスを選択します。フロントエンド、リバースプロキシ、またはロードバランサーアプリケーションが、例えば Elastic Load Balancing からなど、他の方法で外部トラフィックを受信する場合は、このタイプの Service Connect 設定を使用できます。

クライアント/サーバーサービス

ネットワークまたはウェブサービスアプリケーションを実行する Amazon ECS サービスです。このサービスには、名前空間と少なくとも 1 つのエンドポイントが設定されている必要があります。サービス内の各タスクには、エンドポイントを使用してアクセスできます。Service Connect プロキシコンテナは、エンドポイント名とポートをリッスンして、タスク内のアプリケーションコンテナにトラフィックを誘導します。

いずれかのコンテナが公開してポート上でネットワークトラフィックをリッスンする場合は、クライアント/サーバーサービスを選択してください。これらのアプリケーションは、同じ名前空間内の他のクライアント/サーバーサービスに接続する必要はありませんが、クライアント設定が必要です。バックエンド、ミドルウェア、ビジネス層、またはほとんどのマイクロサービスはこのタイプの Service Connect 設定を使用できます。フロントエンド、リバースプロキシ、またはロードバランサーアプリケーションに、同じ名前空間の Service Connect を使用して設定した他のサービスからのトラフィックを受信させる場合、これらのサービスはこのタイプの Service Connect 設定を使用する必要があります。

Service Connect 機能は関連サービスの仮想ネットワークを作成します。同じサービス設定を複数の異なる名前空間で使用することで、独立していても同一のアプリケーションセットを実行できます。Service Connect は Amazon ECS サービスのプロキシコンテナを定義します。これにより、同じタスク定義を使用して、Service Connect 設定が異なるさまざまな名前空間で同一のアプリケーションを実行できます。サービスが作成する各タスクは、タスク内のプロキシコンテナを実行します。

Service Connect は同じ名前空間内の Amazon ECS サービス間の接続に適しています。次のアプリケーションで Service Connect で設定された Amazon ECS サービスに接続するためには、追加の相互接続方法を使用する必要があります。

- 他の名前空間で設定されているタスク
- Service Connect 用に設定されていないタスク
- Amazon ECS の外部にあるその他のアプリケーション

これらのアプリケーションは Service Connect プロキシ経由で接続できますが、Service Connect エンドポイント名を解決することはできません。

これらのアプリケーションで Amazon ECS タスクの IP アドレスを解決するには、別の相互接続方法を使用する必要があります。

トピック

- 料金
- Amazon ECS Service Connect コンポーネント
- Amazon ECS Service Connect 設定の概要
- 共有 AWS Cloud Map 名前空間を使用した Amazon ECS Service Connect
- Amazon ECS Service Connect アクセスログ

- Amazon ECS Service Connect ゾーン認識ルーティング
- Amazon ECS Service Connect トラフィックを暗号化する
- AWS CLI を使用した Amazon ECS Service Connect の設定

料金

- Amazon ECS Service Connect の料金は、コンテナ化されたワークロードをホストするために AWS Fargate または Amazon EC2 インフラストラクチャを使用するかどうかによって異なります。AWS Outposts で Amazon ECS を使用する場合、料金は Amazon EC2 を直接使用する場合に使われる同じモデルに従います。詳細については、Amazon ECS 料金表を参照してください。
- Amazon ECS Service Connect の使用には追加料金はかかりません。
- Service Connect で使用する場合に AWS Cloud Map の使用に追加料金は発生しません。
- お客様は、vCPU やメモリなど、Amazon ECS Service Connect で使用されるコンピューティングリソースに対して料金を支払います。Amazon ECS Service Connect エージェントはお客様のタスク内で実行されるため、実行に追加料金はかかりません。タスクリソースは、お客様のワークロードと Amazon ECS Service Connect エージェント間で共有されます。
- AWS Private CA で Amazon ECS Service Connect トラフィック暗号化機能を使用する場合、お客様は作成するプライベート認証機関と発行される TLS 証明書ごとに料金を支払います。詳細については、「AWS Private Certificate Authority の料金」を参照してください。TLS 証明書の月額コストを見積もるには、TLS が有効になっている Amazon ECS サービスの数を把握し、それに証明書のコストを掛けて、さらに 6 を掛ける必要があります。Amazon ECS Service Connect は TLS 証明書を 5 日ごとに自動的にローテーションするため、平均して 1 か月あたり Amazon ECS サービスごとに 6 つの証明書が発行されます。

Amazon ECS Service Connect コンポーネント

Amazon ECS Service Connect を使用する場合、ネットワークリクエストを受け取るサーバーアプリケーション (クライアント/サーバーサービス) を実行するか、リクエストを行うクライアントアプリケーション (クライアントサービス) を実行するように各 Amazon ECS サービスを設定します。

Service Connect の使用を開始する準備が整ったら、クライアント/サーバーサービスから始めます。新しいサービスまたは既存のサービスに Service Connect 設定を追加できます。Amazon ECS は、名前空間に Service Connect エンドポイントを作成します。さらに、Amazon ECS は現在実行中のタスクを置き換える新しいデプロイをサービス内に作成します。

既存のタスクやその他のアプリケーションは、既存のエンドポイントや外部アプリケーションに引き続き接続できます。クライアント/サーバーサービスがスケールアウトによってタスクを追加すると、クライアントからの新しい接続はすべてのタスク間で分散されます。クライアント/サーバーサービスが更新されると、クライアントからの新しい接続は新バージョンのタスク間で分散されます。

既存のタスクを解決して新しいエンドポイントに接続することはできません。このエンドポイントを解決して接続できるのは、同じ名前空間に Service Connect 設定があり、このデプロイ後に実行を開始した新しいタスクだけです。

つまり、クライアントアプリケーションのオペレーターがアプリの設定が変更されるタイミングを決定しますが、サーバーアプリケーションのオペレーターはいつでも設定を変更できます。名前空間のエンドポイントのリストは、名前空間のサービスがデプロイされるたびに変更される可能性があります。既存のタスクと置換タスクは、最新のデプロイ後も同じように動作し続けます。

次に挙げるサンプルを参考にしてください。

まず、パブリックインターネット上で利用可能なアプリケーションを 1 つの AWS CloudFormation テンプレートと 1 つの CloudFormation スタックで作成していると仮定します。CloudFormation がパブリック検出と到達可能性を作成するのは、フロントエンドクライアントサービスを含めて、最後にする必要があります。フロントエンドクライアントサービスが実行されていて一般に公開されているのに、バックエンドは公開されていないという期間の発生を避けるために、サービスの作成はこの順序で行う必要があります。これにより、その期間中にエラーメッセージが一般に送信されるのを防ぐことができます。AWS CloudFormation では `dependsOn` を使用して、複数の Amazon ECS サービスを並列または同時に作成できないことを CloudFormation に示す必要があります。クライアントタスクが接続するバックエンドクライアント/サーバーサービスごとに、フロントエンドクライアントサービスに `dependsOn` を追加する必要があります。

次に、フロントエンドサービスが Service Connect 設定なしで存在すると仮定します。タスクは既存のバックエンドサービスに接続しています。まず、フロントエンドが使用する DNS または `clientAlias` と同じ名前を使用して、バックエンドサービスにクライアント/サーバー Service Connect 設定を追加します。これにより、新しいデプロイが作成されると共に、すべてのデプロイのロールバック検出や、AWS マネジメントコンソール、AWS CLI、AWS SDK などの、バックエンドサービスをロールバックして以前のデプロイと設定に戻す方法が作成されます。バックエンドサービスのパフォーマンスと動作に問題がなければ、クライアントまたはクライアント/サーバー Service Connect 設定をフロントエンドサービスに追加します。それらの新しいタスクに追加された Service Connect プロキシを使用するのは、新しいデプロイのタスクのみです。この設定に問題がある場合は、デプロイロールバック検出または AWS マネジメントコンソール、AWS CLI、AWS SDK などのバックエンドサービスをロールバックして以前のデプロイと設定に戻す方法を使用することで、ロー

ルバックして以前の設定に戻すことができます。Service Connect ではなく DNS に基づく別のサービス検出システムを使用する場合、フロントエンドまたはクライアントアプリケーションは、ローカル DNS キャッシュの有効期限が切れてから新しいエンドポイントと変更されたエンドポイント設定の使用を開始しますが、これには通常数時間かかります。

ネットワーク

デフォルトでは、Service Connect プロキシは、タスク定義ポートマッピングから `containerPort` をリッスンします。セキュリティグループルールでは、クライアントが実行されるサブネットからこのポートへの受信 (イングレス) トラフィックを許可する必要があります。

Service Connect サービス設定でポート番号を設定した場合であっても、Service Connect プロキシがリッスンするクライアント/サーバーサービスのポートは変更されません。このポート番号を設定すると、Amazon ECS はそれらのタスク内の Service Connect プロキシ上で、クライアントサービスが接続するエンドポイントのポートを変更します。クライアントサービスのプロキシは、`containerPort` を使用してクライアント/サーバーサービスのプロキシに接続します。

Service Connect プロキシがリッスンするポートを変更する場合は、クライアント/サーバーサービスの Service Connect 設定で `ingressPortOverride` を変更してください。このポート番号を変更する場合は、このサービスへのトラフィックが使用するこのポートのインバウンドトラフィックを許可する必要があります。

Service Connect 用に設定された Amazon ECS サービスにアプリケーションが送信するトラフィックは、Amazon VPC およびサブネットにおいて、ルートテーブルルールおよびネットワーク ACL ルールにより、使用している `containerPort` と `ingressPortOverride` のポート番号が許可されている必要があります。

Service Connect を使用して VPC 間でトラフィックを送信できます。ルートテーブルルール、ネットワーク ACL、セキュリティグループに対する同じ要件が両方の VPC に適用されます。

例えば、2 つのクラスターは異なる VPC でタスクを作成します。各クラスターのサービスは、同じ名前空間を使用するように設定されています。これら 2 つのサービスのアプリケーションは、VPC DNS 設定なしで名前空間のすべてのエンドポイントを解決できます。ただし、VPC ピアリング、VPC またはサブネットのルートテーブル、そして VPC ネットワーク ACL により、`containerPort` と `ingressPortOverride` のポート番号のトラフィックが許可されない限り、プロキシは接続できません。

`bridge` ネットワークモードを使用するタスクでは、動的ポートの上限範囲でのトラフィックを許可するインバウンドルールを含むセキュリティグループを作成する必要があります。次に、Service Connect クラスター内のすべての EC2 インスタンスにセキュリティグループを割り当てます。

Service Connect プロキシ

Service Connect 設定を使用してサービスを作成または更新した場合、新しいタスクが開始されるたびに、Amazon ECS はタスクに新しいコンテナを追加します。別のコンテナを使用するこのパターンは、`sidecar` と呼ばれます。このコンテナはタスク定義には存在せず、設定できません。Amazon ECS は、サービス内のコンテナ設定を管理します。これにより、Service Connect を使用せずに、複数のサービス、名前空間、タスク間で同じタスク定義を再利用できます。

プロキシリソース

- タスク定義では、CPU パラメータとメモリパラメータを設定する必要があります。

  Service Connect プロキシコンテナの処理能力を向上させるために、お客様のタスク CPU とメモリに、256 CPU ユニットと 64 MiB 以上のメモリを追加することをお勧めします。AWS Fargate では、ユーザーが設定できる最小のメモリ容量は 512 MiB です。Amazon EC2 では、タスク定義メモリが必要です。
- サービスでは、Service Connect 設定でログ設定を設定します。
- このサービスのタスクがピーク負荷時に 1 秒あたり 500 を超えるリクエストを受信すると予想される場合は、Service Connect プロキシコンテナのこのタスク定義で、タスク CPU に 512 CPU ユニットを追加することをお勧めします。
- 名前空間内に 100 個以上の Service Connect サービス、または名前空間内のすべての Amazon ECS サービスに合計で 2000 個以上のタスクを作成する場合、Service Connect プロキシコンテナのタスクに対して 128 MiB のメモリを追加することをお勧めします。これは、名前空間内のすべての Amazon ECS サービスで使用されるすべてのタスク定義で行う必要があります。

プロキシ設定

アプリケーションは、アプリケーションと同じタスクでサイドカーコンテナ内のプロキシに接続します。Amazon ECS は、アプリケーションが同じ名前空間のエンドポイント名に接続されている場合にのみアプリケーションがプロキシに接続するように、タスクとコンテナを設定します。その他のすべてのトラフィックはプロキシを使用しません。他のトラフィックには、同じ VPC 内の IP アドレス、AWS サービス エンドポイント、外部トラフィックが含まれます。

負荷分散

Service Connect は、Service Connect エンドポイントのタスク間の負荷分散にラウンドロビン方式を使用するようにプロキシを設定します。接続元のタスクにあるローカルプロキシは、エンドポイントを提供するクライアントサーバーサービス内のタスクの 1 つを選択します。

例えば、「local」という名前空間のクライアントサービスとして設定されたサービスで WordPress を実行するタスクについて考えてみます。MySQL データベースを実行する 2 つのタスクを持つ、他のサービスがあります。このサービスは、同じ名前空間内の Service Connect を通じて `mysql` と呼ばれるエンドポイントを提供するように構成されています。WordPress タスクでは、WordPress アプリケーションはエンドポイント名を使用してデータベースに接続します。この名前への接続は、同じタスクのサイドカーコンテナで実行されるプロキシに送られます。その後、プロキシはラウンドロビン方式を使用してどちらの MySQL タスクにも接続できます。

負荷分散戦略: ラウンドロビン

外れ値の検知

この機能では、以前に失敗した接続についてプロキシが保持しているデータを使用して、失敗した接続があったホストに新しい接続を送信しないようにします。Service Connect は、プロキシの外れ値検出機能を設定して、パッシブなヘルスチェックを提供します。

前の例を使用すると、プロキシはいすれかの MySQL タスクに接続できます。プロキシが特定の MySQL タスクに複数の接続を行い、直近 30 秒間以内に 5 つ以上の接続が失敗した場合、プロキシはその MySQL タスクを 30 ~ 300 秒間回避します。

再試行

Service Connect は、プロキシを通過して失敗した接続を再試行するようにプロキシを設定し、2 回目の試行では、前の接続のホストを使用しません。これにより、Service Connect を介した各接続が 1 回限りの理由で失敗することがなくなります。

再試行回数: 2

タイムアウト

Service Connect は、クライアント/サーバーアプリケーションが応答するまで最大時間待機するようにプロキシを構成します。デフォルトのタイムアウト値は 15 秒ですが、更新できます。

任意指定のパラメータ:

idleTimeout - アイドル時に接続がアクティブな状態を維持する時間 (秒)。`0` の値は、`idleTimeout` を無効にします。

`HTTP/HTTP2/GRPC` の `idleTimeout` デフォルトは 5 分です。

`TCP` に対する `idleTimeout` デフォルトは 1 時間です。

perRequestTimeout - リクエストごとにアップストリームが完全なレスポンスで応答するまで待機する時間。0 の値は perRequestTimeout をオフにします。これは、アプリケーションコンテナの appProtocol が HTTP/HTTP2/GRPC の場合にのみ設定できます。デフォルト値は 15 秒です。

> **Note**
>
> idleTimeout を perRequestTimeout より短い時間に設定すると、perRequestTimeout ではなく idleTimeout が到達した時点で接続は終了します。

考慮事項

Service Connect を使用する場合は、次の点を考慮してください。

- Fargate で実行されるタスクが Service Connect を使用するには、Fargate Linux プラットフォームバージョン 1.4.0 以上を使用する必要があります。
- コンテナインスタンスの Amazon ECS エージェントバージョンには 1.67.2 以降が必要です。
- コンテナインスタンスが Service Connect を使用するには、Amazon ECS 最適化 Amazon Linux 2023 AMI バージョン 20230428 以降、または Amazon ECS 最適化 Amazon Linux 2 AMI バージョン 2.0.20221115 を実行する必要があります。これらのバージョンには、Amazon ECS コンテナエージェントに加えて Service Connect エージェントがあります。Service Connect エージェントの詳細については、GitHub の「Amazon ECS Service Connect エージェント」を参照してください。
- コンテナインスタンスには、リソース arn:aws:ecs:*region*:*0123456789012*:task-set/*cluster*/* に対する ecs:Poll のアクセス許可が必要です。ecsInstanceRole を使用している場合は、アクセス許可を追加する必要はありません。AmazonEC2ContainerServiceforEC2Role 管理ポリシーには、必要なアクセス許可があります。詳細については、「Amazon ECS コンテナインスタンスの IAM ロール」を参照してください。
- bridge ネットワークモードを使用し、Service Connect を使用するタスクは、hostname コンテナ定義パラメータをサポートしません。
- Service Connect を使用するには、タスク定義でタスクメモリ制限を設定する必要があります。詳細については、「Service Connect プロキシ」を参照してください。
- コンテナのメモリ制限を設定するタスク定義はサポートされていません。

コンテナにはコンテナメモリ制限を設定できますが、タスクメモリ制限はコンテナメモリ制限の合計よりも大きい数値に設定する必要があります。タスク制限内の追加の CPU とメモリで、コンテナ制限に割り当てられていないものは、Service Connect プロキシコンテナや、コンテナ制限を設定していない他のコンテナによって使用されます。詳細については、「Service Connect プロキシ」を参照してください。

- 同じ AWS アカウントに存在する、または AWS Resource Access Manager を使用して AWS アカウントと共有されている同じリージョンに存在する任意の AWS Cloud Map 名前空間を使用するように Service Connect を設定できます。共有名前空間の使用について詳しくは、「共有 AWS Cloud Map 名前空間を使用した Amazon ECS Service Connect」を参照してください。
- 各サービスは 1 つの名前空間のみに属することができます。
- サービスが作成したタスクのみがサポートされます。
- すべてのエンドポイントは、名前空間内で一意である必要があります。
- すべての検出名は、名前空間内で一意である必要があります。
- アプリケーションが新しいエンドポイントを解決できるようにするには、既存のサービスを再デプロイする必要があります。最新のデプロイ後に名前空間に追加された新しいエンドポイントは、タスク設定には追加されません。詳細については、「the section called “Service Connect コンポーネント”」を参照してください。
- Service Connect は、クラスターが削除されても名前空間を削除しません。AWS Cloud Map で名前空間を削除する必要があります。
- Application Load Balancer のトラフィックは、デフォルトで `awsvpc` ネットワークモードの Service Connect エージェントを経由してルーティングされます。サービス以外のトラフィックを Service Connect エージェントをバイパスさせたい場合は、Service Connect サービス設定の `ingressPortOverride` パラメータを使用してください。
- Service Connect with TLS は `bridge` ネットワークモードをサポートしていません。`awsvpc` ネットワークモードのみサポートされています。
- `awsvpc` モードでは、Service Connect プロキシは IPv4 専用およびデュアルスタック設定のサービスの IPv4 localhost `127.0.0.1` にトラフィックを転送します。IPv6 専用設定のサービスの場合、プロキシはトラフィックを IPv6 localhost `::1` に転送します。サービスが IPv4 専用またはデュアルスタック設定から IPv6 専用に更新されたときに、両方の localhost をリッスンするようにアプリケーションを設定することをお勧めします。詳細については、「EC2 の Amazon ECS タスクネットワークオプション」および「Fargate における Amazon ECS タスクのネットワークオプション」を参照してください。

Service Connect では、次に示すの機能はサポートされていません。

- Windows コンテナ
- HTTP 1.0
- スタンドアロンのタスク
- CodeDeploy および外部デプロイタイプを搭載したブルー/グリーン使用するサービス
- Amazon ECS Anywhere の `External` コンテナインスタンスは、Service Connect ではサポートされていません。
- PPv2
- FIPS モード

Amazon ECS Service Connect 設定の概要

Service Connect を使用する場合、リソースで設定する必要があるパラメータがあります。

Amazon ECS リソースの設定パラメータを以下の表に示します。

パラメータの場所	アプリケーションタイプ	説明	必須
タスク定義	クライアント	クライアントタスク定義には、Service Connect が利用できる変更はありません。	該当なし
タスク定義	クライアント/サーバー	サーバーは、コンテナの `portMappings` のポートに `name` フィールドを追加する必要があります。詳細については、`portMappings` を参照してください。	はい
タスク定義	クライアント/サーバー	サーバーは、オプションでアプリケーションプロトコル (HTTP など) を提供して、サーバーアプリケーションのプロトコル固有のメトリクス (HTTP `5xx` など) を受け取ることができます。	いいえ

パラメータの場所	アプリケーションタイプ	説明	必須
サービスの定義	クライアント	クライアントサービスは、名前空間の結合を設定するには `serviceConnectConfiguration` を追加する必要があります。この名前空間には、このサービスが検出する必要のあるすべてのサーバーサービスが含まれていなければなりません。詳細については、「serviceConnectConfiguration」を参照してください。	はい
サービスの定義	クライアント/サーバー	サーバーサービスは、サービスが利用できる DNS 名、ポート番号、名前空間を設定するために `serviceConnectConfiguration` を追加する必要があります。詳細については、「serviceConnectConfiguration」を参照してください。	はい
クラスター	クライアント	クラスターはデフォルトの Service Connect 名前空間を追加できます。クラスター内の新しいサービスは、サービスで Service Connect が設定されている場合は名前空間を継承します。	いいえ
クラスター	クライアント/サーバー	サーバーサービスに適用されるクラスターには、Service Connect が利用できる変更はありません。サーバーのタスク定義とサービスは、それぞれの設定を行う必要があります。	該当なし

Service Connect の設定手順の概要

次の手順では、Service Connect の設定方法の概要を示します。

> **Important**
>
> - Service Connect がお客様のアカウントに AWS Cloud Map サービスを作成します。手動でのインスタンスの登録/登録解除、インスタンス属性の変更、サービスの削除を行ってこれらの AWS Cloud Map リソースを変更すると、アプリケーショントラフィックまたは以降のデプロイで予期しない動作が発生することがあります。
> - Service Connect は、タスク定義内のリンクをサポートしていません。

1. タスク定義のポートマッピングにポート名を追加します。さらに、アプリケーションのレイヤー 7 プロトコルを識別して、追加のメトリクスを取得できます。
2. AWS Cloud Map 名前空間を使用してクラスターを作成するか、共有名前空間を使用する、または名前空間を個別に作成します。単純に構成できるように、名前空間に必要な名前でクラスターを作成し、名前空間に同名を指定します。この場合、Amazon ECS は必要な設定で新しい HTTP 名前空間を作成します。Service Connect は Amazon Route 53 では DNS ホストゾーンを使用または作成しません。
3. サービスを設定して、名前空間内に Service Connect エンドポイントを作成します。
4. サービスをデプロイしてエンドポイントを作成します。Amazon ECS は各タスクに Service Connect プロキシコンテナを追加し、AWS Cloud Map に Service Connect エンドポイントを作成します。このコンテナはタスク定義では設定されていないため、タスク定義を変更せずに再利用して、同じ名前空間または複数の名前空間に複数のサービスを作成できます。
5. クライアントアプリをサービスとしてデプロイしてエンドポイントに接続します。Amazon ECS は Service Connect エンドポイントへの接続を各タスクの Service Connect プロキシ経由で行います。

   アプリケーションは Service Connect エンドポイントへの接続にのみプロキシを使用します。プロキシを使用するための追加の設定はありません。プロキシは、ラウンドロビン負荷分散、外れ値検出、再試行を実行します。プロキシの詳細については、「Service Connect プロキシ」を参照してください。
6. Amazon CloudWatch の Service Connect プロキシ経由のトラフィックを監視します。

クラスターの設定

クラスターの作成時または更新時に、Service Connect のデフォルトの名前空間を設定できます。デフォルトとして指定する名前空間名は、同じ AWS リージョンとアカウントに存在するか、同じ

AWS リージョンに存在し、AWS Resource Access Manager を使用して別の AWS アカウントと共有するようにできます。

クラスターを作成してデフォルトの Service Connect 名前空間を指定した場合、Amazon ECS が名前空間を作成する間、そのクラスターは PROVISIONING ステータスで待機します。クラスターのステータスには、名前空間のステータスを示す attachment が表示されます。アタッチメントは、デフォルトでは AWS CLI に表示されません。表示するには、--include ATTACHMENTS を追加する必要があります。

AWS RAM を使用して AWS アカウントと共有されている名前空間を使用する場合は、クラスター設定で名前空間の Amazon リソースネーム (ARN) を指定します。共有 AWS Cloud Map 名前空間の詳細については、「共有 AWS Cloud Map 名前空間を使用した Amazon ECS Service Connect」を参照してください。

サービス設定

Service Connect は、必要最小限の設定で済むように設計されています。タスク定義で、Service Connect で使用する各ポートマッピングの名前を設定する必要があります。サービスでクライアントサービスを作成するには、Service Connect をオンにして AWS アカウントの名前空間または共有名前空間を選択する必要があります。クライアント/サーバーサービスを作成するには、いずれかのポートマッピングの名前と一致する単一の Service Connect サービス設定を追加する必要があります。Amazon ECS はタスク定義からポート番号とポート名を再利用して、Service Connect サービスとエンドポイントを定義します。これらの値を上書きするには、コンソールで他のパラメータ [Discovery] (検出)、[DNS]、[Port] (ポート) を使用するか、discoveryName、clientAliases それぞれを Amazon ECS API で使用します。

初期ヘルスチェックの設定

Service Connect は、トラフィックをルーティングする前にタスクが正常であることを確認します。タスクが起動すると (デプロイ、スケーリング、または置き換え中)、Service Connect はタスクの状態をモニタリングして、トラフィックを受け入れる準備ができていることを確認します。この動作を有効にするには、タスク定義で必須コンテナのヘルスチェックを定義する必要があります。

初期ヘルスチェックの動作は、タスクがトラフィックを受け入れる準備ができた状態に達するまでの潜在的な遅延を考慮します。

- タスクが HEALTHY の場合は、トラフィックに対して直ちに使用できます。
- タスクのヘルスが UNKNOWN の場合、Service Connect はタスクの必須コンテナのコンテナヘルスチェック設定 (「the section called “ヘルスチェック”」を参照) に従って、最大 8 minutes のタイムアウトを計算してから、UNKNOWN のままであってもトラフィックで利用できるようにします。

- タスクが UNHEALTHY になっているときは、Amazon ECS が代替タスクを起動する場合があります。正常なタスクが利用できない場合は、サービスの設定に基づいてデプロイがロールバックされる可能性があります。

進行中のすべてのトラフィックについて、Service Connect は外れ値検出に基づくパッシブヘルスチェックを使用してトラフィックを効率的にルーティングします。

共有 AWS Cloud Map 名前空間を使用した Amazon ECS Service Connect

Amazon ECS Service Connect は、同じ AWS リージョン内の複数の AWS アカウントで共有 AWS Cloud Map 名前空間を使用することをサポートしています。この機能を使用すると、異なる AWS アカウントで実行されているサービスが Service Connect を介して相互に検出して通信することが可能な分散アプリケーションを作成できます。共有名前空間は、安全なクロスアカウントリソース共有を可能にする AWS Resource Access Manager (AWS RAM) を使用して管理されます。共有名前空間の詳細については、「AWS Cloud Map デベロッパーガイド」の「クロスアカウント AWS Cloud Map 名前空間共有」を参照してください。

> **Important**
>
> Service Connect が名前空間で適切に動作するには、`AWSRAMPermissionCloudMapECSFullPermission` マネージドアクセス許可を使用して名前空間を共有する必要があります。

Service Connect で共有 AWS Cloud Map 名前空間を使用する場合は、複数の AWS アカウントのサービスが同じサービス名前空間に参加できます。これは、複数の AWS アカウントが存在する組織で、セキュリティと分離を維持しながら、アカウントの境界を越えてサービス間通信を維持する必要がある場合に特に有効です。

> **Note**
>
> 異なる VPC にあるサービスと通信するには、VPC 間接続を設定する必要があります。これは、VPC ピアリング接続を使用して実現できます。詳細については、「Amazon Virtual Private Cloud ピアリングガイド」の「VPC ピアリング接続の作成または削除」を参照してください。

Amazon ECS Service Connect での共有 AWS Cloud Map 名前空間の使用

Service Connect の共有 AWS Cloud Map 名前空間の設定には、名前空間所有者による名前空間の作成、AWS Resource Access Manager (AWS RAM) を介した所有者による共有、リソース共有を受け入れるコンシューマー、共有名前空間を使用するように Service Connect を設定するコンシューマーのステップが含まれます。

ステップ 1: AWS Cloud Map 名前空間を作成する

名前空間所有者は、他のアカウントと共有される AWS Cloud Map 名前空間を作成します。

AWS マネジメントコンソールを使用して共有するための名前空間を作成するには

1. https://console.aws.amazon.com/cloudmap/ で AWS Cloud Map コンソールを開きます。
2. [名前空間の作成] を選択します。
3. 名前空間名を入力します。この名前は、すべての参加アカウントのサービスによって使用されます。
4. [名前空間タイプ] で、ユースケースに適したタイプを選択します。
   - API コール – DNS 機能のないサービス検出用の HTTP 名前空間。
   - VPC での API コールと DNS クエリ – VPC 内のプライベート DNS クエリを使用したサービス検出用のプライベート DNS 名前空間。
   - API コールとパブリック DNS クエリ – パブリック DNS クエリによるサービス検出用のパブリック DNS 名前空間。
5. [名前空間の作成] を選択します。

ステップ 2: AWS RAM を使用して名前空間を共有する

名前空間の所有者は、AWS RAM を使用して名前空間を他の AWS アカウントと共有します。

AWS RAM コンソールを使用して名前空間を共有するには

1. AWS RAM コンソール (https://console.aws.amazon.com/ram/) を開きます。
2. [リソースの共有の作成] を選択します。
3. [Name] (名前) に、リソース共有のわかりやすい名前を入力します。
4. [リソース] セクションで、次の操作を行います。
   a. [リソースタイプ]で、クラウドマップ名前空間を選択します。

b. 前のステップで作成した名前空間を選択します。

5. [マネージドアクセス許可] セクションで、[AWSRAMPermissionCloudMapECSFullPermission] を指定します。

> **Important**
>
> Service Connect が名前空間で適切に動作するには、`AWSRAMPermissionCloudMapECSFullPermission` マネージドアクセス許可を使用して名前空間を共有する必要があります。

6. [プリンシパル] セクションで、名前空間を共有する AWS アカウントを指定します。アカウント ID または組織単位 ID を入力できます。
7. [リソースの共有の作成] を選択します。

ステップ 3: リソース共有を受け入れる

名前空間コンシューマーアカウントが共有名前空間を使用するには、リソース共有の招待を受け入れる必要があります。

AWS RAM コンソールを使用してリソース共有の招待を承諾するには

1. コンシューマーアカウントで、AWS RAM コンソール (https://console.aws.amazon.com/ram/) を開きます。
2. ナビゲーションペインで [Shared with me] (自分と共有)、[リソース共有] の順に選択します。
3. リソース共有の招待を選択し、[リソースの共有を承認] を選択します。
4. 同意したら、リソースの詳細から共有名前空間 ARN を書き留めます。この ARN は、Service Connect サービスを設定する際に使用します。

ステップ 4: 共有名前空間を使用して Amazon ECS サービスを設定する

共有名前空間を受け入れると、名前空間コンシューマーは、共有名前空間を使用するように Amazon ECS サービスを設定できます。設定は通常の名前空間の使用と類似していますが、名前の代わりに名前空間 ARN を指定する必要があります。サービス作成手順の詳細については、「Amazon ECS のローリング更新デプロイの作成」を参照してください。

AWS マネジメントコンソールを使用して共有名前空間でサービスを作成するには

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
2. [クラスター] ページで、サービスを作成するクラスターを選択します。
3. [サービス] で [作成] を選択します。
4. ワークロードに応じて他の詳細情報を入力したら、[Service Connect] セクションで [Service Connect を使う] を選択します。
5. [名前空間] には、共有名前空間の完全な ARN を入力します。

   ARN 形式は `arn:aws:servicediscovery:`*`region`*`:`*`account-id`*`:namespace/`*`namespace-id`* です。
6. 必要に応じて、サービスタイプ (クライアントまたはクライアントサーバー) の残りの Service Connect 設定を構成します。
7. サービスの作成プロセスを完了します。

AWS CLI または AWS SDK を使用してサービスを設定するには、`serviceConnectConfiguration` の `namespace` パラメータで共有名前空間 ARN を指定します。

```
aws ecs create-service \
    --cluster my-cluster \
    --service-name my-service \
    --task-definition my-task-def \
    --service-connect-configuration '{
        "enabled": true,
        "namespace": "arn:aws:servicediscovery:us-west-2:123456789012:namespace/ns-
abcdef1234567890",
        "services": [{
            "portName": "web",
            "discoveryName": "my-service",
            "clientAliases": [{
                "port": 80,
                "dnsName": "my-service"
            }]
        }]
    }'
```

考慮事項

Service Connect で共有 AWS Cloud Map 名前空間を使用する場合は、次の点を考慮してください。

- AWS RAM は、共有名前空間を使用する AWS リージョンで利用できる必要があります。
- 共有名前空間は、Amazon ECS のサービスおよびクラスターと同じ AWS リージョンに存在する必要があります。
- 共有名前空間で Service Connect を設定する場合は、ID ではなく名前空間 ARN を使用する必要があります。
- HTTP、プライベート DNS、パブリック DNS 名前空間のすべての名前空間タイプがサポートされています。
- 共有名前空間へのアクセスが取り消されると、名前空間とのやり取りを必要とする Amazon ECS オペレーション (`CreateService`、`UpdateService`、`ListServicesByNamespace` など) は失敗します。共有名前空間のアクセス許可に関する問題のトラブルシューティングの詳細については、「共有 AWS Cloud Map 名前空間を使用した Amazon ECS Service Connect のトラブルシューティング」を参照してください。
- 共有プライベート DNS 名前空間で DNS クエリを使用するサービス検出の場合:
  - 名前空間所有者は、名前空間に関連付けられたプライベートホストゾーンの ID とコンシューマーの VPC を使用して `create-vpc-association-authorization` を呼び出す必要があります。

    ```
    aws route53 create-vpc-association-authorization --hosted-zone-id Z1234567890ABC --
    vpc VPCRegion=us-east-1,VPCId=vpc-12345678
    ```

  - 名前空間コンシューマーは、プライベートホストゾーンの ID を使用して `associate-vpc-with-hosted-zone` を呼び出す必要があります。

    ```
    aws route53 associate-vpc-with-hosted-zone --hosted-zone-id Z1234567890ABC --vpc
     VPCRegion=us-east-1,VPCId=vpc-12345678
    ```

- リソース共有を管理できるのは、名前空間所有者のみです。
- 名前空間コンシューマーは、共有名前空間内でサービスを作成および管理できますが、名前空間自体を変更することはできません。
- 検出名は、サービスを作成するアカウントを問わず、共有名前空間内で一意であることが必要です。
- 共有名前空間内のサービスは、名前空間にアクセスできる他の AWS アカウントのサービスを検出して接続できます。

- Service Connect で TLS を有効にし、共有名前空間を使用する場合、AWS Private CA 認証局 (CA) は名前空間に限定されます。共有名前空間へのアクセスが取り消されると、CA へのアクセスは停止します。
- 共有名前空間を使用する場合、名前空間の所有者とコンシューマーはデフォルトでクロスアカウントの Amazon CloudWatch メトリクスにアクセスできません。ターゲットメトリクスは、クライアントサービスを所有するアカウントにのみ発行されます。クライアントサービスを所有するアカウントは、クライアント/サーバーサービスを所有するアカウントが受信したメトリクスにアクセスできません。その逆も同様です。メトリクスへのクロスアカウントアクセスを許可するには、CloudWatch クロスアカウントオブザーバビリティを設定します。クロスアカウントオブザーバビリティの設定の詳細については、「Amazon CloudWatch ユーザーガイド」の「CloudWatch のクロスアカウントオブザーバビリティ」を参照してください。Service Connect の CloudWatch メトリクスの詳細については、「Amazon ECS CloudWatch メトリクス」を参照してください。

共有 AWS Cloud Map 名前空間を使用した Amazon ECS Service Connect のトラブルシューティング

次の情報を使用して、共有 AWS Cloud Map 名前空間と Service Connect の問題をトラブルシューティングします。エラーメッセージの検索の詳細については、「Amazon ECS のトラブルシューティング」を参照してください。

アクセス許可の問題に関連するエラーメッセージは、アクセス許可が欠落していることが原因で、または名前空間へのアクセスが取り消された場合に表示されます。

> **Important**
>
> Service Connect が名前空間で適切に動作するには、`AWSRAMPermissionCloudMapECSFullPermission` マネージドアクセス許可を使用して名前空間を共有する必要があります。

エラーメッセージは、次のいずれかの形式で表示されます。

<OperationName> オペレーションを呼び出すときにエラー (ClientException) が発生しました: ユーザー: arn:aws:iam::<account-id>:user/<user-name> にはリソース: <ResourceArn> に対して <ActionName> を実行するための権限が付与されていません: これは、リソースベースのポリシーで <ActionName> アクションが許可されていないためです

この形式のエラーメッセージは、次のシナリオにおいて生成される可能性があります。

クラスターの作成または更新の失敗

これらの問題は、`CreateCluster` や `UpdateCluster` などの Amazon ECS オペレーションが AWS Cloud Map アクセス許可が欠落しているために失敗した場合に発生します。このオペレーションには、次の AWS Cloud Map アクションに対するアクセス許可が必要です。

- `servicediscovery:GetNamespace`

リソース共有の招待がコンシューマーアカウントで承諾され、Service Connect 設定で正しい名前空間 ARN が使用されていることを確認します。

サービスの作成または更新の失敗

これらの問題は、`CreateService` や `UpdateService` などの Amazon ECS オペレーションが AWS Cloud Map アクセス許可が欠落しているために失敗した場合に発生します。このオペレーションには、次の AWS Cloud Map アクションに対するアクセス許可が必要です。

- `servicediscovery:CreateService`
- `servicediscovery:GetNamespace`
- `servicediscovery:GetOperation` (新しい AWS Cloud Map サービスを作成する場合)
- `servicediscovery:GetService` (AWS Cloud Map サービスが既に存在する場合)

リソース共有の招待がコンシューマーアカウントで承諾され、Service Connect 設定で正しい名前空間 ARN が使用されていることを確認します。

`ListServicesByNamespace` オペレーションが失敗する

この問題は、Amazon ECS `ListServicesByNamespace` オペレーションが失敗した場合に発生します。このオペレーションには、次の AWS Cloud Map アクションに対するアクセス許可が必要です。

- `servicediscovery:GetNamespace`

この問題を解決するには。

- コンシューマーアカウントに `servicediscovery:GetNamespace` アクセス許可が付与されていることを確認します。
- 名前ではなく API を呼び出す場合は、名前空間 ARN を使用します。
- リソース共有がアクティブであり、招待が承諾されていることを確認します。

ユーザー: <iam-user> は、ID ベースのポリシーで明示的に拒否されたリソース: <ResourceArn> に対して <ActionName> を実行することを承認されていません。

この形式のエラーメッセージは、次のシナリオにおいて生成される可能性があります。

サービスの削除が失敗し、**DRAINING** 状態のままになる

この問題は、名前空間へのアクセスが取り消されたことで `servicediscovery:DeleteService` アクセス許可がなくなり、それが原因で Amazon ECS `DeleteService` オペレーションが失敗する場合に発生します。サービスは最初は正常に削除されたように見えますが、DRAINING 状態でスタックします。このエラーメッセージは Amazon ECS サービスイベントとして表示されます。

この問題を解決するには、名前空間所有者が名前空間をコンシューマーアカウントと共有して、サービスの削除を完了できるようにする必要があります。

サービスのタスクが実行に失敗する

この問題は、アクセス許可が欠落しているためにタスクが開始されない場合に発生します。このエラーメッセージは、停止したタスクエラーとして表示されます。詳細については、「Amazon ECS の停止したタスクのエラーを解決する」を参照してください。

タスクを実行するには、次の AWS Cloud Map アクションが必要です。

- `servicediscovery:GetOperation`
- `servicediscovery:RegisterInstance`

コンシューマーアカウントに必要なアクセス許可が付与されており、共有名前空間にアクセスできることを確認します。

タスクがクリーンに停止しないか、**DEACTIVATING** または **DEPROVISIONING** の状態のままになる

この問題は、アクセス許可が欠落しているためにシャットダウン中にタスクが AWS Cloud Map サービスから登録解除できなかったことが原因で発生します。エラーは、`DescribeTasks` API を使用して取得できるタスクアタッチメントに `statusReason` として表示されます。詳細については、「Amazon Elastic Container Service API リファレンス」の「DescribeTasks」を参照してください。

タスクを停止するには、次の AWS Cloud Map アクションが必要です。

- `servicediscovery:DeregisterInstance`
- `servicediscovery:GetOperation`

共有名前空間へのアクセスが取り消された場合、名前空間へのアクセスが復元されるまでタスクは DEACTIVATING または DEPROVISIONING の状態のままになる可能性があります。名前空間所有者に名前空間へのアクセスの復元をリクエストします。

Amazon ECS Service Connect アクセスログ

Amazon ECS Service Connect は、Service Connect プロキシによって処理された個別のリクエストに関する詳細なテレメトリを提示するアクセスログをサポートします。アクセスログは、HTTP メソッド、パス、レスポンスコード、フラグ、タイミング情報などのリクエストごとのトラフィックメタデータをキャプチャすることで、既存のアプリケーションログを補完します。これにより、リクエストレベルのトラフィックパターンとサービスインタラクションをより詳細に観察できるため、効果的なトラブルシューティングとモニタリングが可能になります。

アクセスログを有効にするには、serviceConnectConfiguration オブジェクトで logConfiguration オブジェクトと accessLogConfiguration オブジェクトの両方を指定します。ログの形式と、ログについて accessLogConfiguration にクエリパラメータが含まれるかどうかを設定できます。ログは、logConfiguration で指定されたログドライバーによって送信先ロググループに配信されます。

```
{
    "serviceConnectConfiguration": {
        "enabled": true,
        "namespace": "myapp.namespace",
        "services": [
            ...
        ],
        "logConfiguration": {
            "logDriver": "awslogs",
            "options": {
                "awslogs-group": "my-envoy-log-group",
                "awslogs-region": "us-west-2",
                "awslogs-stream-prefix": "myapp-envoy-logs"
            }
        },
         "accessLogConfiguration": {
            "format": "TEXT",
            "includeQueryParameters": "ENABLED"
        }
    }
}
```

考慮事項

アクセスログへのアクセスを有効にする場合は、次の点を考慮してください。

- アクセスログとアプリケーションログはどちらも /dev/stdout に書き込まれます。アクセスログをアプリケーションログから分離するには、カスタムの Fluent Bit または Fluentd の設定で awsfirelens ログドライバーを使用することをお勧めします。
- awslogs ログドライバーを使用して、アプリケーションとアクセスログを同じ CloudWatch の宛先に送信することをお勧めします。
- アクセスログは、プラットフォームバージョン 1.4.0 以降を使用する Fargate サービスでサポートされています。
- リクエスト ID やトークンなどのクエリパラメータは、デフォルトでアクセスログから除外されます。アクセスログにクエリパラメータが含まれるようにするには、includeQueryParameters を "ENABLED" に設定します。

アクセスログの形式

アクセスログは、JSON 形式のディクショナリまたはテキスト形式の文字列のいずれかでフォーマットできますが、異なるタイプのアクセスログでサポートされているコマンド演算子は異なります。

HTTP アクセスログ

HTTP ログには、デフォルトで次のコマンド演算子が含まれています。

Text

```
[%START_TIME%] "%REQ(:METHOD)% %REQ(X-ENVOY-ORIGINAL-PATH?:PATH)% %PROTOCOL%"
%RESPONSE_CODE% %BYTES_RECEIVED% %BYTES_SENT% %DURATION%
%RESP(X-ENVOY-UPSTREAM-SERVICE-TIME)% "%REQ(X-FORWARDED-FOR)%" "%REQ(USER-AGENT)%"
"%REQ(X-REQUEST-ID)%" "%REQ(:AUTHORITY)%" "%UPSTREAM_HOST%"\n
```

JSON

```
{
  "start_time": "%START_TIME%",
  "method": "%REQ(:METHOD)%",
  "path": "%REQ(X-ENVOY-ORIGINAL-PATH?:PATH)%",
  "protocol": "%PROTOCOL%",
  "response_code": "%RESPONSE_CODE%",
  "bytes_received": "%BYTES_RECEIVED%",
```

```
  "bytes_sent": "%BYTES_SENT%",
  "duration_ms": "%DURATION%",
  "upstream_service_time": "%RESP(X-ENVOY-UPSTREAM-SERVICE-TIME)%",
  "forwarded_for": "%REQ(X-FORWARDED-FOR)%",
  "user_agent": "%REQ(USER-AGENT)%",
  "request_id": "%REQ(X-REQUEST-ID)%",
  "authority": "%REQ(:AUTHORITY)%",
  "upstream_host": "%UPSTREAM_HOST%"
}
```

HTTP2 アクセスログ

HTTP ログに含まれるコマンド演算子に加えて、HTTP2 ログにはデフォルトで %STREAM_ID% 演算子が含まれます。

Text

```
[%START_TIME%] "%REQ(:METHOD)% %REQ(X-ENVOY-ORIGINAL-PATH?:PATH)% %PROTOCOL%"
%RESPONSE_CODE% %BYTES_RECEIVED% %BYTES_SENT% %DURATION%
%RESP(X-ENVOY-UPSTREAM-SERVICE-TIME)% "%REQ(X-FORWARDED-FOR)%" "%REQ(USER-AGENT)%"
"%REQ(X-REQUEST-ID)%" "%REQ(:AUTHORITY)%" "%UPSTREAM_HOST%" "%STREAM_ID%"\n
```

JSON

```
{
  "start_time": "%START_TIME%",
  "method": "%REQ(:METHOD)%",
  "path": "%REQ(X-ENVOY-ORIGINAL-PATH?:PATH)%",
  "protocol": "%PROTOCOL%",
  "response_code": "%RESPONSE_CODE%",
  "bytes_received": "%BYTES_RECEIVED%",
  "bytes_sent": "%BYTES_SENT%",
  "duration": "%DURATION%",
  "upstream_service_time": "%RESP(X-ENVOY-UPSTREAM-SERVICE-TIME)%",
  "forwarded_for": "%REQ(X-FORWARDED-FOR)%",
  "user_agent": "%REQ(USER-AGENT)%",
  "request_id": "%REQ(X-REQUEST-ID)%",
  "authority": "%REQ(:AUTHORITY)%",
  "upstream_host": "%UPSTREAM_HOST%",
  "stream_id": "%STREAM_ID%"
}
```

gRPC アクセスログ

HTTP ログに含まれるコマンド演算子に加えて、gRPC アクセスログにはデフォルトで %STREAM_ID% および %GRPC_STATUS()% の演算子が含まれます。

Text

```
[%START_TIME%] "%REQ(:METHOD)% %REQ(X-ENVOY-ORIGINAL-PATH?:PATH)% %PROTOCOL%"
%RESPONSE_CODE% %GRPC_STATUS()% %BYTES_RECEIVED% %BYTES_SENT% %DURATION%
%RESP(X-ENVOY-UPSTREAM-SERVICE-TIME)% "%REQ(X-FORWARDED-FOR)%" "%REQ(USER-AGENT)%"
"%REQ(X-REQUEST-ID)%" "%REQ(:AUTHORITY)%" "%UPSTREAM_HOST%" "%STREAM_ID%"\n
```

JSON

```
{
  "start_time": "%START_TIME%",
  "method": "%REQ(:METHOD)%",
  "path": "%REQ(X-ENVOY-ORIGINAL-PATH?:PATH)%",
  "protocol": "%PROTOCOL%",
  "response_code": "%RESPONSE_CODE%",
  "grpc_status": "%GRPC_STATUS()%",
  "bytes_received": "%BYTES_RECEIVED%",
  "bytes_sent": "%BYTES_SENT%",
  "duration": "%DURATION%",
  "upstream_service_time": "%RESP(X-ENVOY-UPSTREAM-SERVICE-TIME)%",
  "forwarded_for": "%REQ(X-FORWARDED-FOR)%",
  "user_agent": "%REQ(USER-AGENT)%",
  "request_id": "%REQ(X-REQUEST-ID)%",
  "authority": "%REQ(:AUTHORITY)%",
  "upstream_host": "%UPSTREAM_HOST%",
  "stream_id": "%STREAM_ID%"
}
```

TCP アクセスログ

次のコマンド演算子は、デフォルトで TCP アクセスログに含まれています。

Text

```
[%START_TIME%] %DOWNSTREAM_REMOTE_ADDRESS% %DOWNSTREAM_REMOTE_PORT%
%BYTES_RECEIVED% %BYTES_SENT% %DURATION%
```

```
%CONNECTION_TERMINATION_DETAILS% %CONNECTION_ID%\n
```

JSON

```
{
  "start_time": "%START_TIME%",
  "downstream_remote_address": "%DOWNSTREAM_REMOTE_ADDRESS%",
  "downstream_remote_port": "%DOWNSTREAM_REMOTE_PORT%",s
  "bytes_received": "%BYTES_RECEIVED%",
  "bytes_sent": "%BYTES_SENT%",
  "duration": "%DURATION%",
  "connection_termination_details": "%CONNECTION_TERMINATION_DETAILS%",
  "connection_id": %CONNECTION_ID%
}
```

これらのコマンド演算子の詳細については、Envoy ドキュメントの「コマンド演算子」を参照してください。

Amazon ECS Service Connect のアクセスログの有効化

Service Connect を使用する Amazon ECS サービスでは、アクセスログはデフォルトでは有効になっていません。以下の方法でアクセスログを有効にできます。

AWS CLI を使用してアクセスログを有効にする

次のコマンドは、サービスの作成時に `accessLogConfiguration` を指定することで AWS CLI を使用して、Amazon ECS サービスのアクセスログを有効にする方法を示しています。

```
aws ecs create-service \
    --cluster my-cluster \
    --service-name my-service \
    --task-definition my-task-def \
    --service-connect-configuration '{
        "enabled": true,
        "namespace": "arn:aws:servicediscovery:us-west-2:123456789012:namespace/ns-
abcdef1234567890",
        "services": [{
            "portName": "web",
            "discoveryName": "my-service",
            "clientAliases": [{
                "port": 80,
                "dnsName": "my-service"
```

```
            }]
        }],
        "logConfiguration": {
            "logDriver": "awslogs",
            "options": {
                "awslogs-group": "my-envoy-log-group",
                "awslogs-region": "us-west-2",
                "awslogs-stream-prefix": "myapp-envoy-logs"
            }
        },
         "accessLogConfiguration": {
            "format": "TEXT",
            "includeQueryParameters": "ENABLED"
        }
    }'
```

コンソールを使用してアクセスログを有効にする

サービス作成手順の詳細については、「Amazon ECS のローリング更新デプロイの作成」を参照してください。

AWS マネジメントコンソールを使用して共有名前空間でサービスを作成するには

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
2. [クラスター] ページで、サービスを作成するクラスターを選択します。
3. [サービス] で [作成] を選択します。
4. ワークロードに応じて他の詳細情報を入力したら、[Service Connect] セクションで [Service Connect を使う] を選択します。
5. 必要に応じて、サービスタイプ (クライアントまたはクライアントサーバー) の Service Connect 設定を構成します。
6. [アクセスログ設定] を展開します。[形式] で、[JSON] または [TEXT] を選択します。
7. アクセスログにクエリパラメータが含まれるようにするには、[クエリパラメータを含める] を選択します。
8. サービスの作成プロセスを完了します。

Amazon ECS Service Connect ゾーン認識ルーティング

Amazon ECS Service Connect ゾーン認識ルーティングでは、トラフィックはクライアントと同じアベイラビリティーゾーン (AZ) 内に留まります。これにより、追加のインフラストラクチャやアプ

リケーションコードの変更を必要とせずに、AZ 間のデータ転送コストとレイテンシーを削減できます。エンドポイントが AZ 間で均等に分散されている場合、トラフィックの 80% 以上がローカルのままになります。また、トラフィックの重みはエンドポイントのスケールに合わせて動的に調整され、ターゲットサービス間で負荷のバランスが保たれます。Service Connect を使用するすべての新規および既存のサービスでは、ゾーン認識ルーティングがデフォルトで有効になっています。

ゾーン認識ルーティングの仕組み

このアルゴリズムは、Service Connect プロキシの Envoy ゾーン認識ルーティング機能を使用して、エンドポイントのローカル性に基づいてトラフィックをルーティングします。

1. エンドポイント検出 - プロキシは、送信先サービスで使用可能なすべてのエンドポイントとその AZ 配置を検出します。
2. 同じ AZ の優先順位付け - プロキシは、送信元 (クライアント) と送信先 (サーバー) の AZ 間のエンドポイント分布の比率に基づいて、同じ AZ にとどまるトラフィックの割合を計算します。エンドポイントがゾーン間で均等に分散されている場合、同じ AZ へのトラフィックは通常 80% を超えます。
3. 残余容量ルーティング - 同じ AZ にとどまれないトラフィックは、残余容量に基づいて他の AZ に分散されます。プロキシは各ゾーンの残余容量を計算し、正の残余容量を持つゾーン間でクロス AZ トラフィックを比例的に分割します。エンドポイントがスケールアップまたはスケールダウンすると、プロキシはルーティングの重みをリアルタイムで再計算します。
4. 過負荷保護とフォールバック - 単一の AZ への過負荷を防ぐために、プロキシは送信先サービスで使用可能なエンドポイントとして、少なくとも AZ 数の 2 倍を必要とします。3 AZ リージョンの場合、これは少なくとも 6 つのエンドポイントが必要であることを意味します。このしきい値を下回ると、ゾーン認識ルーティングは無効化され、ラウンドロビン負荷分散を使用してトラフィックが均等に分散されます。エンドポイント数がしきい値を超えると、ルーティングは自動的に再有効化されます。同じ AZ 内のエンドポイントが使用できなくなった場合、可用性を維持するためにトラフィックは他の AZ 間で再分散されます。

Amazon EC2 でゾーン認識ルーティングを検証する

Docker ランタイムを使用する Amazon EC2 コンテナインスタンスでは、Service Connect エージェントコンテナ内の Envoy プロキシ統計を調べて、ゾーン認識ルーティングがアクティブであることを確認します。コンテナインスタンスは、ECS 最適化 AMI バージョン `20260323` 以降を使用する必要があります。この手順では、AWS Systems Manager Session Manager のホストアクセスと `docker exec` が必要です。AWS Fargate または `containerd` を使用するコンテナインスタンスの

場合は、Amazon Virtual Private Cloud フローログを使用して、「the section called “ゾーン認識ルーティングのモニタリング”」の説明に従ってルーティング動作を検証します。

Docker を使用して Amazon EC2 のゾーン認識ルーティングを検証するには

1. AWS Systems Manager Session Manager を使用してコンテナインスタンスに接続します。*instance-id* を、コンテナインスタンス ID に置き換えます。

```
aws ssm start-session --target instance-id
```

2. Service Connect エージェントコンテナでシェルを開きます。

```
sudo docker exec -it $(sudo docker ps --filter "name=ecs-service-connect" -q | head
 -1) /bin/sh
```

3. Envoy 管理インターフェイスからゾーンルーティング統計をクエリします。

```
curl --unix-socket /tmp/envoy_admin.sock http://unix/stats | grep "lb_zone"
```

4. 出力の検証 ゾーン認識ルーティングがアクティブな正常なデプロイでは、クロスゾーンリクエスト数はゼロになります。

```
cluster.my-service.lb_zone_routing_cross_zone: 0
cluster.my-service.lb_zone_cluster_too_small: 0
```

次の表は、ゾーンルーティングの主要なメトリクスを説明しています。

メトリクス	説明
lb_zone_routing_cross_zone	別の AZ のエンドポイントにルーティングされたリクエストの数。持続的な値が 0 の場合、すべてのトラフィックが同じ AZ 内にとどまることを保証します。
lb_zone_cluster_too_small	エンドポイント数が最小しきい値を下回ったため、ゾーン認識ルーティングがバイパスされた回数。初期デプロイ時のゼロ以外の値は想定内であり、エンドポイントが正常になると解消さ

メトリクス	説明
	れます。安定したデプロイでは 0 が表示されます。

ゾーン認識ルーティングのモニタリング

`az-id` フィールドで Amazon Virtual Private Cloud フローログを使用して、ネットワークレベルでトラフィックパターンを観察します。これにより、各フローレコードの発信元と終端の AZ が表示され、同一 AZ トラフィックとクロス AZ トラフィックの比率を測定できます。AWS Cost Explorer を使用して、ゾーン認識ルーティングを有効にするためにサービスを再デプロイする前後のクロス AZ データ転送料金を追跡することもできます。

フローログの設定の詳細については、「Amazon Virtual Private Cloud ユーザーガイド」の「VPC フローログ」を参照してください。

考慮事項

ゾーン認識ルーティングを使用する場合は、次の点に注意してください。

- ゾーン認識ルーティングは、すべての起動タイプで機能します。
- ゾーン認識ルーティングは、AWS Resource Access Manager を介して共有されるクロスアカウント Service Connect 名前空間と互換性があります。
- 既存のサービス (クライアントとサーバーの両方) では、ゾーン認識ルーティングを有効化するために 1 回限りの再デプロイが必要です。最初の再デプロイ後、ルーティングはエンドポイントの変更に応じて、追加の再デプロイなしに動的に調整されます。
- Amazon EC2 起動タイプの場合、コンテナインスタンスは Amazon ECS 最適化 AMI バージョン `20260323` 以降を使用する必要があります。

Amazon ECS Service Connect トラフィックを暗号化する

Amazon ECS Service Connect は、Amazon ECS サービスの Transport Layer Security (TLS) 証明書による自動トラフィック暗号化をサポートしています。Amazon ECS サービスを AWS Private Certificate Authority(AWS Private CA) に向けると、Amazon ECS Service Connect 間のトラフィックを暗号化するための TLS 証明書が Amazon ECS によって自動的にプロビジョニングされます。Amazon ECS は、トラフィックの暗号化に使用される TLS 証明書を生成、ローテーション、および配布します。

Service Connect による自動トラフィック暗号化は、業界をリードする暗号化機能を使用してサービス間の通信を保護し、セキュリティ要件を満たすのに役立ちます。256-bit ECDSA および 2048-bit RSA 暗号化で AWS Private Certificate Authority TLS 証明書と暗号化をサポートしています。また、プライベート証明書と署名キーを完全に制御できるため、コンプライアンス要件を満たすのに役立ちます。デフォルトでは、TLS 1.3 はサポートされていますが、TLS 1.0 ~ 1.2 はサポートされていません。Service Connect は、次の暗号を使用して TLS 1.3 をサポートしています。

- TLS_AES_128_GCM_SHA256
- TLS_AES_256_GCM_SHA384
- TLS_CHACHA20_POLY1305_SHA256

> **Note**
>
> TLS 1.3 を使用するには、ターゲットのリスナーでも有効にする必要があります。Amazon ECS エージェントを通過するインバウンドトラフィックとアウトバウンドトラフィックのみが暗号化されます。

Service Connect と Application Load Balancer ヘルスチェック

Service Connect は、Application Load Balancer のヘルスチェックと TLS 1.3 暗号化を使って使用できます。

Application Load Balancer の設定

Application Load Balancer を次の設定に従って設定します。

- TLS 1.3 セキュリティポリシー (「ELBSecurityPolicy-TLS13-1-2-2021-06」など) を使用して TLS リスナーを設定します。詳細については、「Application Load Balancer のセキュリティポリシー」を参照してください。
- 次の設定でターゲットグループを作成します。
  - プロトコルを HTTPS に設定します。
  - ターゲットグループを TLS リスナーにアタッチします。
  - Service Connect サービスのコンテナポートと一致するようにヘルスチェックポートを設定します。

Service Connect の設定

以下の設定を使用してサービスを設定します。

- `bridge` ネットワークモードはサポートされていないため、`awsvpc` ネットワークモードを使用するようにサービスを設定します。
- サービスの Service Connect を有効にします。
- 次の設定を使用してロードバランサー設定をセットアップします。
  - Application Load Balancer 用に設定したターゲットグループを指定します。
  - Service Connect TLS サービスのコンテナポートと一致するようにコンテナポートを設定します。
- サービスに `ingressPortOverride` を設定しないでください。詳細については、「Amazon Elastic Container Service API リファレンス」の「ServiceConnectService」を参照してください。

考慮事項

Application Load Balancer、TLS、および Service Connect を使用する場合は、次の点を考慮してください。

- Service Connect を TLS 暗号化で使用するときは、HTTPS ヘルスチェックに `bridge` ネットワークモードの代わりに `awsvpc` ネットワークモードを使用します。HTTP ヘルスチェックは引き続き `bridge` モードで動作します。
- デフォルトの HTTPS ポート (443) ではなく、Service Connect サービスのコンテナポートと一致するようにターゲットグループのヘルスチェックポートを設定します。

AWS Private Certificate Authority 証明書および Service Connect

インフラストラクチャ IAM ロールが必要です。このロールの詳細については、「Amazon ECS インフラストラクチャの IAM ロール」を参照してください。

AWS Private Certificate AuthorityService Connect 用の モード

AWS Private Certificate Authority は汎用モードと短時間有効モードの 2 つのモードで実行できます。

- 汎用 — 任意の有効期限を設定できる証明書を発行します。
- 短い有効期間 — 最大有効期間が 7 日間の証明書。

Amazon ECS は両方のモードをサポートしていますが、有効期間の短い証明書を使用することをお勧めします。デフォルトでは、証明書は 5 日ごとにローテーションされ、短期モードで実行すると一般的な用途に比べてコストを大幅に節約できます。

Service Connect は証明書の失効をサポートしておらず、代わりに証明書を頻繁にローテーションする短い有効期間の証明書を利用します。Secrets Manager のマネージドローテーションを使用してローテーション頻度を変更したり、シークレットを無効化または削除したりする権限がありますが、これを行うと次のような結果が生じる可能性があります。

- ローテーション頻度の短縮 - ローテーション頻度を短くすると、AWS Private CA、AWS KMS および Secrets Manager、Auto Scaling のローテーションのワークロードが増加するため、コストが高くなります。
- ローテーション頻度が長い - ローテーション頻度が 7 日を超えると、アプリケーションの通信は失敗します。
- シークレットの削除 - シークレットを削除するとローテーションが失敗し、お客様のアプリケーション通信に影響があります。

シークレットローテーションに失敗すると、AWS CloudTrail で `RotationFailed` イベントがに公開されます。`RotationFailed` 用の CloudWatch アラームを設定することもできます。

> **Important**
>
> シークレットにレプリカリージョンを追加しないでください。これにより、Amazon ECS にはリージョンをレプリケーションから削除するアクセス許可がないため、Amazon ECS はシークレットを削除できなくなります。レプリケーションを既に追加している場合は、次のコマンドを実行します。
>
> ```
> aws secretsmanager remove-regions-from-replication \
>  --secret-id SecretId \
>  --remove-replica-regions region-name
> ```

下位認証機関

AWS Private CA、ルートまたは下位を Service Connect TLS に導入して、サービスのエンドエンティティ証明書を発行できます。提供された発行者は、あらゆる場所で署名者および信頼の根源として扱われます。エンドエンティティ証明書は、アプリケーションのさまざまな部分について、さ

まざまな下位 CA から発行できます。AWS CLI を使用する際に、信頼チェーンを確立する CA の Amazon リソースネーム (ARN) を指定します。

オンプレミス認証機関

オンプレミス CA を使用するには、AWS Private Certificate Authority で下位 CA を作成して設定します。これにより、Amazon ECS ワークロード用に発行されたすべての TLS 証明書が、オンプレミスで実行するワークロードとトラストチェーンを共有し、安全に接続できるようになります。

> **Important**
>
> AWS Private CA に必要なタグ `AmazonECSManaged : true` を自分に追加してください。

Infrastructure as Code

Infrastructure as Code (IaC) ツールで Service Connect TLS を使用する場合は、サービスがドレイン状態のままになるという問題を避けるため、依存関係を正しく設定することが重要です。AWS KMS キーが提供されている場合は、Amazon ECS サービスを終了した後、IAM ロールと AWS Private CA 依存関係を削除する必要があります。

Service Connect に使用される名前空間が共有名前空間である場合は、共有 AWS Private CA リソースを使用することを選択できます。詳細については、「AWS Private Certificate Authority ユーザーガイド」の「クロスアカウントアクセス許可をアタッチする」を参照してください。

Service Connect と Secrets Manager

Amazon ECS Service Connect を TLS 暗号化で使用する場合、サービスは次の方法で Secrets Manager とやり取りします。

Service Connect は、提供されたインフラストラクチャロールを使用して Secrets Manager 内にシークレットを作成します。これらのシークレットは、Service Connect サービス間のトラフィックを暗号化するための TLS 証明書の関連するプライベートキーを保存するために使用されます。

> **Warning**
>
> Service Connect によるこれらのシークレットの自動作成と管理により、サービスに TLS 暗号化を実装するプロセスが合理化されます。ただし、潜在的なセキュリティへの影響に注意することが重要です。Secrets Manager への読み取りアクセス権を持つ他の IAM ロールは、自動的に作成されたシークレットにアクセスできる場合があります。これにより、アクセス

コントロールが適切に設定されていない場合、機密性の高い暗号化マテリアルが権限のない当事者に公開される可能性があります。
このリスクを軽減するには、次のベストプラクティスに従ってください。

- 特に Service Connect によって作成されたシークレットについては、Secrets Manager へのアクセスを慎重に管理および制限します。
- IAM ロールとそのアクセス許可を定期的に監査して、最小特権の原則が維持されていることを確認します。

Secrets Manager に読み取りアクセスを許可する場合は、Service Connect によって作成された TLS プライベートキーを除外することを検討してください。これを行うには、IAM ポリシーの条件を使用して、パターンに一致する ARN を持つシークレットを除外します。

```
"arn:aws:secretsmanager:::secret:ecs-sc!"
```

`ecs-sc!` プレフィックスを持つすべてのシークレットに対する `GetSecretValue` アクションを拒否する IAM ポリシーの例:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": "arn:aws:secretsmanager:*:*:secret:ecs-sc!*"
        }
    ]
}
```

Note

これは一般的な例であり、特定のユースケースと AWS アカウント設定に基づいて調整する必要がある場合があります。IAM ポリシーは、セキュリティを維持しながら意図したアクセスを提供するように、常に徹底的なテストを行ってください。

Service Connect が Secrets Manager とやり取りする方法を理解することで、自動 TLS 暗号化の利点を活用しながら、Amazon ECS サービスのセキュリティをより適切に管理できます。

Service Connect および AWS Key Management Service

AWS Key Management Service を使用して、Service Connect リソースを暗号化および復号化できます。AWS KMS は、AWS によってデータを保護する暗号キーを作成および管理できるサービスです。

Service Connect で AWS KMS を使用する場合は、AWS が管理する AWS 所有キーを使用するか、既存の AWS KMS キーを選択できます。新しい AWS KMS キーを作成して使用することもできます。

独自の暗号化キーを提供する

独自のキーマテリアルを提供することも、AWS Key Management Service を通じて外部キーストアを使用して独自のキーを AWS KMS へインポートし、Amazon ECS Service Connect でそのキーの Amazon リソースネーム (ARN) を指定することもできます。

AWS KMS ポリシーの例を次に示します。すべての *[######]* を独自の値に置き換えます。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "id",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/role-name"
      },
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:GenerateDataKey",
        "kms:GenerateDataKeyPair"
      ],
      "Resource": "*"
    }
  ]
```

```
}
```

キーポリシーの詳細については、「AWS Key Management Service デベロッパーガイド」の「キーポリシーを作成する」を参照してください。

> **Note**
>
> Service Connect は対称暗号化 AWS KMS キーのみをサポートします。他のタイプの AWS KMS キーを使用して Service Connect リソースを暗号化することはできません。AWS KMS キーが対称暗号化キーかどうかを判別するには、「非対称 KMS キーを識別する」を参照してください。

AWS Key Management Service 対称暗号化キーの詳細については、「AWS Key Management Service デベロッパーガイド」の「対称暗号化 AWS KMS キー」を参照してください。

Amazon ECS Service Connect の TLS の有効化

Service Connect サービスを作成または更新するとき、トラフィック暗号化を有効にします。

AWS マネジメントコンソール を使用して既存のネームスペース内のサービスのトラフィック暗号化を有効にするには

1. インフラストラクチャ IAM ロールが必要です。このロールの詳細については、「Amazon ECS インフラストラクチャの IAM ロール」を参照してください。
2. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
3. ナビゲーションペインで [名前空間] を選択します。
4. トラフィックの暗号化を有効にする [サービス] のある [名前空間] を選択します。
5. トラフィック暗号化を有効にする [サービス] を選択します。
6. [サービスの更新] を選択し、[Service Connect] セクションまでスクロールします。
7. サービス情報の下にある [トラフィック暗号化を有効にする] を選択して TLS を有効にします。
8. [Service Connect TLS ロール] では、既存のインフラストラクチャ IAM ロールを選択するか、新しいロールを作成します。
9. [署名者認証機関] では、既存の認証機関を選択するか、新しい認証機関を作成します。

   詳細については、「AWS Private Certificate Authority 証明書および Service Connect」を参照してください。

10. AWS KMS key を選択 では、AWS を所有済みのマネージドキーを選択するか、別のキーを選択できます。新しいものを作成することもできます。

AWS CLI を使用してサービスの TLS を設定する例については、「AWS CLI を使用した Amazon ECS Service Connect の設定」を参照してください。

Amazon ECS Service Connect で TLS が有効になっていることを確認する

Service Connect は、Service Connect エージェントで TLS を開始し、宛先エージェントで TLS を終了します。その結果、アプリケーションコードが TLS のやりとりを認識することはありません。次の手順を使用して、TLS が有効になっていることを確認します。

1. アプリケーションイメージに openssl CLI を含めます。
2. SSM 経由でタスクに接続するには、サービスで ECS Exec を有効にします。代わりに、サービスと同じ Amazon VPC で Amazon EC2 インスタンスを起動することもできます。
3. 検証したいサービスからタスクの IP とポートを取得します。タスクの IP アドレスは AWS Cloud Map コンソールで取得できます。この情報は、名前空間のサービス詳細ページに記載されています。
4. 次の例のように execute-command を使用して、タスクのいずれかに口グオンします。または、ステップ 2 で作成した Amazon EC2 インスタンスに口グオンします。

```
$ aws ecs execute-command --cluster cluster-name \
    --task task-id  \
    --container container-name \
    --interactive \
    --command "/bin/sh"
```

> **Note**
>
> DNS 名を直接呼び出しても証明書は公開されません。

5. 接続したシェルで、openssl CLI を使用してタスクに添付されている証明書を確認および表示します。

   例:

```
openssl s_client -connect 10.0.147.43:6379 < /dev/null 2> /dev/null \
| openssl x509 -noout -text
```

レスポンスの例:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            <serial-number>
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: <issuer>
        Validity
            Not Before: Jan 23 21:38:12 2024 GMT
            Not After : Jan 30 22:38:12 2024 GMT
        Subject: <subject>
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    <pub>
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Subject Alternative Name:
                DNS:redis.yelb-cftc
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Authority Key Identifier:
                keyid:<key-id>

            X509v3 Subject Key Identifier:
                1D:<id>
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
    Signature Algorithm: ecdsa-with-SHA256
        <hash>
```

AWS CLI を使用した Amazon ECS Service Connect の設定

AWS CLI で Service Connect を使用する Fargate タスクで Amazon ECS サービスを作成できます。

> **Note**
>
> デュアルスタックサービスエンドポイントを使用することで、IPv4 と IPv6 の両方を介して AWS CLI、SDK、および Amazon ECS API から Amazon ECS とやり取りできます。詳細については、「Amazon ECS デュアルスタックエンドポイントの使用」を参照してください。

前提条件

Service Connect の前提条件は次のとおりです。

- AWS CLI の最新バージョンがインストールされ、設定されていることを確認します。詳細については、「AWS CLIの最新バージョンのインストールまたは更新」を参照してください。
- IAM ユーザーに AmazonECS_FullAccess IAM ポリシー例で指定されている必要なアクセス権限があること。
- VPC、サブネット、ルートテーブルおよびセキュリティグループが使用できるように作成されています。詳細については、「the section called “仮想プライベートクラウドを作成する”」を参照してください。
- `ecsTaskExecutionRole` という名前のタスク実行ロールがあり、`AmazonECSTaskExecutionRolePolicy` 管理ポリシーがそのロールにアタッチされています。このロールにより、Fargate は NGINX アプリケーションログと Service Connect プロキシログを Amazon CloudWatch Logs に書き込むことができます。詳細については、「タスク実行 ロールの作成」を参照してください。

ステップ 1: クラスターを作成する

次の手順に従って Amazon ECS クラスターと名前空間を作成します。

Amazon ECS クラスターと AWS Cloud Map 名前空間を作成するには

1. `tutorial` という名前の Amazon ECS クラスターを作成して使用します。パラメータ `--service-connect-defaults` は、クラスターのデフォルト名前空間を設定します。この出力の例では、`service-connect` という名前の AWS Cloud Map 名前空間はこのアカウントおよび AWS リージョンには存在しないため、名前空間は Amazon ECS によって作成されています。名前空間はアカウント内の AWS Cloud Map で作成され、他のすべての名前空間でも表示されるため、目的を示す名前を使用します。

```
aws ecs create-cluster --cluster-name tutorial --service-connect-defaults
 namespace=service-connect
```

出力:

```
{
    "cluster": {
        "clusterArn": "arn:aws:ecs:us-west-2:123456789012:cluster/tutorial",
        "clusterName": "tutorial",
        "serviceConnectDefaults": {
            "namespace": "arn:aws:servicediscovery:us-
west-2:123456789012:namespace/ns-EXAMPLE"
        },
        "status": "PROVISIONING",
        "registeredContainerInstancesCount": 0,
        "runningTasksCount": 0,
        "pendingTasksCount": 0,
        "activeServicesCount": 0,
        "statistics": [],
        "tags": [],
        "settings": [
            {
                "name": "containerInsights",
                "value": "disabled"
            }
        ],
        "capacityProviders": [],
        "defaultCapacityProviderStrategy": [],
        "attachments": [
            {
                "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
                "type": "sc",
                "status": "ATTACHING",
                "details": []
            }
        ],
        "attachmentsStatus": "UPDATE_IN_PROGRESS"
    }
}
}
```

2. クラスターが作成されていることを確認します。

```
aws ecs describe-clusters --clusters tutorial
```

出力:

```
{
    "clusters": [
        {
            "clusterArn": "arn:aws:ecs:us-west-2:123456789012:cluster/tutorial",
            "clusterName": "tutorial",
            "serviceConnectDefaults": {
                "namespace": "arn:aws:servicediscovery:us-
west-2:123456789012:namespace/ns-EXAMPLE"
            },
            "status": "ACTIVE",
            "registeredContainerInstancesCount": 0,
            "runningTasksCount": 0,
            "pendingTasksCount": 0,
            "activeServicesCount": 0,
            "statistics": [],
            "tags": [],
            "settings": [],
            "capacityProviders": [],
            "defaultCapacityProviderStrategy": []
        }
    ],
    "failures": []
}
```

3. (オプション) 名前空間が AWS Cloud Map で作成されていることを確認します。これは AWS Cloud Map で作成されているため、AWS マネジメントコンソールまたは通常の AWS CLI 設定を使用できます。

   例えば、以下の AWS CLI を使用します。

   ```
   aws servicediscovery get-namespace --id ns-EXAMPLE
   ```

   出力:

   ```
   {
       "Namespace": {
   ```

```
        "Id": "ns-EXAMPLE",
        "Arn": "arn:aws:servicediscovery:us-west-2:123456789012:namespace/ns-
EXAMPLE",
        "Name": "service-connect",
        "Type": "HTTP",
        "Properties": {
            "DnsProperties": {
                "SOA": {}
            },
            "HttpProperties": {
                "HttpName": "service-connect"
            }
        },
        "CreateDate": 1661749852.422,
        "CreatorRequestId": "service-connect"
    }
}
```

ステップ 2: サーバー用のサービスを作成する

Service Connect 機能は、Amazon ECS 上の複数のアプリケーションを相互接続することを目的としています。これらのアプリケーションの少なくとも 1 つは、接続するウェブサービスを提供する必要があります。このステップでは、以下を作成します。

- 未修正の公式 NGINX コンテナイメージを使用し、Service Connect 設定を含むタスク定義。
- このサービスへのトラフィック用にサービス検出とサービスメッシュプロキシを提供するために Service Connect を設定する Amazon ECS サービス定義。この設定では、クラスター設定からデフォルトの名前空間を再利用することで、各サービスに対して行うサービス設定の量を減らします。
- Amazon ECS サービス。タスク定義を使用して 1 つのタスクを実行し、Service Connect プロキシ用の追加コンテナを挿入します。プロキシは、タスク定義のコンテナポートマッピングからポートをリッスンします。Amazon ECS で実行されているクライアントアプリケーションで、クライアントタスクのプロキシは、タスク定義ポート名、サービス検出名またはサービスクライアントエイリアス名、およびクライアントエイリアスからのポート番号へのアウトバウンド接続をリッスンします。

Service Connect を使用してウェブサービスを作成するには

1. Fargate と互換性があり、awsvpc ネットワークモードを使用するタスク定義を登録します。以下の手順に従ってください。

   a. 次のタスク定義の内容で、service-connect-nginx.json というファイルを作成します。

   このタスク定義は、ポートマッピングに name および appProtocol パラメータを追加することによって Service Connect を構成します。複数のポートが使用されている場合、ポート名によりサービス設定でこのポートをより識別しやすくなります。また、ポート名は、名前空間内の他のアプリケーションが使用する検出可能な名前としてもデフォルトで使用されています。

   サービスでは ECS Exec が有効になっているため、タスク定義にはタスク IAM ロールが含まれています。

   > **Important**
   >
   > このタスク定義では、logConfiguration を使用して stdout および stderr から Amazon CloudWatch Logs に nginx 出力を送信します。このタスク実行ロールには、CloudWatch Logs ロググループを作成するために必要な追加の権限はありません。AWS マネジメントコンソールまたは AWS CLI を使用して、CloudWatch Logs にロググループを作成します。nginx ログを CloudWatch Logs に送信したくない場合、logConfiguration を削除できます。
   > タスク実行ロールの AWS アカウント ID を AWS アカウント ID に置き換えます。

```
{
    "family": "service-connect-nginx",
    "executionRoleArn": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
    "taskRoleArn": "arn:aws:iam::123456789012:role/ecsTaskRole",
    "networkMode": "awsvpc",
    "containerDefinitions": [
        {
        "name": "webserver",
        "image": "public.ecr.aws/docker/library/nginx:latest",
        "cpu": 100,
        "portMappings": [
```

```
            {
                "name": "nginx",
                "containerPort": 80,
                "protocol": "tcp",
                "appProtocol": "http"
            }
        ],
        "essential": true,
        "logConfiguration": {
            "logDriver": "awslogs",
            "options": {
                "awslogs-group": "/ecs/service-connect-nginx",
                "awslogs-region": "region",
                "awslogs-stream-prefix": "nginx"
            }
        }
        }
    ],
    "cpu": "256",
    "memory": "512"
}
```

b. 次の service-connect-nginx.json ファイルを使用して、タスク定義を登録します。

```
aws ecs register-task-definition --cli-input-json file://service-connect-
nginx.json
```

2. サービスを作成します。

a. 作成する Amazon ECS サービスの内容で、service-connect-nginx-service.json という名前のファイルを作成します。この例では、前のステップで作成したタスク定義を使用します。このタスク定義の例では awsvpcConfiguration ネットワークモードを使用しているため、awsvpc が必要となります。

ECS サービスを作成する際に、Fargate と、Service Connect をサポートする LATEST プラットフォームバージョンを指定します。securityGroups および subnets は、Amazon ECS を使用するための要件を満たしている VPC に属している必要があります。Amazon VPC コンソールからセキュリティグループとサブネット ID を取得できます。

このサービスは、serviceConnectConfiguration パラメータを追加して Service Connect を設定します。クラスターにはデフォルトの名前空間が設定されているた

め、名前空間は必要ありません。名前空間内の ECS で実行されているクライアントアプリケーションは、portName および clientAliases のポートを使用してこのサービスに接続します。例えば、nginx はルートロケーション / にウェルカムページを提供しているため、http://nginx:80/ を使用してこのサービスにアクセスできます。Amazon ECS で実行されていない、または同じ名前空間にない外部アプリケーションは、タスクの IP アドレスとタスク定義のポート番号を使用して、Service Connect プロキシ経由でこのアプリケーションにアクセスできます。ご使用の tls 設定に合わせて、awsPcaAuthorityArn、kmsKey および IAM ロールの roleArn に対する証明書 arn を追加します。

このサービスは、logConfiguration を使用して stdout および stderr から Amazon CloudWatch Logs にサービス接続プロキシの出力を送信します。このタスク実行ロールには、CloudWatch Logs ロググループを作成するために必要な追加の権限はありません。AWS マネジメントコンソールまたは AWS CLI を使用して、CloudWatch Logs にロググループを作成します。このロググループを作成し、CloudWatch Logs にプロキシログを保存することをお勧めします。プロキシログを CloudWatch Logs に送信したくない場合、logConfiguration を削除できます。

```
{
    "cluster": "tutorial",
    "deploymentConfiguration": {
        "maximumPercent": 200,
        "minimumHealthyPercent": 0
    },
    "deploymentController": {
        "type": "ECS"
    },
    "desiredCount": 1,
    "enableECSManagedTags": true,
    "enableExecuteCommand": true,
    "launchType": "FARGATE",
    "networkConfiguration": {
        "awsvpcConfiguration": {
            "assignPublicIp": "ENABLED",
            "securityGroups": [
                "sg-EXAMPLE"
            ],
            "subnets": [
                "subnet-EXAMPLE",
                "subnet-EXAMPLE",
```

```
                "subnet-EXAMPLE"
            ]
           }
    },
    "platformVersion": "LATEST",
    "propagateTags": "SERVICE",
    "serviceName": "service-connect-nginx-service",
    "serviceConnectConfiguration": {
        "enabled": true,
        "services": [
            {
                "portName": "nginx",
                "clientAliases": [
                    {
                        "port": 80
                    }
                ],
                "tls": {
                   "issuerCertificateAuthority": {
                      "awsPcaAuthorityArn": "certificateArn"
                   },
                   "kmsKey": "kmsKey",
                   "roleArn": "iamRoleArn"
                }
            }
        ],
        "logConfiguration": {
            "logDriver": "awslogs",
            "options": {
                "awslogs-group": "/ecs/service-connect-proxy",
                "awslogs-region": "region",
                "awslogs-stream-prefix": "service-connect-proxy"
            }
        }
    },
    "taskDefinition": "service-connect-nginx"
}
```

b. 次の service-connect-nginx-service.json のファイルを使用して、サービスを作成します。

```
aws ecs create-service --cluster tutorial --cli-input-json file://service-
connect-nginx-service.json
```

出力:

```
{
    "service": {
        "serviceArn": "arn:aws:ecs:us-west-2:123456789012:service/tutorial/
service-connect-nginx-service",
        "serviceName": "service-connect-nginx-service",
        "clusterArn": "arn:aws:ecs:us-west-2:123456789012:cluster/tutorial",
        "loadBalancers": [],
        "serviceRegistries": [],
        "status": "ACTIVE",
        "desiredCount": 1,
        "runningCount": 0,
        "pendingCount": 0,
        "launchType": "FARGATE",
        "platformVersion": "LATEST",
        "platformFamily": "Linux",
        "taskDefinition": "arn:aws:ecs:us-west-2:123456789012:task-definition/
service-connect-nginx:1",
        "deploymentConfiguration": {
            "deploymentCircuitBreaker": {
                "enable": false,
                "rollback": false
            },
            "maximumPercent": 200,
            "minimumHealthyPercent": 0
        },
        "deployments": [
            {
                "id": "ecs-svc/3763308422771520962",
                "status": "PRIMARY",
                "taskDefinition": "arn:aws:ecs:us-west-2:123456789012:task-
definition/service-connect-nginx:1",
                "desiredCount": 1,
                "pendingCount": 0,
                "runningCount": 0,
                "failedTasks": 0,
                "createdAt": 1661210032.602,
```

```
                "updatedAt": 1661210032.602,
                "launchType": "FARGATE",
                "platformVersion": "1.4.0",
                "platformFamily": "Linux",
                "networkConfiguration": {
                    "awsvpcConfiguration": {
                        "assignPublicIp": "ENABLED",
                        "securityGroups": [
                            "sg-EXAMPLE"
                        ],
                        "subnets": [
                            "subnet-EXAMPLEf",
                            "subnet-EXAMPLE",
                            "subnet-EXAMPLE"
                        ]
                    }
                },
                "rolloutState": "IN_PROGRESS",
                "rolloutStateReason": "ECS deployment ecs-
svc/3763308422771520962 in progress.",
                "failedLaunchTaskCount": 0,
                "replacedTaskCount": 0,
                "serviceConnectConfiguration": {
                    "enabled": true,
                    "namespace": "service-connect",
                    "services": [
                        {
                            "portName": "nginx",
                            "clientAliases": [
                                {
                                    "port": 80
                                }
                            ]
                        }
                    ],
                    "logConfiguration": {
                        "logDriver": "awslogs",
                        "options": {
                            "awslogs-group": "/ecs/service-connect-proxy",
                            "awslogs-region": "us-west-2",
                            "awslogs-stream-prefix": "service-connect-proxy"
                        },
                        "secretOptions": []
                    }
```

```
                },
                "serviceConnectResources": [
                    {
                        "discoveryName": "nginx",
                        "discoveryArn": "arn:aws:servicediscovery:us-
west-2:123456789012:service/srv-EXAMPLE"
                    }
                ]
            }
        ],
        "roleArn": "arn:aws:iam::123456789012:role/aws-service-role/
ecs.amazonaws.com/AWSServiceRoleForECS",
        "version": 0,
        "events": [],
        "createdAt": 1661210032.602,
        "placementConstraints": [],
        "placementStrategy": [],
        "networkConfiguration": {
            "awsvpcConfiguration": {
                "assignPublicIp": "ENABLED",
                "securityGroups": [
                    "sg-EXAMPLE"
                ],
                "subnets": [
                    "subnet-EXAMPLE",
                    "subnet-EXAMPLE",
                    "subnet-EXAMPLE"
                ]
            }
        },
        "schedulingStrategy": "REPLICA",
        "enableECSManagedTags": true,
        "propagateTags": "SERVICE",
        "enableExecuteCommand": true
    }
}
```

指定した serviceConnectConfiguration は、出力の最初のデプロイ内に表示されます。タスクに変更を加える必要がある方法で ECS サービスを変更すると、Amazon ECS によって新しいデプロイが作成されます。

ステップ 3: 接続できることを確認する

Service Connect が設定され動作していることを確認するには、次の手順に従って外部アプリケーションからウェブサービスに接続します。次に、Service Connect プロキシが作成した CloudWatch にある追加のメトリクスを確認します。

外部アプリケーションからウェブサービスに接続するには

- タスク IP アドレスを使用して、タスク IP アドレスとコンテナポートに接続する

  AWS CLI を使用して `aws ecs list-tasks --cluster tutorial` を使用したタスク ID を取得します。

  サブネットとセキュリティグループがタスク定義のポート上のパブリックインターネットからのトラフィックを許可している場合、コンピュータからパブリック IP に接続できます。ただし、パブリック IP は「describe-tasks」からは利用できないため、手順として、Amazon EC2 AWS マネジメントコンソールまたは AWS CLI に移動して Elastic Network Interface の詳細を取得する必要があります。

  この例では、同じ VPC 内の Amazon EC2 インスタンスはタスクのプライベート IP を使用します。アプリケーションは nginx ですが、`server: envoy` ヘッダーには Service Connect プロキシが使用されていることが表示されます。Service Connect プロキシはタスク定義のコンテナポートをリッスンしています。

```
$ curl -v 10.0.19.50:80/
*   Trying 10.0.19.50:80...
* Connected to 10.0.19.50 (10.0.19.50) port 80 (#0)
> GET / HTTP/1.1
> Host: 10.0.19.50
> User-Agent: curl/7.79.1
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< server: envoy
< date: Tue, 23 Aug 2022 03:53:06 GMT
< content-type: text/html
< content-length: 612
< last-modified: Tue, 16 Apr 2019 13:08:19 GMT
< etag: "5cb5d3c3-264"
< accept-ranges: bytes
```

```
< x-envoy-upstream-service-time: 0
<
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Service Connect メトリクスを表示するには

Service Connect プロキシは、アプリケーション (HTTP、HTTP2、gRPC、または TCP 接続) メトリクスを CloudWatch メトリクス内に作成します。CloudWatch コンソールを使用する場合、Amazon ECS 名前空間にある [DiscoveryName]、([DiscoveryName, ServiceName, ClusterName])、[TargetDiscoveryName]、および ([TargetDiscoveryName, ServiceName, ClusterName]) という追加のメトリクスディメンションを確認します。これらのメトリクスおよびディメンションの詳細については、「Amazon CloudWatch Logs ユーザーガイド」の「利用可能なメトリクスを表示する」を参照してください。

サービス検出を使用して Amazon ECS サービスを DNS 名で接続する

Amazon ECS サービスはオプションで Amazon ECS サービス検出を使用するように設定できます。サービス検出では、AWS Cloud MapAmazon ECS サービスの HTTP および DNS 名前空間を管理する API アクション。詳細については、『AWS Cloud Map 開発者ガイド』の「AWS Cloud Map とは」を参照してください。

サービスの検出はリ以下のAWSージョンで使用できます。

リージョン名	リージョン
米国東部 (バージニア北部)	us-east-1
米国東部 (オハイオ)	us-east-2
米国西部 (北カリフォルニア)	us-west-1
米国西部 (オレゴン)	us-west-2
アフリカ (ケープタウン)	af-south-1
アジアパシフィック (香港)	ap-east-1
アジアパシフィック (台北)	ap-east-2
アジアパシフィック (ムンバイ)	ap-south-1
アジアパシフィック (ハイデラバード)	ap-south-2
アジアパシフィック (東京)	ap-northeast-1
アジアパシフィック (ソウル)	ap-northeast-2
アジアパシフィック (大阪)	ap-northeast-3
アジアパシフィック (シンガポール)	ap-southeast-1
アジアパシフィック (シドニー)	ap-southeast-2
アジアパシフィック (ジャカルタ)	ap-southeast-3
アジアパシフィック (メルボルン)	ap-southeast-4

リージョン名	リージョン
アジアパシフィック (マレーシア)	ap-southeast-5
アジアパシフィック (ニュージーランド)	ap-southeast-6
アジアパシフィック (タイ)	ap-southeast-7
カナダ (中部)	ca-central-1
カナダ西部 (カルガリー)	ca-west-1
中国 (北京)	cn-north-1
中国 (寧夏)	cn-northwest-1
欧州 (フランクフルト)	eu-central-1
欧州 (チューリッヒ)	eu-central-2
欧州 (アイルランド)	eu-west-1
欧州 (ロンドン)	eu-west-2
欧州 (パリ)	eu-west-3
欧州 (ミラノ)	eu-south-1
欧州 (ストックホルム)	eu-north-1
イスラエル (テルアビブ)	il-central-1
欧州 (スペイン)	eu-south-2
中東 (アラブ首長国連邦)	me-central-1
メキシコ (中部)	mx-central-1
中東 (バーレーン)	me-south-1
南米 (サンパウロ)	sa-east-1

リージョン名	リージョン
AWS GovCloud (米国東部)	us-gov-east-1
AWS GovCloud (米国西部)	us-gov-west-1

サービス検出の概念

サービス検出は次のコンポーネントで構成されます。

- サービス検出名前空間: 同じドメイン名 (`example.com` など) を共有するサービス検出サービスの論理グループです。ここにトラフィックをルーティングします。`aws servicediscovery create-private-dns-namespace` コマンドを呼び出しまたは Amazon ECS のコンソールを使用して名前空間を作成できます。`aws servicediscovery list-namespaces` コマンドを使用して、現在のアカウントで作成された名前空間に関するサマリー情報を確認できます。サービス検出コマンドの詳細については、「AWS Cloud Map (サービス検出) AWS CLI Reference Guide」の「`create-private-dns-namespace`」および「`list-namespaces`」を参照してください。
- サービス検出サービス: サービス検出名前空間にあり、名前空間のサービス名および DNS 設定から設定されます。これは、次の主要なコンポーネントを提供します。
  - サービスレジストリ: DNS あるいは AWS Cloud Map API アクションを介してサービスを検索し、サービスに接続するために使用できる 1 つ以上の利用可能なエンドポイントを返すことができます。
- サービスディスカバリインスタンス: サービスディスカバリサービスにあり、サービスディレクトリ内の各 Amazon ECS サービスに関連付けられた属性で構成されます。
  - インスタンスの属性: 次のメタデータは、サービスディスカバリ を使用するように設定された各 Amazon ECS サービスのカスタム属性として追加されます。
    - **`AWS_INSTANCE_IPV4`** –A レコードの場合、インスタンスの詳細が検出されると、Route 53 など、DNS クエリへの応答として AWS Cloud Map が返す IPv4 アドレスおよび `192.0.2.44` が返されます。
    - **`AWS_INSTANCE_IPV6`** –AAAA レコードの場合は、インスタンスの詳細が検出されると、Route 53 など、DNS クエリへの応答として AWS Cloud Map が返す IPv6 アドレスおよび `2001:0db8:85a3:0000:0000:abcd:0001:2345` が返されます。`AWS_INSTANCE_IPv4` と `AWS_INSTANCE_IPv6` の両方が Amazon ECS デュアルスタックサービスに追加されます。Amazon ECS IPv6 専用サービスにのみ `AWS_INSTANCE_IPv6` が追加されます。

  - **AWS_INSTANCE_PORT** – サービスディスカバリサービスに関連付けられたポート値。
  - **AVAILABILITY_ZONE** – タスクが起動したアベイラビリティーゾーン。EC2 を使用するタスクの場合、これはコンテナインスタンスが存在するアベイラビリティーゾーンです。Fargate を使用するタスクの場合、これは Elastic Network Interface が存在するアベイラビリティーゾーンです。
  - **REGION** タスクが存在するリージョン。
  - **ECS_SERVICE_NAME** – タスクが属している Amazon ECS サービスの名前。
  - **ECS_CLUSTER_NAME** – タスクが属している Amazon ECS クラスターの名前。
  - **EC2_INSTANCE_ID** タスクが配置されていたコンテナインスタンスの ID。タスクが Fargate を使用している場合、このカスタム属性は追加されません。
  - **ECS_TASK_DEFINITION_FAMILY** タスクが使用しているタスク定義ファミリー。
  - **ECS_TASK_SET_EXTERNAL_ID** タスクセットが外部デプロイ用に作成され、サービス検出レジストリに関連付けられている場合、ECS_TASK_SET_EXTERNAL_ID 属性にはタスクセットの外部 ID が含まれます。
- Amazon ECS ヘルスチェック: Amazon ECS はコンテナレベルのヘルスチェックを定期的に実行します。エンドポイントがヘルスチェックに失敗した場合、このエンドポイントは DNS ルーチングから削除され、異常とマークされます。

サービスの検出に関する考慮事項

サービス検出を使用する際には、以下の点を考慮する必要があります。

- プラットフォームバージョンが v1.1.0 以降を使用する場合、サービスの検出は Fargate タスクでサポートされます。詳細については、「Amazon ECS 向け Fargate プラットフォームバージョン」を参照してください。
- サービス検出を使用するように構成されたサービスには、サービスごとに 1,000 タスクに制限があります。これは、Route 53 サービスクォータによるものです。
- Amazon ECS コンソールでのサービスの作成ワークフローでは、プライベート DNS 名前空間へのサービスの登録のみがサポートされます。AWS Cloud Map プライベート DNS 名前空間が作成されると、Route 53 プライベートホストゾーンが自動的に作成されます。
- DNS 解決を成功させるには、VPC DNS 属性を設定する必要があります。属性の設定方法については、を参照してください。VPC の DNS サポートのAmazon VPC User Guide。
- Amazon ECS は、共有 AWS Cloud Map 名前空間へのサービスの登録をサポートしていません。

- パブリック名前空間が使用されている場合でも、 サービス用に作成された DNS レコードは、パブリック IP アドレスではなく、タスクのプライベート IP アドレスに常に登録されます。
- &service-discovery-first; では、awsvpc、bridge、host のいずれかのネットワークモードをタスクで指定する必要があります (none はサポートされていません)。
- サービスタスク定義が awsvpc ネットワークモードを使用する場合、各サービスタスクに A または SRV のレコードを自由に組み合わせて作成できます。SRV レコードを使用する場合は、ポートが必要です。サービスがデュアルスタックサブネットを使用している場合は、さらに AAAA レコードを作成できます。サービスが IPv6 専用サブネットを使用している場合、A レコードを作成することはできません。
- サービスタスク定義が bridge または host ネットワークモードを使用する場合、SRV レコードのみがサポートされる DNS レコードタイプです。各サービスタスクの SRV レコードを作成します。SRV レコードのコンテナ名とコンテナポートの組み合わせをタスク定義から指定する必要があります。
- サービスの検出サービスの DNS レコードは、VPC 内でクエリを実行できます。これは、次の形式を使用します: <service-discovery-service-name>.<service-discovery-namespace>。
- サービス名で DNS クエリを実行すると、A と AAAA のレコードはタスクに対応する IP アドレスのセットを返します。SRV は、各タスクの IP アドレスとポートのセットを返します。
- 8 つ以下の正常なレコードがある場合、Route 53 はすべての DNS クエリに正常なすべてのレコードを返します。
- すべてのレコードが異常である場合、Route 53 は DNS クエリに最大 8 つの異常なレコードを返します。
- サービス検出はロードバランサーの背後にあるサービスに設定できますが、サービス検出トラフィックは必ずタスクにルーティングされ、ロードバランサーにはルーティングされません。
- サービス検出は Classic Load Balancer の使用をサポートしていません。
- Amazon ECS サービスのサービス検出により管理されるコンテナレベルのヘルスチェックを使用することをお勧めします。
  - HealthCheckCustomConfig—Amazon ECS; はユーザーに代わってヘルスチェックを管理します。Amazon ECS は、コンテナとヘルスチェックの情報、およびタスクの状態を使用して、ヘルスを AWS Cloud Map で更新します。これは、--health-check-custom-configパラメータを使用してサービス検出サービスの作成時に指定します。詳細については、AWS Cloud MapAPI リファレンスの「HealthCheckCustomConfig」を参照してください。
- サービス検出を使用するときに作成される AWS Cloud Map リソースは、手動でクリーンアップする必要があります。

- タスクとインスタンスはコンテナのヘルスチェックが値を返すまで UNHEALTHY として登録されます。ヘルスチェックが成功した場合、ステータスは HEALTHY に更新されます。コンテナのヘルスチェックに失敗すると、サービス検出インスタンスは登録解除されます。

サービス検出の料金

Amazon ECS サービスディスカバリを使用しているお客様には、Route 53 リソースおよび AWS Cloud Map 検出 API オペレーションの料金が発生します。これには、Route 53 ホストゾーンの作成とサービスレジストリへのクエリのコストが含まれます。詳細については、AWS Cloud Mapデベロッパーガイドのの概念およびAWS Cloud Mapの料金を参照してください。

Amazon ECS は、コンテナレベルのヘルスチェックを実行し、この結果を AWS Cloud Map カスタムヘルスチェック API オペレーションに公開します。現在のところ、これは追加コストなしでお客様に提供されています。パブリックに公開されているタスクにネットワークヘルスチェックを設定する場合、このヘルスチェックに対しては課金されます。

サービス検出を使用する新しい Amazon ECS サービスの作成

AWS CLI でサービス検出を使用する Fargate タスクを含むサービスを作成する方法について説明します。

サービス検出をサポートする AWS リージョン のリストについては、「サービス検出を使用して Amazon ECS サービスを DNS 名で接続する」を参照してください。

Fargate をサポートするリージョンの情報については、「the section called “AWS Fargate リージョン”」 を参照してください。

> **Note**
>
> デュアルスタックサービスエンドポイントを使用することで、IPv4 と IPv6 の両方を介して AWS CLI、SDK、および Amazon ECS API から Amazon ECS とやり取りできます。詳細については、「Amazon ECS デュアルスタックエンドポイントの使用」を参照してください。

前提条件

個のチュートリアルを開始する前に、次の前提条件を満たしていることを確認します。

- AWS CLI の最新バージョンがインストールされ、設定されていること。詳細については、「AWS CLIの最新バージョンのインストールまたは更新」を参照してください。

- 「Amazon ECS を使用するようにセットアップする」で説明されているステップが完了しました。
- IAM ユーザーに AmazonECS_FullAccess IAM ポリシー例で指定されている必要なアクセス権限があること。
- 少なくとも 1 つの VPC と 1 つのセキュリティグループを作成している。詳細については、「the section called “仮想プライベートクラウドを作成する”」を参照してください。

ステップ 1: AWS Cloud Map でサービス検出リソースを作成する

サービス検出の名前空間およびサービス検出サービスを作成するには、次のステップに従います。

1. プライベート Cloud Map サービス検出の名前空間を作成します。この例では、tutorial と呼ばれる名前空間を作成します。*vpc-abcd1234* を、既存のいずれかの VPC の ID に置き換えます。

```
aws servicediscovery create-private-dns-namespace \
      --name tutorial \
      --vpc vpc-abcd1234
```

このコマンドの出力は次のとおりです。

```
{
    "OperationId": "h2qe3s6dxftvvt7riu6lfy2f6c3jlhf4-je6chs2e"
}
```

2. 前のステップの出力の OperationId を使用して、プライベート名前空間が正常に作成されたことを確認します。名前空間 ID は後続のコマンドで使用するため、書き留めておきます。

```
aws servicediscovery get-operation \
      --operation-id h2qe3s6dxftvvt7riu6lfy2f6c3jlhf4-je6chs2e
```

出力は次のとおりです。

```
{
    "Operation": {
        "Id": "h2qe3s6dxftvvt7riu6lfy2f6c3jlhf4-je6chs2e",
        "Type": "CREATE_NAMESPACE",
        "Status": "SUCCESS",
        "CreateDate": 1519777852.502,
```

```
        "UpdateDate": 1519777856.086,
        "Targets": {
           "NAMESPACE": "ns-uejictsjen2i4eeg"
        }
    }
}
```

3. 前のステップの出力からの NAMESPACE ID を使用して、サービス検出サービスを作成します。この例では、myapplication という名前のサービスが作成されます。サービス ID と ARN は後続のコマンドで使用するため、書き留めておきます。

```
aws servicediscovery create-service \
      --name myapplication \
      --dns-config "NamespaceId="ns-
uejictsjen2i4eeg",DnsRecords=[{Type="A",TTL="300"}]" \
      --health-check-custom-config FailureThreshold=1
```

出力は次のとおりです。

```
{
    "Service": {
       "Id": "srv-utcrh6wavdkggqtk",
        "Arn": "arn:aws:servicediscovery:region:aws_account_id:service/srv-
utcrh6wavdkggqtk",
        "Name": "myapplication",
        "DnsConfig": {
            "NamespaceId": "ns-uejictsjen2i4eeg",
            "DnsRecords": [
                {
                    "Type": "A",
                    "TTL": 300
                }
            ]
        },
        "HealthCheckCustomConfig": {
            "FailureThreshold": 1
        },
        "CreatorRequestId": "e49a8797-b735-481b-a657-b74d1d6734eb"
    }
}
```

ステップ 2: Amazon ECS リソースを作成する

Amazon ECS クラスター、タスク定義、サービスを作成するには、次のステップに従います。

1. Amazon ECS クラスターを作成します。この例では、tutorial という名前のクラスターを作成します。

```
aws ecs create-cluster \
      --cluster-name tutorial
```

2. Fargate と互換性があり、awsvpc ネットワークモードを使用するタスク定義を登録します。以下の手順に従ってください。

   a. 次のタスク定義の内容で、fargate-task.json というファイルを作成します。

```
{
    "family": "tutorial-task-def",
        "networkMode": "awsvpc",
        "containerDefinitions": [
            {
                "name": "sample-app",
                "image": "public.ecr.aws/docker/library/httpd:2.4",
                "portMappings": [
                    {
                        "containerPort": 80,
                        "hostPort": 80,
                        "protocol": "tcp"
                    }
                ],
                "essential": true,
                "entryPoint": [
                    "sh",
                    "-c"
                ],
                "command": [
                    "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample
 App</title> <style>body {margin-top: 40px; background-color: #333;} </style>
 </head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample
 App</h1> <h2>Congratulations!</h2> <p>Your application is now running on a
 container in Amazon ECS.</p> </div></body></html>' >  /usr/local/apache2/
htdocs/index.html && httpd-foreground\""
                ]
            }
```

```
        ],
        "requiresCompatibilities": [
            "FARGATE"
        ],
        "cpu": "256",
        "memory": "512"
}
```

b. fargate-task.json を使用してタスク定義を登録します。

```
aws ecs register-task-definition \
      --cli-input-json file://fargate-task.json
```

3. 次のステップに従って、ECS サービスを作成します。

a. 作成する ECS サービスの内容で、ecs-service-discovery.json という名前のファイルを作成します。この例では、前のステップで作成したタスク定義を使用します。このタスク定義の例では awsvpcConfiguration ネットワークモードを使用しているため、awsvpc が必要となります。

ECS サービスを作成する際に、Fargate と、サービス検出をサポートする LATEST プラットフォームバージョンを指定します。AWS Cloud Map でサービス検出サービスが作成される場合、registryArn は返される ARN です。securityGroups および subnets は、Cloud Map 名前空間の作成に使用される VPC に属している必要があります。Amazon VPC コンソールからセキュリティグループとサブネット ID を取得できます。

```
{
    "cluster": "tutorial",
    "serviceName": "ecs-service-discovery",
    "taskDefinition": "tutorial-task-def",
    "serviceRegistries": [
       {
          "registryArn":
 "arn:aws:servicediscovery:region:aws_account_id:service/srv-utcrh6wavdkggqtk"
       }
    ],
    "launchType": "FARGATE",
    "platformVersion": "LATEST",
    "networkConfiguration": {
       "awsvpcConfiguration": {
          "assignPublicIp": "ENABLED",
          "securityGroups": [ "sg-abcd1234" ],
```

```
            "subnets": [ "subnet-abcd1234" ]
        }
    },
    "desiredCount": 1
}
```

b. ecs-service-discovery.json を使用して ECS サービスを作成します。

```
aws ecs create-service \
      --cli-input-json file://ecs-service-discovery.json
```

ステップ 3: AWS Cloud Map でサービス検出を検証する

サービス検出情報をクエリして、すべてが正常に作成されたことを確認します。サービス検出を設定した後、AWS Cloud Map API オペレーションを使用するか、VPC 内のインスタンスから dig を呼び出すことができます。以下の手順に従ってください。

1. サービス検出サービス ID を使用して、サービス検出インスタンスを一覧表示します。リソースクリーンアップのインスタンス ID (太字でマーク) を書き留めます。

```
aws servicediscovery list-instances \
      --service-id srv-utcrh6wavdkggqtk
```

出力は次のとおりです。

```
{
    "Instances": [
        {
            "Id": "16becc26-8558-4af1-9fbd-f81be062a266",
            "Attributes": {
                "AWS_INSTANCE_IPV4": "172.31.87.2"
                "AWS_INSTANCE_PORT": "80",
                "AVAILABILITY_ZONE": "us-east-1a",
                "REGION": "us-east-1",
                "ECS_SERVICE_NAME": "ecs-service-discovery",
                "ECS_CLUSTER_NAME": "tutorial",
                "ECS_TASK_DEFINITION_FAMILY": "tutorial-task-def"
            }
        }
    ]
```

```
}
```

2. サービス検出の名前空間、サービス、および ECS クラスター名などの追加パラメータを使用して、サービス検出インスタンスに関する詳細をクエリします。

```
aws servicediscovery discover-instances \
      --namespace-name tutorial \
      --service-name myapplication \
      --query-parameters ECS_CLUSTER_NAME=tutorial
```

3. サービス検出サービス用に Route 53 ホストゾーンに作成された DNS レコードは、次の AWS CLI コマンドでクエリを実行できます。

   a. 名前空間 ID を使用して、名前空間に関する情報を取得しますが、これには Route 53 ホストゾーン ID が含まれます。

```
aws servicediscovery \
      get-namespace --id ns-uejictsjen2i4eeg
```

   出力は次のとおりです。

```
{
    "Namespace": {
        "Id": "ns-uejictsjen2i4eeg",
        "Arn": "arn:aws:servicediscovery:region:aws_account_id:namespace/ns-
uejictsjen2i4eeg",
        "Name": "tutorial",
        "Type": "DNS_PRIVATE",
        "Properties": {
             "DnsProperties": {
                "HostedZoneId": "Z35JQ4ZFDRYPLV"
            }
        },
        "CreateDate": 1519777852.502,
        "CreatorRequestId": "9049a1d5-25e4-4115-8625-96dbda9a6093"
    }
}
```

   b. 前のステップの Route 53 ホストゾーン ID (太字のテキストを参照) を使用して、ホストゾーンのリソースレコードセットを取得します。

```
aws route53 list-resource-record-sets \
```

```
      --hosted-zone-id Z35JQ4ZFDRYPLV
```

4. dig を使用して、VPC 内のインスタンスから DNS をクエリすることもできます。

```
dig +short myapplication.tutorial
```

ステップ 4: クリーンアップする

このチュートリアルが終了したら、未使用のリソースに対する料金が発生しないように、それに関連付けられたリソースをクリーンアップします。以下の手順に従ってください。

1. 前に書き留めたサービス ID とインスタンス ID を使用して、サービス検出サービスインスタンスの登録を解除します。

```
aws servicediscovery deregister-instance \
      --service-id srv-utcrh6wavdkggqtk \
      --instance-id 16becc26-8558-4af1-9fbd-f81be062a266
```

出力は次のとおりです。

```
{
    "OperationId": "xhu73bsertlyffhm3faqi7kumsmx274n-jh0zimzv"
}
```

2. 前のステップの出力の OperationId を使用して、サービス検出インスタンスが正常に登録解除されたことを確認します。

```
aws servicediscovery get-operation \
      --operation-id xhu73bsertlyffhm3faqi7kumsmx274n-jh0zimzv
```

```
{
  "Operation": {
        "Id": "xhu73bsertlyffhm3faqi7kumsmx274n-jh0zimzv",
        "Type": "DEREGISTER_INSTANCE",
        "Status": "SUCCESS",
        "CreateDate": 1525984073.707,
        "UpdateDate": 1525984076.426,
        "Targets": {
            "INSTANCE": "16becc26-8558-4af1-9fbd-f81be062a266",
            "ROUTE_53_CHANGE_ID": "C5NSRG1J4I1FH",
```

```
            "SERVICE": "srv-utcrh6wavdkggqtk"
        }
    }
}
```

3. サービス ID を使用してサービス検出のサービスを削除します。

```
aws servicediscovery delete-service \
      --id srv-utcrh6wavdkggqtk
```

4. 名前空間 ID を使用してサービス検出の名前空間を削除します。

```
aws servicediscovery delete-namespace \
      --id ns-uejictsjen2i4eeg
```

出力は次のとおりです。

```
{
    "OperationId": "c3ncqglftesw4ibgj5baz6ktaoh6cg4t-jh0ztysj"
}
```

5. 前のステップの出力の OperationId を使用して、サービス検出名前空間が正常に削除されたことを確認します。

```
aws servicediscovery get-operation \
      --operation-id c3ncqglftesw4ibgj5baz6ktaoh6cg4t-jh0ztysj
```

出力は次のとおりです。

```
{
    "Operation": {
        "Id": "c3ncqglftesw4ibgj5baz6ktaoh6cg4t-jh0ztysj",
        "Type": "DELETE_NAMESPACE",
        "Status": "SUCCESS",
        "CreateDate": 1525984602.211,
        "UpdateDate": 1525984602.558,
        "Targets": {
            "NAMESPACE": "ns-rymlehshst7hhukh",
            "ROUTE_53_CHANGE_ID": "CJP2A2M86XW3O"
        }
    }
```

```
}
```

6. Amazon ECS サービスに必要な数を更新して `0` にします。次のステップでサービスを削除するには、これを実行する必要があります。

```
aws ecs update-service \
      --cluster tutorial \
      --service ecs-service-discovery \
      --desired-count 0
```

7. Amazon ECS サービスを削除します。

```
aws ecs delete-service \
      --cluster tutorial \
      --service ecs-service-discovery
```

8. Amazon ECS クラスターを削除します。

```
aws ecs delete-cluster \
      --cluster tutorial
```

Amazon VPC Lattice を使用して Amazon ECS サービスを接続、監視、保護する

Amazon VPC Lattice は、コードを変更することなく、Amazon ECS のお客様が AWS コンピューティングサービス、VPC、アカウント全体に 構築されたアプリケーションを観察、保護、モニタリングできるようにする完全マネージド型アプリケーションネットワーキングサービスです。

VPC Lattice は、コンピューティングリソースのコレクションであるターゲットグループを使用します。これらのターゲットは、アプリケーションまたはサービスを実行するものであり、Amazon EC2 インスタンス、IP アドレス、Lambda 関数、Application Load Balancer を含みます。Amazon ECS サービスを VPC Lattice ターゲットグループに関連付けることで、お客様は Amazon ECS タスクを VPC Lattice の IP ターゲットとして有効にできるようになりました。Amazon ECS は、登録されたサービスのタスクが起動されると、VPC Lattice ターゲットグループにタスクを自動的に登録します。

> **Note**
>
> 5 つの VPC Lattice 設定を使用する場合、より少ない設定を使用する場合よりも、デプロイ時間がわずかに長くなることがあります。

リスナールールは、条件が満たされたときに、指定されたターゲットグループにトラフィックを転送するために使用されます。リスナーは、設定したポート上のプロトコルを使用して接続リクエストをチェックします。サービスは、リスナーの設定時に定義したルールに基づいて、登録済みターゲットにリクエストをルーティングします。

また、Amazon ECS は、VPC Lattice ヘルスチェックに基づいてタスクが異常と判断された場合、タスクを自動的に置き換えます。VPC Lattice に関連付けると、Amazon ECS のお客様は、AWS Resource Access Manager を使用したクラスター、VPC、アカウント間のサービスへの接続、認可と認証のための IAM 統合、高度なトラフィック管理機能など、VPC Lattice の他の多くのクロスコンピューティング接続、セキュリティ、オブザーバビリティ機能を利用することもできます。

Amazon ECS のお客様は、以下の方法で VPC Lattice の恩恵を受けることができます。

- デベロッパーの生産性の向上 - VPC Lattice は、ネットワーク、セキュリティ、オブザーバビリティの課題をすべてのコンピューティングプラットフォームにおいて統一された方法で処理することで、機能の構築に集中できるようにし、デベロッパーの生産性を向上させます。
- セキュリティ体制の向上 - VPC Lattice を使用すると、デベロッパーはアプリケーションとコンピューティングプラットフォーム間の通信を簡単に認証および保護し、転送中の暗号化を適用して、VPC Lattice Auth ポリシーを用いたきめ細かなアクセス制御を適用できます。これにより、業界をリードする規制およびコンプライアンス要件を満たす、より強力なセキュリティ体制を採用できます。
- アプリケーションのスケーラビリティと耐障害性の向上 - VPC Lattice では、パス、ヘッダー、メソッドベースのルーティング、認証、認可、モニタリングなどの機能を持つデプロイされたアプリケーションのネットワークを作成できます。これらの利点は、ワークロードのリソースオーバーヘッドなしで提供され、大幅なレイテンシーを発生させることなく、1 秒あたり数百万のリクエストを生成するマルチクラスターデプロイをサポートできます。
- 異種インフラストラクチャによるデプロイの柔軟性 - VPC Lattice は、Amazon ECS、Fargate、Amazon EC2、Amazon EKS、Lambda などのすべてのコンピューティングサービスで一貫した機能を提供し、組織が各アプリケーションに適したインフラストラクチャを柔軟に選択できるようにします。

VPC Lattice が他の Amazon ECS サービスと連携する仕組み

Amazon ECS で VPC Lattice を使用すると、他の Amazon ECS サービスの使用方法が変わる可能性がありますが、他は変わりません。

Application Load Balancer

VPC Lattice の Application Load Balancer ターゲットグループタイプで使用するための、Amazon ECS サービスにリンクする特定の Application Load Balancer を作成する必要がなくなりました。代わりに VPC Lattice ターゲットグループを使用して Amazon ECS サービスを設定するだけで済みます。同時に、Amazon ECS で Application Load Balancer を引き続き使用することを選択することもできます。

Amazon ECS ローリングデプロイ

VPC Lattice では Amazon ECS ローリングデプロイのみが使用可能であり、Amazon ECS はデプロイ中にタスクをサービスに安全に追加および削除します。コードデプロイとブルー/グリーンデプロイはサポートされていません。

VPC Lattice の詳細については、「Amazon VPC Lattice ユーザーガイド」を参照してください。

VPC Lattice を使用するサービスを作成する

AWS マネジメントコンソールまたは AWS CLI のいずれかを使用して、VPC Lattice でサービスを作成できます。

前提条件

個のチュートリアルを開始する前に、次の前提条件を満たしていることを確認します。

- AWS CLI の最新バージョンがインストールされ、設定されていること。詳細については、「AWS Command Line Interface のインストール」を参照してください。

  > **Note**
  >
  > デュアルスタックサービスエンドポイントを使用することで、IPv4 と IPv6 の両方を介して AWS CLI、SDK、および Amazon ECS API から Amazon ECS とやり取りできます。詳細については、「Amazon ECS デュアルスタックエンドポイントの使用」を参照してください。

- 「Amazon ECS を使用するようにセットアップする」で説明されているステップが完了しました。
- IAM ユーザーに AmazonECS_FullAccess IAM ポリシー例で指定されている必要なアクセス権限があること。

AWS マネジメントコンソールで VPC Lattice を使用するサービスを作成する

AWS マネジメントコンソールを使用して VPC Lattice でサービスを作成するには、次のステップに従います。

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションページで、[クラスター] を選択します。
3. [クラスター] ページで、サービスを作成するクラスターを選択します。
4. [Services] (サービス) タブから、[Create] (作成) を選択します。

   以前にサービスを作成したことがない場合は、「コンソールを使用した Amazon ECS サービスの作成」のステップに従い、VPC Lattice セクションまで進んだら、これらのステップを続行します。
5. ボタンをオンにして [VPC Lattice を有効にする] ことを選択します。
6. 既存のロールを使用するには、[Amazon ECS の ECS インフラストラクチャロール] で、VPC Lattice ターゲットグループの作成時に使用する作成済みのロールを選択します。新しいロールを作成するには、[ECS インフラストラクチャロールの作成] を選択します。
7. VPC を選択します。

   [VPC] は、タスク定義を登録したときに選択したネットワークモードによって異なります。EC2 起動タイプで `host` または `network` のモードを使用する場合は、VPC を選択します。

   `awsvpc` モードでは、VPC は [ネットワーク] で選択した VPC に基づいて自動的に選択され、変更することはできません。
8. [ターゲットグループ] で、1 つまたは複数のターゲットグループを選択します。少なくとも 1 つのターゲットグループを選択する必要があります。最大 5 つのターゲットグループを選択できます。追加のターゲットグループを追加するには、[ターゲットグループの追加] を選択します。選択した各ターゲットグループの [ポート名]、[プロトコル]、および [ポート] を選択します。ターゲットグループを削除するには、[削除] を選択します。

   > **Note**
   >
   > - 既存のターゲットグループを追加する場合は、AWS CLI を使用する必要があります。AWS CLI を使用してターゲットグループを追加する方法については、「AWS Command Line Interface リファレンス」の「register-targets」を参照してください。
   > - VPC Lattice サービスは複数のターゲットグループを持つことができますが、各ターゲットグループは 1 つのサービスにのみ追加できます。

- IPv6 専用設定でサービスを作成するには、IP アドレスタイプが IPv6 のターゲットグループを選択します。

9. この時点で、VPC Lattice コンソールに移動してセットアップを続行します。ここでは、リスナーのデフォルトアクションまたは既存の VPC Lattice サービスのルールに新しいターゲットグループを含めます。

   詳細については、「Listener rules for your VPC Lattice service」を参照してください。

> **Important**
>
> セキュリティグループにインバウンドルール vpc-lattice プレフィックスを許可しないと、タスクやヘルスチェックが失敗する可能性があります。

AWS CLIで VPC Lattice を使用するサービスを作成する

AWS CLI を使用して、VPC Lattice でサービスを作成します。各###############を独自の情報に置き換えます。

1. ターゲットグループ設定ファイルを作成します。以下の例は tg-config.json という名前です

```
{
    "ipAddressType": "IPV4",
    "port": 443,
    "protocol": "HTTPS",
    "protocolVersion": "HTTP1",
    "vpcIdentifier": "vpc-f1663d9868EXAMPLE"
}
```

2. 以下のコマンドを使用して、VPC Lattice ターゲットグループを作成します。

```
aws vpc-lattice create-target-group \
    --name my-lattice-target-group-ip \
    --type IP \
    --config file://tg-config.json
```

> **Note**
>
> IPv6 専用設定でサービスを作成するには、IP アドレスタイプが IPv6 のターゲットグループを作成します。詳細については、「AWS CLI コマンドリファレンス」の「create-log-group」を参照してください。

出力例:

```
{
    "arn": "arn:aws:vpc-lattice:us-east-2:123456789012:targetgroup/
tg-0eaa4b9ab4EXAMPLE",
    "config": {
        "healthCheck": {
            "enabled": true,
            "healthCheckIntervalSeconds": 30,
            "healthCheckTimeoutSeconds": 5,
            "healthyThresholdCount": 5,
            "matcher": {
                "httpCode": "200"
            },
            "path": "/",
            "protocol": "HTTPS",
            "protocolVersion": "HTTP1",
            "unhealthyThresholdCount": 2
        },
        "ipAddressType": "IPV4",
        "port": 443,
        "protocol": "HTTPS",
        "protocolVersion": "HTTP1",
        "vpcIdentifier": "vpc-f1663d9868EXAMPLE"
    },
    "id": "tg-0eaa4b9ab4EXAMPLE",
    "name": "my-lattice-target-group-ip",
    "status": "CREATE_IN_PROGRESS",
    "type": "IP"
}
```

3. 以下の *ecs-service-vpc-lattice.json* という名前の JSON ファイルは、VPC Lattice ターゲットグループに Amazon ECS サービスをアタッチするために使用される例です。次の例

の portName は、タスク定義の portMappings プロパティの name フィールドで定義したものと同じです。

```
{
    "serviceName": "ecs-service-vpc-lattice",
    "taskDefinition": "ecs-task-def",
        "vpcLatticeConfigurations": [
        {
            "targetGroupArn": "arn:aws:vpc-lattice:us-
west-2:123456789012:targetgroup/tg-0eaa4b9ab4EXAMPLE",
            "portName": "testvpclattice",
            "roleArn": "arn:aws:iam::123456789012:role/
ecsInfrastructureRoleVpcLattice"
        }
    ],
    "desiredCount": 5,
    "role": "ecsServiceRole"
}
```

次のコマンドを使用して Amazon ECS サービスを作成し、前述の JSON の例を使用して VPC Lattice ターゲットグループにアタッチします。

```
aws ecs create-service \
    --cluster clusterName \
    --serviceName ecs-service-vpc-lattice \
    --cli-input-json file://ecs-service-vpc-lattice.json
```

> **Note**
>
> VPC Lattice は、Amazon ECS Anywhere ではサポートされていません。

Amazon ECS タスクがスケールインイベントによって終了するのを防ぐ

Amazon ECS タスクスケールイン保護を使用すると、サービスの自動スケーリングまたはデプロイからのスケールインイベントによってタスクが終了されるのを防ぐことができます。

特定のアプリケーションでは、使用率が低いときやサービスのデプロイ中に、ミッションクリティカルなタスクがスケールインイベントによって終了されるのを防ぐメカニズムが必要です。例えば、次のようになります。

- ビデオトランスコーディングジョブなどのキュー処理型の非同期アプリケーションでは、サービスの累積使用率が低い場合でも、一部のタスクは何時間も実行する必要があります。
- あるゲームアプリケーションでは、ゲームサーバーは Amazon ECS タスクとして実行されますが、サーバー再起動時の起動待ち時間を短縮するため、すべてのユーザーがログアウトした場合でも実行を続ける必要があります。
- 新しいコードバージョンをデプロイする場合、再処理にはコストがかかるため、タスクは実行し続ける必要があります。

サービスに属するタスクがスケールインイベントで終了されることを防ぐには、`ProtectionEnabled` 属性を `true` に設定します。`ProtectionEnabled` を true に設定すると、タスクはデフォルトで 2 時間保護されます。その後、保護期間は、`ExpiresInMinutes` 属性を使用してカスタマイズできます。タスクの保護期間は最短で 1 分間、最長で 2,880 分 (48 時間) です。AWS CLI を使用している場合は、`--protection-enabled` オプションを指定できます。

タスクが必要な作業を完了した後は、`ProtectionEnabled` 属性を `false` に設定して、以降のスケールインイベントによってタスクが終了されるようにできます。AWS CLI を使用している場合は、`--no-protection-enabled` オプションを指定できます。

タスクスケールイン保護メカニズム

Amazon ECS コンテナエージェントエンドポイントまたは Amazon ECS API を使用して、タスクスケールイン保護を設定および取得できます。

- Amazon ECS コンテナエージェントエンドポイント

  保護の必要性を自己判断できるタスクには、Amazon ECS コンテナエージェントエンドポイントを使用することをお勧めします。このアプローチは、キューベースのワークロードまたはジョブ処理ワークロードに使用します。

  コンテナが、例えば SQS メッセージを使用するなどして処理を開始すると、コンテナ内からタスクスケールイン保護のエンドポイントパス「`$ECS_AGENT_URI/task-protection/v1/state`」を使用して `ProtectionEnabled` 属性を設定できます。スケールインイベントの際、Amazon ECS はこのタスクを終了しません。タスクの処理が完了した後は、同じエンドポイ

ントを使用して ProtectionEnabled 属性をクリアし、以降のスケールインイベントでタスクが終了されるようにできます。

Amazon ECS コンテナエージェントエンドポイントの詳細については、「Amazon ECS タスクスケールイン保護のエンドポイント」を参照してください。

- Amazon ECS API

  アプリケーションにアクティブなタスクの状態を追跡するコンポーネントがある場合は、Amazon ECS API を使用してタスクスケールイン保護を設定および取得できます。UpdateTaskProtection を使用して、1 つ以上のタスクを保護対象としてマークします。GetTaskProtection を使用して保護ステータスを取得します。

  このアプローチの例としては、アプリケーションがゲームサーバーセッションを Amazon ECS タスクとしてホストしている場合が挙げられます。ユーザーがサーバー上のセッション (タスク) にログインすると、そのタスクを保護対象としてマークできます。ユーザーがログアウトした後は、サーバーのアイドル状態を維持することの要件に応じて、このタスク限定でクリアすることも、アクティブなセッションがなくなった同様のタスクについて定期的に保護をクリアすることもできます。

  詳細については、「Amazon Elastic Container Service API リファレンス」の「UpdateTaskProtection」と「GetTaskProtection」を参照してください。

両方のアプローチを組み合わせることができます。例えば、Amazon ECS エージェントエンドポイントを使用してコンテナ内からタスク保護を設定し、Amazon ECS API を使用して外部コントローラーサービスからタスク保護を削除するなどです。

考慮事項

タスクスケールイン保護を使用する前に、次の点を考慮してください。

- タスクのスケールイン保護は、サービスからデプロイされたタスクでのみサポートされます。
- タスクのスケールイン保護は、Amazon ECS マネージドインスタンスで実行されているサービスからデプロイされたタスクでサポートされます。
- Amazon ECS エージェントには再試行メカニズムが組み込まれており、そのインターフェイスはよりシンプルであるため、Amazon ECS コンテナエージェントエンドポイントを使用することをお勧めします。
- 既に保護がオンになっているタスクのために UpdateTaskProtection を呼び出すことで、タスクのスケールイン保護の有効期間をリセットできます。

- タスクが必要な作業を完了するのに必要な時間を判断し、それに応じて `expiresInMinutes` プロパティを設定してください。保護の有効期限を必要以上に長く設定すると、コストが発生すると共に、新しいタスクのデプロイが遅れることになります。
- `DEFERRED` ソースサービスリビジョンのクリーンアップを使用すると、Amazon ECS はデプロイの完了後にソースサービスリビジョンのタスクをクリーンアップし、最大 2 週間試行します。Amazon ECS は、スケールイン保護が設定されているタスクを停止できません。デプロイ後 2 週間以上タスクが保護された状態のままである場合、Amazon ECS はそれらのタスクをクリーンアップできません。詳細については、「早期成功基準を使用した Amazon ECS ローリングデプロイの早期完了」を参照してください。
- Amazon ECS コンテナエージェント `1.65.0` 以降ではタスクのスケールイン保護がサポートされています。エージェントを最新バージョンに更新することで、古いバージョンのAmazon ECS コンテナエージェントを使用する Amazon EC2 インスタンスでこの機能のサポートを追加できます。詳細については、「Amazon ECS コンテナエージェントをアップデートする」を参照してください。
- デプロイに関する考慮事項:
  - サービスがローリングアップデートを使用する場合、新しいタスクは作成できますが、古いバージョンを実行しているタスクは、`protectionEnabled` がクリアされるか、または失効するまで終了しません。デプロイ設定の `maximumPercentage` パラメータは、古いタスクが保護されている場合でも新しいタスクを作成できるように値を調整できます。
  - ブルー/グリーン更新が適用されている場合、タスクに `protectionEnabled` があれば、保護されたタスクを含むブルーデプロイは削除されません。トラフィックは新しいタスクに振り分けられ、古いタスクは `protectionEnabled` がクリアされるか期限切れになったときのみ削除されます。CodeDeploy または CloudFormation の更新のタイムアウトによっては、デプロイがタイムアウトして、古いブルータスクが残っている場合があります。
  - CloudFormation を使用する場合、更新スタックのタイムアウトは 3 時間です。そのため、タスク保護を 3 時間以上設定すると、CloudFormation のデプロイで障害が発生してロールバックが発生する可能性があります。

    古いタスクが保護されている間、CloudFormation スタックは `UPDATE_IN_PROGRESS` を表示します。タスクのスケールイン保護が削除されるか、または 3 時間のウィンドウ内に失効する場合、デプロイは成功し、`UPDATE_COMPLETE` ステータスに移行します。デプロイが `UPDATE_IN_PROGRESS` のまま 3 時間以上滞っていると、デプロイは失敗して `UPDATE_FAILED` 状態が表示され、古いタスクセットにロールバックされてしまいます。
  - 保護されたタスクが原因でデプロイ (ローリングまたはブルー/グリーン) が定常状態に到達できない場合、Amazon ECS はサービスイベントを送信し、これによりユーザーは是正措置を講じ

ることができます。タスクの保護ステータスを更新しようとする際に DEPLOYMENT_BLOCKED エラーメッセージが表示される場合、サービスの要求されるタスク数よりも多くの保護されたタスクがあることを意味します。この問題を解決するには、次のいずれかの操作を実行します。

- 現在のタスク保護の有効期限が切れるのを待ちます。次に、タスク保護を設定します。
- どのタスクを停止できるかを決定します。その後、これらのタスクについて protectionEnabled オプションを false に設定して UpdateTaskProtection を使用します。
- サービスの必要なタスク数を増やして、保護されているタスクの数よりも多くします。

タスクスケールイン保護に必要な IAM アクセス許可

タスクには、以下のアクセス許可を持つ Amazon ECS タスクロールが必要です。

- ecs:GetTaskProtection: Amazon ECS コンテナエージェントが GetTaskProtection を呼び出すことを許可します。
- ecs:UpdateTaskProtection: Amazon ECS コンテナエージェントが UpdateTaskProtection を呼び出すことを許可します。

Amazon ECS タスクスケールイン保護のエンドポイント

Amazon ECS コンテナエージェントは、ECS_AGENT_URI 環境変数を Amazon ECS タスクのコンテナに自動的に挿入して、コンテナエージェント API エンドポイントとやり取りする方法を提供します。

保護の必要性を自己判断できるタスクには、Amazon ECS コンテナエージェントエンドポイントを使用することをお勧めします。

コンテナが処理を開始すると、コンテナ内からタスクスケールイン保護のエンドポイントパス「$ECS_AGENT_URI/task-protection/v1/state」を使用して protectionEnabled 属性を設定できます。

コンテナ内からこの URI への PUT リクエストを使用すると、タスクのスケールイン保護が設定されます。この URI への GET リクエストは、タスクの現在の保護ステータスを返します。

タスクスケールイン保護のリクエストパラメータ

次のリクエストパラメータで、${ECS_AGENT_URI}/task-protection/v1/state エンドポイントを使用してタスクスケールインプロテクションを設定できます。

ProtectionEnabled

true を指定して、タスクを保護できるようにマークします。false を指定して保護を解除し、タスクを終了できるようにします。

タイプ: ブール値

必須: はい

ExpiresInMinutes

タスクが保護されている時間 (分)。最小 1 分から最大 2,880 分 (48 時間) まで指定できます。この期間中、自動スケーリングサービスまたはデプロイからのスケールインイベントによってタスクが終了することはありません。この時間が経過すると、protectionEnabled パラメータは false に設定されます。

時間を指定しない場合、タスクは自動的に 120 分 (2 時間) 保護されます。

タイプ: 整数

必須: いいえ

次の例は、異なる継続時間を持つタスク保護を設定する方法を示します。

デフォルトの期間を使用してタスクを保護する方法の例

この例は、デフォルトの期間が 2 時間に設定されているタスクを保護する方法を示しています。

```
curl --request PUT --header 'Content-Type: application/json' ${ECS_AGENT_URI}/task-
protection/v1/state --data '{"ProtectionEnabled":true}'
```

タスクを 60 分間保護する方法の例

この例は、expiresInMinutes パラメータを使用してタスクを 60 分間保護する方法を示しています。

```
curl --request PUT --header 'Content-Type: application/json' ${ECS_AGENT_URI}/task-
protection/v1/state --data '{"ProtectionEnabled":true,"ExpiresInMinutes":60}'
```

タスクを 24 時間保護する方法の例

この例は、expiresInMinutes パラメータを使用してタスクを 24 時間保護する方法を示しています。

```
curl --request PUT --header 'Content-Type: application/json' ${ECS_AGENT_URI}/task-
protection/v1/state --data '{"ProtectionEnabled":true,"ExpiresInMinutes":1440}'
```

Windows コンテナの例

Windows コンテナの場合、curl の代わりに PowerShell の Invoke-RestMethod コマンドレットを使用できます。次の例は、前の curl コマンドに相当する PowerShell コマンドを示しています。

デフォルト期間を持つ Windows コンテナタスクを保護する方法の例

この例では、PowerShell を使用して 2 時間のデフォルト期間を持つタスクを保護する方法を示しています。

```
Invoke-RestMethod -Uri $env:ECS_AGENT_URI/task-protection/v1/state -Method Put -Body
 '{"ProtectionEnabled":true}' -ContentType 'application/json'
```

Windows コンテナタスクを 60 分間保護する方法の例

この例では、PowerShell で expiresInMinutes パラメータを使用してタスクを 60 分間保護する方法を示しています。

```
Invoke-RestMethod -Uri $env:ECS_AGENT_URI/task-protection/v1/state -Method Put -Body
 '{"ProtectionEnabled":true,"ExpiresInMinutes":60}' -ContentType 'application/json'
```

Windows コンテナタスクを 24 時間保護する方法の例

この例では、expiresInMinutes パラメータを使用してタスクを 24 時間保護する方法を示しています。

```
Invoke-RestMethod -Uri $env:ECS_AGENT_URI/task-protection/v1/state -Method Put -Body
 '{"ProtectionEnabled":true,"ExpiresInMinutes":1440}' -ContentType 'application/json'
```

PUT リクエストは次のレスポンスを返します。

```
{
  "protection": {
    "ExpirationDate": "2023-12-20T21:57:44.837Z",
```

```
    "ProtectionEnabled": true,
    "TaskArn": "arn:aws:ecs:us-west-2:111122223333:task/1234567890abcdef0"
  }
}
```

タスクスケールイン保護のレスポンスパラメータ

次の情報が、JSON レスポンスタスクのスケールインプロテクションエンドポイント ${ECS_AGENT_URI}/task-protection/v1/state から返されます。

ExpirationDate

タスクの保護が期限切れになるエポックタイム。タスクが保護されていない場合、この値は NULL です。

ProtectionEnabled

タスクのプロテクションステータス。タスクのスケールインプロテクションが有効になっている場合、値は true です。それ以外の場合は、false です。

TaskArn

コンテナが属しているタスクの完全な Amazon リソースネーム (ARN)。

次の例は、保護されたタスクについて返される詳細を示しています。

```
curl --request GET ${ECS_AGENT_URI}/task-protection/v1/state
```

Windows コンテナの場合、次の PowerShell コマンドを使用して保護ステータスを取得します。

```
Invoke-RestMethod -Uri $env:ECS_AGENT_URI/task-protection/v1/state -Method Get
```

```
{
    "protection":{
        "ExpirationDate":"2023-12-20T21:57:44Z",
        "ProtectionEnabled":true,
        "TaskArn":"arn:aws:ecs:us-west-2:111122223333:task/1234567890abcdef0"
    }
}
```

障害が発生すると、次の情報が返されます。

`Arn`

タスクの完全な Amazon リソースネーム (ARN)。

`Detail`

障害に関する詳細。

`Reason`

失敗の理由。

次の例は、保護されていないタスクについて返される詳細を示しています。

```
{
    "failure":{
        "Arn":"arn:aws:ecs:us-west-2:111122223333:task/1234567890abcdef0",
        "Detail":null,
        "Reason":"TASK_NOT_VALID"
    }
}
```

例外が発生すると、次の情報が返されます。

`requestID`

例外が発生する Amazon ECS API 呼び出しの AWS リクエスト ID。

`Arn`

タスクまたはサービスの完全な Amazon リソースネーム (ARN)。

`Code`

エラーコードです。

`Message`

エラーメッセージです。

> **Note**
>
> `RequestError` または `RequestTimeout` エラーが表示される場合は、ネットワークに問題がある可能性があります。Amazon ECS の VPC エンドポイントを使用してみてください。

次の例は、エラーが発生したときに返される詳細を示したものです。

```
{
    "requestID":"12345-abc-6789-0123-abc",
    "error":{
        "Arn":"arn:aws:ecs:us-west-2:555555555555:task/my-cluster-
name/1234567890abcdef0",
        "Code":"AccessDeniedException",
        "Message":"User: arn:aws:sts::444455556666:assumed-role/my-ecs-task-
role/1234567890abcdef0 is not authorized to perform: ecs:GetTaskProtection on resource:
 arn:aws:ecs:us-west-2:555555555555:task/test/1234567890abcdef0 because no identity-
based policy allows the ecs:GetTaskProtection action"
    }
}
```

ネットワークの問題や Amazon ECS コントロールプレーンがダウンしているなどの理由で Amazon ECS エージェントが Amazon ECS エンドポイントから応答を得られない場合、次のエラーが表示されます。

```
{
  "error": {
    "Arn": "arn:aws:ecs:us-west-2:555555555555:task/my-cluster-name/1234567890abcdef0",
    "Code": "RequestCanceled",
    "Message": "Timed out calling Amazon ECS Task Protection API"
  }
}
```

Amazon ECS エージェントが Amazon ECS からスロットリング例外を受け取ると、次のエラーが表示されます。

```
{
  "requestID": "12345-abc-6789-0123-abc",
  "error": {
    "Arn": "arn:aws:ecs:us-west-2:555555555555:task/my-cluster-name/1234567890abcdef0",
    "Code": "ThrottlingException",
    "Message": "Rate exceeded"
  }
}
```

Amazon ECS および Fargate ワークロードでフォールトインジェクションを使用する

お客様は、Amazon EC2 と Fargate の両方で Amazon ECS によるフォールトインジェクションを利用して、特定の障害シナリオに対するアプリケーションの反応をテストできます。これらのテストは、アプリケーションのパフォーマンスと回復力の最適化に使用できる情報を提供します。

フォールトインジェクションが有効になっている場合、Amazon ECS コンテナエージェントはタスクに新しいフォールトインジェクションエンドポイントへのアクセスを許可します。フォールトインジェクションを使用するには、`enableFaultInjection` タスク定義パラメータの値を `true` に設定してオプトインする必要があります。デフォルト値は `false` です。

```
{
    ...
   "enableFaultInjection": true
}
```

> **Note**
>
> フォールトインジェクションは、`awsvpc` または `host` ネットワークモードを使用するタスクでのみ機能します。
> フォールトインジェクションは、Windows では使用できません。

AWS マネジメントコンソールでフォールトインジェクションを有効にする方法については、「コンソールを使用して Amazon ECS タスク定義の作成」を参照してください。

AWS Fault Injection Service でテストするには、この機能を有効にする必要があります。詳細については、「AWS FIS aws:ecs:タスクアクションを使用します」を参照してください。

> **Note**
>
> Amazon ECS に最適化された新しい AMI を使用しない場合、またはカスタム AMI がある場合は、次の依存関係をインストールします。
>
> - `tc`
> - `sch_netem` カーネルモジュール

Amazon ECS フォールトインジェクションエンドポイント

Amazon ECS コンテナエージェントは、ECS_AGENT_URI 環境変数を Amazon ECS タスクのコンテナに自動的に挿入して、コンテナエージェント API エンドポイントとやり取りする方法を提供します。各エンドポイントには、/start、/stop、および /status エンドポイントが含まれます。エンドポイントは、フォールトインジェクションを有効にしたタスクからのリクエストのみを受け入れ、各エンドポイントにはコンテナあたり 5 秒間に 1 リクエストというレート制限があります。この制限を超えるとエラーが発生します。

> **Note**
>
> フォールトインジェクションエンドポイントを使用するには、Amazon ECS エージェント version 1.88.0+ が必要です。

フォールトインジェクションで使用する 3 つのエンドポイントは以下のとおりです。

- ネットワークブラックホールポートエンドポイント
- ネットワークパケット損失エンドポイント
- ネットワークレイテンシーエンドポイント

リクエストが成功すると、/start エンドポイントを呼び出す場合 running のメッセージが記載された 200 応答コードとなり、/stop エンドポイントの場合は stopped、/status エンドポイントの場合は running または not-running となります。

```
{
    "Status": <string>
}
```

リクエストが成功しなかった場合は、以下のいずれかのエラーコードを返します。

- 400 – Bad request·
- 409 – フォールトインジェクションリクエストが別の実行中の障害と競合する
- 429 – リクエストがスロットリングされました
- 500 – サーバーに予期しないエラーが発生しました

```
{
```

```
 "Error":  <string message>
}
```

> **Note**
>
> 1 回に挿入できるのは、1 つのネットワークレイテンシー障害、または 1 つのネットワークパケット損失障害のいずれかです。複数の結果を挿入しようとすると、リクエストは拒否されます。

ネットワークブラックホールポートエンドポイント

`{ECS_AGENT_URI}/fault/v1/network-blackhole-port` エンドポイントは、タスクのネットワーク名前空間内の特定のポートとプロトコルの受信トラフィックまたは送信トラフィックをドロップし、次の 2 つのモードと互換性があります。

- awsvpc — 変更がタスクネットワーク名前空間に適用されます
- ホスト — 変更はデフォルトのネットワーク名前空間コンテナインスタンスに適用されます

{ECS_AGENT_URI}/fault/v1/network-blackhole-port/start

このエンドポイントは、ネットワークブラックホールポートのフォールトインジェクションを開始し、次のパラメータがあります。

ポート

ブラックホールポートのフォールトインジェクションに使用する指定されたポート。

タイプ: 整数

必須: はい

プロトコル

ブラックホールポートのフォールトインジェクションに使用するプロトコル。

型: 文字列

有効な値: `tcp | udp`

必須: はい

TrafficType

フォールトインジェクションで使用されるトラフィックタイプ。

型: 文字列

有効な値: `ingress | egress`

必須: はい

SourcesToFilter

障害から保護されている IPv4 アドレスまたは IPv6 アドレス、あるいは CIDR ブロックの JSON 配列。

型: 文字列の配列

必須: いいえ

以下は `start` エンドポイントを使用するためのリクエストの例です (*##* の値を独自の値に置き換えてください)。

```
Endpoint: ${ECS_AGENT_URI}/fault/v1/network-blackhole-port/start

Http method:POST

Request payload:
{
    "Port": 1234,
    "Protocol": "tcp|udp",
    "TrafficType": "ingress|egress"
    "SourcesToFilter": ["${IP1}", "${IP2}", ...],
}
```

{ECS_AGENT_URI}/fault/v1/network-blackhole-port/stop

このエンドポイントは、リクエストで指定された障害を停止します。このエンドポイントには以下のパラメータがあります。

ポート

停止する必要がある障害の影響を受けたポート。

タイプ: 整数

必須: はい

プロトコル

障害を停止するために使用するプロトコル。

型: 文字列

有効な値: `tcp | udp`

必須: はい

TrafficType

フォールトインジェクションで使用されるトラフィックタイプ。

型: 文字列

有効な値: `ingress | egress`

必須: はい

以下は `stop` エンドポイントを使用するためのリクエストの例です (`##` の値を独自の値に置き換えてください)。

```
Endpoint: ${ECS_AGENT_URI}/fault/v1/network-blackhole-port/stop

Http method: POST

Request payload:
{
    "Port": 1234,
    "Protocol": "tcp|udp",
    "TrafficType": "ingress|egress",
}
```

{ECS_AGENT_URI}/fault/v1/network-blackhole-port/status

このエンドポイントは、フォールトインジェクションのステータスを確認するために使用されます。このエンドポイントには以下のパラメータがあります。

ポート

障害のステータスを確認するために影響を受けるポート。

タイプ: 整数

必須: はい

プロトコル

障害のステータスを確認する際に使用するプロトコル。

型: 文字列

有効な値: `tcp | udp`

必須: はい

TrafficType

フォールトインジェクションで使用されるトラフィックタイプ。

型: 文字列

有効な値: `ingress | egress`

必須: はい

以下は `status` エンドポイントを使用するためのリクエストの例です (`##` の値を独自の値に置き換えてください)。

```
Endpoint: ${ECS_AGENT_URI}/fault/v1/network-blackhole-port/status

Http method: POST

Request payload:
{
   "Port": 1234,
   "Protocol": "tcp|udp",
   "TrafficType": "ingress|egress",
}
```

ネットワークレイテンシーエンドポイント

{ECS_AGENT_URI}/fault/v1/network-latency エンドポイントは、特定のソースへのトラフィックに対してタスクのネットワークインターフェイスに遅延とジッターを追加します。このエンドポイントは 2 つのモードと互換性があります。

- awsvpc — 変更がタスクネットワークインターフェイスに適用されます
- ホスト — 変更はデフォルトのネットワークインターフェイスに適用されます

{ECS_AGENT_URI}/fault/v1/network-latency/start

この/start エンドポイントは、ネットワークレイテンシーのフォールトインジェクションを開始し、以下のパラメータがあります。

DelayMilliseconds

フォールトインジェクションに使用するネットワークインターフェイスに追加する遅延のミリ秒数。

タイプ: 整数

必須: はい

JitterMilliseconds

フォールトインジェクションに使用するネットワークインターフェイスに追加するジッターのミリ秒数。

タイプ: 整数

必須: はい

ソース

フォールトインジェクションで使用する宛先である IPv4 アドレスまたは IPv6 アドレス、あるいは CIDR ブロックの JSON 配列。

型: 文字列の配列

必須: はい

SourcesToFilter

障害から保護されている IPv4 アドレスまたは IPv6 アドレス、あるいは CIDR ブロックの JSON 配列。SourcesToFilter は Sources よりも優先されます。

型: 文字列の配列

必須: いいえ

以下は /start エンドポイントを使用するためのリクエストの例です (## の値を独自の値に置き換えてください)。

```
Endpoint: ${ECS_AGENT_URI}/fault/v1/network-latency/start

Http method: POST

Request payload:
{
    "DelayMilliseconds": 123,
    "JitterMilliseconds": 123,
    "Sources": ["${IP1}", "${IP2}", ...],
    "SourcesToFilter": ["${IP1}", "${IP2}", ...],
}
```

{ECS_AGENT_URI}/fault/v1/network-latency/stop and /status

{ECS_AGENT_URI}/fault/v1/network-latency/stop エンドポイントは障害を停止し、{ECS_AGENT_URI}/fault/v1/network-latency/status は障害のステータスを確認します。

以下は、 /stop と /status のエンドポイントを使用する際の 2 つのリクエストの例です。どちらも POST HTTP メソッドを使用します。

```
Endpoint: ${ECS_AGENT_URI}/fault/v1/network-latency/stop
```

```
Endpoint: ${ECS_AGENT_URI}/fault/v1/network-latency/status
```

ネットワークパケット損失エンドポイント

{ECS_AGENT_URI}/fault/v1/network-packet-loss エンドポイントは、指定されたネットワークインターフェイスへのパケット損失を追加します。このエンドポイントは、次の 2 つのモードと互換性があります。

- awsvpc — 変更がタスクネットワークインターフェイスに適用されます
- ホスト — 変更はデフォルトのネットワークインターフェイスに適用されます

{ECS_AGENT_URI}/fault/v1/network-packet-loss/start

この /start エンドポイントは、ネットワークパケット損失フォールトインジェクションを開始し、以下のパラメータがあります。

LossPercent

パケット損失の割合

タイプ: 整数

必須: はい

ソース

フォールトインジェクションテストに使用する IPv4 アドレスまたは IPv6 アドレス、あるいは CIDR ブロックの JSON 配列。

型: 文字列の配列

必須: はい

SourcesToFilter

障害から保護されている IPv4 アドレスまたは IPv6 アドレス、あるいは CIDR ブロックの JSON 配列。SourcesToFilter は Sources よりも優先されます。

型: 文字列の配列

必須: いいえ

以下は start エンドポイントを使用するためのリクエストの例です (## の値を独自の値に置き換えてください)。

```
Endpoint: ${ECS_AGENT_URI}/fault/v1/network-packet-loss/start

Http method: POST

{
    "LossPercent": 6,
    "Sources": ["${IP1}", "${IP2}", ...],
    "SourcesToFilter": ["${IP1}", "${IP2}", ...],
}
```

{ECS_AGENT_URI}/fault/v1/network-packet-loss/stop and /status

{ECS_AGENT_URI}/fault/v1/network-packet-loss/stop エンドポイントは障害を停止し、{ECS_AGENT_URI}/fault/v1/network-packet-loss/status は障害のステータスを確認します。1 回でサポートされる障害のタイプは 1 つだけです。

以下は、/stop と /status のエンドポイントを使用する際の 2 つのリクエストの例です。どちらも POST HTTP メソッドを使用します。

```
Endpoint: ${ECS_AGENT_URI}/fault/v1/network-packet-loss/stop
```

```
Endpoint: ${{ECS_AGENT_URI}/fault/v1/network-packet-loss/status
```

Amazon ECS サービスのパラメータの更新

サービスを作成した後、タスクの数など、サービスパラメータを更新する必要がある場合があります。

サービススケジューラが新しいタスクを起動するとき、次のロジックを使用してクラスター内のタスク配置を決定します。

- クラスター内のどのコンテナインスタンスがサービスのタスク定義をサポートできるか判断します。例えば、必要な CPU、メモリ、ポート、コンテナインスタンス属性があります。
- デフォルトで、サービススケジューラは、別の配置戦略を選択できる場合でも、この方法でアベイラビリティーゾーン間でタスクのバランスをとるように試みます。
  - 有効なコンテナインスタンスを、インスタンスと同じアベイラビリティーゾーンでサービスの実行中タスクが少ない順にソートします。例えば、ゾーン A に実行中のサービスタスクが 1 つあり、ゾーン B と C に実行中のサービスタスクがない場合、ゾーン B または C のいずれかの有効なコンテナインスタンスが配置に最適と見なされます。
  - 新しいサービスタスクを最適なアベイラビリティーゾーン内の有効なコンテナインスタンスに配置し (前の手順に基づいて)、実行中のサービスタスクの数が最小のコンテナインスタンスを優先させます。

サービススケジューラがタスクの実行を停止する場合、次のロジックを使用してアベイラビリティーゾーン間のクラスターの負荷バランスを維持します。

- コンテナインスタンスを、インスタンスと同じアベイラビリティーゾーンでサービスの実行中タスクが多い順にソートします。たとえば、実行中のサービスタスクがゾーン A には 1 つ、ゾーン B

とゾーン C にはそれぞれ 2 つずつある場合、ゾーン B またはゾーン C のいずれかのコンテナインスタンスが終了に最適と見なされます。

- 実行中のサービスタスクの数が多いコンテナインスタンスの順に従って、最適なアベイラビリティーゾーン内のコンテナインスタンスで (前述の手順に基づいて) タスクを停止します。

リストを使用して、サービスパラメータを変更できるかどうかを判断してください。

アベイラビリティーゾーンの再調整

サービスにアベイラビリティーゾーンの再調整を使用するかどうか示します。

このパラメータはローリングデプロイで変更できます。

キャパシティプロバイダー戦略

キャパシティプロバイダー戦略の詳細。クラスターの作成、タスクの実行、またはサービスの更新時に、キャパシティプロバイダーを設定できます。

Fargate を使用する場合、キャパシティプロバイダーは FARGATE または FARGATE_SPOT です。

Amazon EC2 を使用する場合、キャパシティプロバイダーは Auto Scaling グループです。

ローリングデプロイとブルー/グリーンデプロイのキャパシティプロバイダーは変更できます。

有効な移行を参照するには、「サービスミュータビリティ」を参照してください。

クラスター

クラスターの名前を変更することはできません。

デプロイ設定

デプロイ設定には、CloudWatch アラーム、障害の検出に使用されるサーキットブレーカー、必要な設定が含まれます。

デプロイサーキットブレーカーは、サービスが定常状態に到達できなかった場合にサービスのデプロイを失敗させるかどうかを決定します。デプロイサーキットブレーカーを使用すると、サービスデプロイは失敗状態に移行し、新しいタスクの起動を停止します。ロールバックオプションを使用すると、サービスのデプロイに失敗すると、サービスは、前回正常に完了したデプロイへとロールバックされます。

Amazon ECS サーキットブレーカーを使用するサービスを更新すると、Amazon ECS はサービスデプロイとサービスリビジョンを作成します。これらのリソースを使用すると、サービス履歴

に関する詳細情報を表示できます。詳細については、「Amazon ECS サービスデプロイを使用してサービス履歴を表示する」を参照してください。

サービススケジューラは、最小ヘルス率と最大ヘルス率のパラメータ (サービスのデプロイ設定) を使用して、デプロイ戦略を判断します。

サービスでローリング更新 (ECS) のデプロイタイプが使用されている場合、最小ヘルス率は、デプロイ時に RUNNING 状態に保つ必要のあるサービスのタスクの下限数をサービスのタスクの必要数のパーセント値 (最も近い整数に切り上げ) で表します。サービスに EC2 を使用するタスクが含まれている場合、DRAINING 状態のコンテナインスタンスがある間は、パラメータも適用されます。追加のクラスターキャパシティーを使用しないデプロイのために、このパラメータを使用します。例えば、サービスで必要数が 4 タスク、最小ヘルス率が 50% とすると、スケジューラは 2 つの新しいタスクを開始する前に、2 つの既存のタスクを停止してクラスターのキャパシティーを解放できます。ロードバランサーを使用しないサービスのタスクは、RUNNING 状態にある場合正常な状態とサービスからみなされます。ロードバランサーを使用するサービスのタスクは、RUNNING 状態にあり、ロードバランサーによって正常と報告された場合に、サービスから正常であるとみなされます。最小ヘルス率のデフォルト値は 100% です。

サービスでローリング更新 (ECS) のデプロイタイプが使用されている場合、maximum percent パラメータは、デプロイ時に PENDING、RUNNING、または STOPPING 状態で使用できるサービスのタスクの上限数をそれらのタスク数の適切なパーセント値 (最も近い整数に切り下げ) として表します。サービスに EC2 を使用するタスクが含まれている場合、DRAINING 状態のコンテナインスタンスがある間は、パラメータも適用されます。このパラメータは、デプロイのバッチサイズを定義するために使用します。例えば、サービスで必要数が 4 タスク、最大ヘルス率の値が 200% とすると、スケジューラは 4 つの古いタスクを停止する前に、4 つの新しいタスクを開始できます。そのために必要なクラスターリソースを使用できることが前提です。最大ヘルス率のデフォルト値は 200% です。

更新中にサービススケジューラがタスクを置き換えるとき、サービスはまずロードバランサーからタスクを削除し (使用されている場合)、接続のドレインが完了するのを待ちます。その後、タスクで実行されているコンテナに docker stop と同等のコマンドが発行されます。この結果、SIGTERM 信号と 30 秒のタイムアウトが発生し、その後に SIGKILL が送信され、コンテナが強制的に停止されます。コンテナが SIGTERM 信号を正常に処理し、その受信時から 30 秒以内に終了する場合、SIGKILL 信号は送信されません。サービススケジューラは、最小ヘルス率と最大ヘルス率の設定で定義されているとおりに、タスクを開始および停止します。

また、コンテナのヘルスチェックまたはロードバランサーのターゲットグループのヘルスチェックが失敗すると、サービススケジューラーによって、異常であると判断されたタスクが置き換えられます。この置き換え動作は、maximumPercent および desiredCount のサービス定義パラ

メータによって異なります。タスクが異常とマークされた場合、サービススケジューラーによってまず置き換えタスクが開始されます。次に以下が発生します。

- 置き換えタスクのヘルスステータスが HEALTHY になると、サービススケジューラーが異常のあるタスクを停止します。
- 置き換えタスクのヘルスステータスが UNHEALTHY の場合、スケジューラーは異常のある置き換えタスクまたは既存の異常タスクのいずれかを停止して、タスク総数が desiredCount と等しくなるようにします。

maximumPercent パラメーターによって、置き換えタスクを先に開始できないようにスケジューラーが制限されている場合、スケジューラーは異常のあるタスクをランダムに 1 つずつ停止して容量を解放してから置き換えタスクを開始します。異常のあるタスクがすべて正常なタスクに置き換えられるまで、起動と停止のプロセスが続きます。異常なタスクがすべて置き換えられ、正常なタスクだけが実行中になると、合計タスク数が desiredCount を超える場合、タスク数が desiredCount になるまで、正常なタスクが無作為に停止されます。maximumPercent および desiredCount の詳細については、「サービス定義パラメータ」を参照してください。

デプロイコントローラー

サービスで使用するデプロイコントローラータイプ。次の 3 つのデプロイコントローラータイプを使用できます。

- ECS
- EXTERNAL
- CODE_DEPLOY

サービスを更新すると、使用されるデプロイコントローラーを更新できます。次のリストは、有効な移行を表示しています。

- CodeDeploy ブルー/グリーンデプロイ (CODE_DEPLOY) から ECS ローリングデプロイまたはブルー/グリーンデプロイ (ECS) に更新します。
- CodeDeploy ブルー/グリーンデプロイ (CODE_DEPLOY) から外部デプロイ (EXTERNAL) に更新します。
- ECS ローリングまたはブルー/グリーンデプロイ (ECS) から外部デプロイ (EXTERNAL) に更新します。
- 外部デプロイ (EXTERNAL) から ECS ローリングデプロイまたはブルー/グリーンデプロイ (ECS) に更新します。

サービスのデプロイコントローラーを更新する際には、次の点を考慮してください。

- VPC Lattice または Amazon ECS Service Connect を使用している場合、サービスのデプロイコントローラーを ECS デプロイコントローラーから他のコントローラーに更新することはできません。
- サービスのデプロイ中に、サービスのデプロイコントローラーを更新することはできません。
- サービスにロードバランサーがない場合、サービスのデプロイコントローラーを CODE_DEPLOY に更新することはできません。
- deploymentConfiguration にアラーム、デプロイサーキットブレーカー、BLUE_GREEN デプロイ戦略が含まれている場合、サービスのデプロイコントローラーを ECS から他のコントローラーに更新することはできません。詳細については、「Amazon ECS サービスデプロイコントローラーと戦略」を参照してください。
- サービスのデプロイコントローラーを ECS から他のコントローラーに更新した場合、コンテナ定義で versionConsistency に指定した値は Amazon ECS で使用されません。
- サービスのデプロイコントローラーを ECS から他のコントローラーに更新しても、UpdateService および DescribeService API レスポンスは依然として taskSets ではなく、deployments を返します。UpdateService および CreateService の詳細については、「Amazon ECS API Reference」の「UpdateService」および「CreateService」を参照してください。
- サービスがローリング更新のデプロイ戦略を使用している場合、デプロイコントローラーを ECS から他のコントローラーに更新すると、deploymentConfiguration の maximumPercent 値の使用方法が変更されます。ローリング更新デプロイで合計タスクの上限として使用されるだけではなく、maximumPercent は異常なタスクを置き換えるために使用されます。スケジューラが異常なタスクを置き換える方法の詳細については、「Amazon ECS サービス」を参照してください。
- サービスのデプロイコントローラーを ECS から他のデプロイコントローラーに更新した場合、ロードバランサー設定で指定した advancedConfiguration は無視されます。詳細については、「Amazon ECS API リファレンス」の「LoadBalancer」および「AdvancedConfiguration」を参照してください。

CloudFormation を使用してサービスのデプロイコントローラーを更新する際には、実行する移行のタイプに応じて次の点を考慮してください。

- EXTERNAL デプロイコントローラー情報に加え、TaskSet および PrimaryTaskSetリソースを含む CloudFormation テンプレートがあって、EXTERNAL から ECS に更新するときにテンプレートからタスクセットのリソースを削除する場合、デプロイコントローラーが ECS に更新されると、DescribeTaskSet および DeleteTaskSet API コールは 400 エラーを返します。CloudFormation スタックを UPDATE_COMPLETE ステータスに移行しても、タスクセット

のリソースで CloudFormation の削除が失敗します。詳細については、「AWS CloudFormation ユーザーガイド」の「リソースはスタックから削除されましたが、削除されない」を参照してください。この問題を解決するには、Amazon ECS `DeleteTaskSet` API を使用してタスクセットを直接削除します。タスクセットを削除する方法の詳細については、「Amazon Elastic Container Service API リファレンス」の「DeleteTaskSet」を参照してください。

- 新しいタスク定義で `CODE_DEPLOY` から `ECS` に移行して、CloudFormation でロールバックオペレーションを実行すると、Amazon ECS `UpdateService` リクエストは次のエラーで失敗します。

```
Resource handler returned message: "Invalid request provided: Unable to update
task definition on services with a CODE_DEPLOY deployment controller. Use AWS
CodeDeploy to trigger a new deployment. (Service: Ecs, Status Code: 400,
Request ID: 0abda1e2-f7b3-4e96-b6e9-c8bc585181ac) (SDK Attempt Count: 1)"
(RequestToken: ba8767eb-c99e-efed-6ec8-25011d9473f0, HandlerErrorCode:
 InvalidRequest)
```

- `ECS` から `EXTERNAL` デプロイコントローラーへの移行が正常に完了した後には、Amazon ECS がデプロイを管理しなくなるため、`ACTIVE` タスクセットを手動で削除する必要があります。タスクセットを削除する方法の詳細については、「Amazon Elastic Container Service API リファレンス」の「DeleteTaskSet」を参照してください。

希望タスク数

タスクのインスタンスをサービス内に配置し、実行し続ける数です。

サービスを一時的に停止したい場合は、この値を 0 に設定します。次に、サービスを開始する準備ができたら、サービスを元の値で更新します。

このパラメータは、ローリングデプロイとブルー/グリーンデプロイの場合変更できます。

マネージドタグの有効化

サービス内のタスクに Amazon ECS マネージドタグをオンにするか否かを決定します。

更新後に起動されたタスクのみに更新が反映されます。すべてのタスクのタグを更新する場合、強制デプロイオプションを使用します。

このパラメータは、ローリングデプロイとブルー/グリーンデプロイの場合変更できます。

ECS Exec の有効化

Amazon ECS Exec を使用するかどうかを決定します。

サービスの作成時に設定された値を上書きしたくない場合は、このアクションを実行するときにこれを null に設定できます。

このパラメータはローリングデプロイで変更できます。

ヘルスチェックの猶予期間

Amazon ECS サービススケジューラが、タスクが最初に開始された後で異常な Elastic Load Balancing、VPC Lattice、コンテナのヘルスチェックを無視する期間 (秒単位)。ヘルスチェックの猶予期間値を指定しない場合、デフォルト値 `0` が使用されます。ヘルスチェックを使用しない場合、`healthCheckGracePeriodSeconds` は使用されません。

サービスのタスクが開始してヘルスチェックに応答するまでに時間がかかる場合は、ヘルスチェックの猶予期間として最大 2,147,483,647 秒 (約 69 年) まで指定できます。この間は、Amazon ECS サービススケジューラはヘルスチェックのステータスを無視します。この猶予期間により、サービススケジューラがタスクを異常とマークして時間より前に停止することがなくなります。

このパラメータは、ローリングデプロイとブルー/グリーンデプロイの場合変更できます。

ロードバランサー

ロードバランサーを更新するとき、サービスにリンクされたロールを使用する必要があります。

Elastic Load Balancing ロードバランサーオブジェクトのリストです。これには、ロードバランサー名、コンテナ名、ロードバランサーからアクセスするコンテナポートが含まれます。コンテナ名は、コンテナの定義に表示されるものです。

Amazon ECS は、Elastic Load Balancing ロードバランサーまたは Amazon ECS コンテナインスタンスに関連付けられたセキュリティグループを自動的には更新しません。

ロードバランサー設定を追加、更新、削除すると、Amazon ECS は、更新された Elastic Load Balancing 設定で新しいタスクを開始し、その後新しいタスクを実行中に古いタスクを停止します。

ローリング更新を使用するサービスの場合、Elastic Load Balancing ターゲットグループを追加、更新、削除できます。1 つのターゲットグループから複数のターゲットグループに、および複数のターゲットグループから 1 つのターゲットグループに更新できます。

ブルー/グリーンデプロイを使用するサービスの場合、CodeDeploy を介して `CreateDeployment` を使用し、Elastic Load Balancing ターゲットグループを更新できます。ブ

ルー/グリーンデプロイでは、複数のターゲットグループがサポートされていないことに注意してください。詳細については、「Amazon ECS サービスに複数のターゲットグループを登録する」を参照してください。

外部デプロイコントローラーを使用するサービスの場合、CreateTaskSet を使用してロードバランサーを追加、更新、削除できます。外部デプロイでは、複数のターゲットグループがサポートされていないことに注意してください。詳細については、「Amazon ECS サービスに複数のターゲットグループを登録する」を参照してください。

空のリストを渡してロードバランサーを削除します。

このパラメータはローリングデプロイで変更できます。

ネットワーク設定

サービスのネットワーク設定です。

このパラメータはローリングデプロイで変更できます。

配置の制約事項

使用するサービスを更新するための、タスク配置制約オブジェクトの配列です。値を指定しないと、サービスの既存の配置制約は変更されません。この値を指定すると、サービスに定義されているすべての既存の配置制約が上書きされます。既存の配置制約をすべて削除するには、空の配列を指定します。

タスクごとに最大 10 個の制約を指定できます。この制限数には、タスク定義内の制約と、実行時に指定される制約が含まれます。

このパラメータは、ローリングデプロイとブルー/グリーンデプロイの場合変更できます。

配置戦略

使用するサービスを更新するためのタスク配置戦略オブジェクトです。値を指定しないと、サービスの既存の配置戦略は変更されません。この値を指定すると、サービスに定義されているすべての既存の配置戦略が上書きされます。既存の配置戦略を削除するには、空のオブジェクトを指定します。

このパラメータは、ローリングデプロイとブルー/グリーンデプロイの場合変更できます。

プラットフォームバージョン

サービスが実行される Fargate プラットフォームバージョンです。

Linux プラットフォームのバージョンを使用するサービスは、Windows プラットフォームのバージョンを更新できません。その逆も同様です。

このパラメータはローリングデプロイで変更できます。

タグの伝播

タグをタスク定義またはサービスからタスクへ伝播するかどうかを決定します。値を指定しない場合、タグは伝播されません。

更新後に起動されたタスクのみに更新が反映されます。すべてのタスクのタグを更新するには、`forceNewDeployment` を `true` に設定して、Amazon ECS が更新されたタグで新しいタスクを開始するようにします。

このパラメータは、ローリングデプロイとブルー/グリーンデプロイの場合変更できます。

Service Connect の設定

Amazon ECS Service Connect の設定です。このパラメータは、サービスがどのようにアプリケーション内の他のサービスに接続するかを決定します。

このパラメータはローリングデプロイで変更できます。

サービスレジストリ

サービスレジストリを更新するとき、サービスにリンクされたロールを使用する必要があります。

このサービスに割り当てるサービス検出レジストリの詳細。詳細については、「サービス検出」を参照してください。

サービスレジストリ設定を追加、更新、削除すると、Amazon ECS は更新されたサービスレジストリ設定で新しいタスクを開始し、その後新しいタスクの実行時に古いタスクを停止します。

空のリストを渡して、サービスレジストリを削除します。

このパラメータはローリングデプロイで変更できます。

タスク定義

サービス内のタスクに使用するタスクの定義とリビジョンです。

タスク定義でコンテナが使用するポートを変更する場合は、更新後のポートを使用するようにコンテナインスタンスのセキュリティグループを更新する必要がある場合があります。

サービスのタスク定義を更新する場合は、ロードバランサー設定で指定されたコンテナ名とコンテナポートはタスク定義のままにしておく必要があります。

コンテナイメージのプル動作は、コンピューティングオプションによって異なります。詳細については、以下のいずれかを参照してください。

- Amazon ECS の AWS Fargate 用のアーキテクト
- Amazon ECS 用の EC2 キャパシティのアーキテクト
- Amazon ECS の外部インスタンス (Amazon ECS Anywhere)

このパラメータはローリングデプロイで変更できます。

ボリューム設定

configuredAtLaunch だったボリュームの詳細です。タスク定義で configuredAtLaunch が true に設定されていると、デプロイ中に作成およびアタッチされるサービス内のタスクごとに、このサービスパラメータによって 1 つの Amazon EBS ボリュームが設定されます。ServiceManagedEBSVolumeConfiguration では、サイズ、volumeType、IOPS、スループット、スナップショット、暗号化を設定できます。ボリュームの name は、タスク定義の name と一致する必要があります。null に設定すると、新しいデプロイがトリガーされません。それ以外では、この設定が既存の設定と異なると、新しいデプロイがトリガーされます。

このパラメータはローリングデプロイで変更できます。

VPC Lattice の設定

サービスの VPC Lattice 設定です。サービス間の通信のためにサービスと VPC Lattice を統合する方法を定義します。

このパラメータはローリングデプロイで変更できます。

AWS CDK に関する考慮事項

AWS CDK はリソースの状態を追跡しません。サービスを作成しているのか、更新しているかはわかりません。お客様は、エスケープハッチを使用して ecs Service L1 コンストラクトに直接アクセスする必要があります。

エスケープハッチの詳細については、「AWS Cloud Development Kit (AWS CDK) v2 デベロッパーガイド」の「AWS コンストラクトライブラリからコンストラクトをカスタマイズする」を参照してください。

既存のサービスを ecs.Service コンストラクトに移行するには、次を実行します。

1. エスケープハッチを使用して、Service L1 コンストラクトにアクセスします。
2. Service L1 コンストラクトで次のプロパティを手動で設定します。

   サービスが Amazon EC2 容量を使用している場合:

   - daemon?
   - placementConstraints?
   - placementStrategies?
   - awsvpc ネットワークモードを使用する場合は、vpcSubnets? および securityGroups? コンストラクトを設定する必要があります。

   サービスが Fargate を使用している場合:

   - FargatePlatformVersion
   - vpcSubnets? と securityGroups? コンストラクト。
3. launchType を次のように設定します。

```
const cfnEcsService = service.node.findChild('Service') as ecs.CfnService;
cfnEcsService.launchType = "FARGATE";
```

起動タイプからキャパシティプロバイダーに移行するには、次を実行します。

1. エスケープハッチを使用して、Service L1 コンストラクトにアクセスします。
2. capacityProviderStrategies? コンストラクトを追加します。
3. サービスをデプロイします。

Amazon ECS サービスを更新する

サービスを作成した後、タスクの数など、サービスパラメータを更新する必要がある場合があります。

Amazon ECS サーキットブレーカーを使用するサービスを更新すると、Amazon ECS はサービスデプロイとサービスリビジョンを作成します。これらのリソースを使用すると、サービス履歴に関する詳細情報を表示できます。詳細については、「Amazon ECS サービスデプロイを使用してサービス履歴を表示する」を参照してください。

前提条件

サービスを更新する前に、デプロイタイプで変更可能なサービスパラメータを確認してください。変更可能なパラメータの詳細なリストについては、「Amazon ECS サービスのパラメータの更新」を参照してください。

手順

Console

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
2. [クラスター] ページで、クラスターを選択します。
3. クラスターの詳細ページの [サービス] セクションで、サービスの横にあるチェックボックスを選択し、[更新] を選択します。
4. サービスで新しいデプロイを開始するには、[新しいデプロイの強制] を選択します。
5. [タスク定義] の場合、タスク定義ファミリーとリビジョンを選択します。

   > **Important**
   >
   > コンソールは、選択したタスク定義ファミリーおよびリビジョンが、定義されたコンピューティング設定と互換性があることを確認します。警告が表示された場合は、タスク定義の互換性と、選択したコンピューティング設定の両方を確認します。

6. [Replica] (レプリカ) を使用した場合、[Desired tasks] (必要なタスク) に、サービス内で起動および維持するタスクの数を入力します。
7. [レプリカ] を選択した場合、Amazon ECS がアベイラビリティーゾーン間のタスクの分散をモニタリングし、不均衡が発生したときにタスクを再分散させるには、[アベイラビリティーゾーンのサービス再調整] で、[アベイラビリティーゾーンのサービス再調整] を選択します。
8. [Min running tasks] (実行中のタスクの最小化) の場合、デプロイ時に RUNNING の状態に保つ必要のあるサービス内のタスクの下限数をタスクの必要数のパーセント値 (最も近い整数に切り上げ) で入力します。詳細については、[Deployment configuration] (デプロイ設定) を参照してください。
9. [Max running tasks] (実行中のタスクの最大化)) には、デプロイ時に RUNNING または PENDING 状態にできるサービスのタスクの上限数を必要数のタスクのパーセント値 (最も近い整数に切り下げ) で入力します。
10. サービスにタスクがデプロイされる方法を設定するには、[デプロイオプション] を展開してオプションを設定します。

a. [デプロイコントローラータイプ] には、サービスデプロイのコントローラーを指定します。Amazon ECS コンソールは、ECS コントローラータイプをサポートします。

b. [デプロイ戦略] には、Amazon ECS がサービスの新しいバージョンをデプロイするために使用される戦略を選択します。

c. [デプロイ戦略] の選択に応じて、次の操作を行います。

デプロイ戦略	Steps
ローリング更新	a. [最小実行タスク %] には、サービスのデプロイ中に実行が必要なタスクの最小パーセント値を指定します。詳細については、「タスクを置き換えて Amazon ECS サービスをデプロイする」を参照してください。 b. [最大実行タスク %] には、サービスのデプロイ中に実行できるタスクの最大パーセント値を指定します。詳細については、「タスクを置き換えて Amazon ECS サービスをデプロイする」を参照してください。

デプロイ戦略	Steps	
Blue/Green	ベイク時間には、ブルーおよびグリーンのサービスリビジョンが同時に実行される時間を分単位で指定します。詳細については、「Amazon ECS ブルー/グリーンデプロイ」を参照してください。	

d. ライフサイクルステージに Lambda 関数を実行するには、[デプロイライフサイクルフック] で一意の Lambda 関数ごとに次の操作を行います。

i. [Add] (追加) を選択します。

実行する一意の関数ごとにこの操作を繰り返します。

ii. [Lambda 関数] には、関数名を入力します。

iii. [ロール] には、ブルー/グリーンのアクセス許可を持つ前提条件で作成したロールを選択します。

詳細については、「Amazon ECS ブルー/グリーンデプロイの Lambda 関数に必要となるアクセス許可」を参照してください。

iv. [ライフサイクルステージ] には、Lambda 関数が実行されるステージを選択します。

v. (オプション) [フックの詳細] には、フックに関する情報を提供するキーおよび値のペアを入力します。

11. [デプロイ設定] で、ターゲットサービスリビジョンが希望数に達する前にデプロイを完了するには、早期成功基準を設定します。Amazon ECS がソースサービスリビジョンのタスクをクリーンアップするタイミングを選択することもできます。

a. [早期成功基準] をオンにします。

b. [正常率] には、Amazon ECS がデプロイを完了する前に、ターゲットサービスリビジョンで実行され、正常である必要があるタスクの割合を入力します。

c. [ソースサービスリビジョンのクリーンアップ] で、[ブロッキング] または [延期] を選択します。

12. Amazon ECS がデプロイの障害を検出して処理する方法を設定するには、[Deployment failure detection] (デプロイ障害検出) を展開し、オプションを選択します。

    a. タスクを開始できない場合にデプロイを停止するには、[Use the Amazon ECS deployment circuit breaker] (Amazon ECS デプロイサーキットブレーカーを使用する) を選択します。

       デプロイサーキットブレーカーによってデプロイが失敗状態に設定されたときに、ソフトウェアがデプロイを最後に完了したデプロイ状態に自動的にロールバックするようにするには、[失敗時のロールバック] を選択します。

    b. アプリケーションメトリクスに基づいてデプロイを停止するには、[CloudWatch アラームを使用する] を選択します。次に、[CloudWatch アラーム名] からアラームを選択します。新しいアラームを作成するには、CloudWatch コンソールに移動します。

       CloudWatch アラームによってデプロイが失敗状態に設定されたときに、ソフトウェアがデプロイを最後に完了したデプロイ状態に自動的にロールバックするようにするには、[失敗時のロールバック] を選択します。

13. 計算オプションを変更するには、[コンピュート構成] を展開してから以下の操作を実行します。

    a. AWS Fargate のサービスにとって、[Platform version] (プラットフォームのバージョン) で、新しいバージョンを選択します。

    b. キャパシティプロバイダー戦略を使用するサービスの場合、[キャパシティプロバイダー戦略] で、次を実行します。

       - キャパシティプロバイダーをさらに追加するには、[さらに追加] を選択します。その後、[キャパシティプロバイダー] で、キャパシティプロバイダーを選択します。
       - キャパシティプロバイダーを削除するには、キャパシティプロバイダーの右側にある [削除] を選択します。

       Auto Scaling グループキャパシティープロバイダーを使用するサービスは、Fargate キャパシティープロバイダーを使用するように更新することはできません。Fargate キャパシティプロバイダーを使用するサービスは、Auto Scaling グループキャパシティプロバイダーを使用するように更新できません。

14. (オプション) サービスの自動スケーリングを設定するには、[サービスの自動スケーリング] を展開し、次のパラメータを指定します。トラフィックフローからの過去のロードデータを調べる予測自動スケーリングを使用するには、サービスを作成した後にそれを設定します。

詳細については、「履歴パターンを使用して予測スケーリングで Amazon ECS サービスをスケールする」を参照してください。

a. サービスの自動スケーリングを使用するには、[Service auto scaling] (サービスの自動スケーリング) を選択します。

b. [タスクの最小数] に、サービスの自動スケーリングで使用するタスクの下限数を入力します。必要な数がこの数を下回ることはありません。

c. [タスクの最大数] に、サービスの自動スケーリングで使用するタスクの上限数を入力します。必要な数がこの数を超えることはありません。

d. ポリシータイプを選択します。[スケーリングポリシータイプ] で、次のいずれかのオプションを選択します。

使用するポリシータイプ	この操作を行います
ターゲット追跡	a. [スケーリングポリシータイプ] で [ターゲットの追跡] を選択します。 b. [Policy Name] (ポリシー名) にこのポリシーの名前を入力します。 c. [ECS サービスメトリクス] で次のいずれかのメトリクスを選択します。 • [ECSServiceAverageCPUUtilization] — サービスの平均 CPU 使用率。 • [ECSServiceAverageMemoryUtilization] — サービスのメモリ平均使用率。 • [ALBRequestCountPerTarget] - Application Load Balancer ターゲットグループ内のターゲットごとに完了したリクエストの数。 d. [Target value] (ターゲット値) には、選択したメトリックに対してサービスが保持する値を入力します。 e. [スケールアウトのクールダウン期間] には、ス

使用するポリシータイプ	この操作を行います	
	ケールインアクティビティが完了してから別のスケールアウトアクティビティが開始されるまでの時間 (秒) を入力します。 f. [スケールインのクールダウン期間] には、スケールインアクティビティが完了してから別のスケールインアクティビティが開始されるまでの時間 (秒) を入力します。 g. ポリシーがスケールインアクティビティを実行しないようにするには、[Turn off scale-in] (スケールインをオフにする) を選択します。 h. • (オプション) トラフィックの増加に合わせてスケーリングポリシーをスケールアウトさせたいが、トラフィックが減少してもスケールインする必要がない場合は、[スケールインをオフにする] を選択します。	

使用するポリシータイプ	この操作を行います	
ステップスケーリング	a. [スケーリングポリシータイプ] で [ステップスケーリング] を選択します。 b. [ポリシー名] で、このポリシー名を入力します。 c. [アラーム名] に、アラームの一意の名前を入力します。 d. [Amazon ECS サービスメトリクス] で、アラームに使用するサービスメトリクスを選択します。 e. [統計] で、アラーム統計を選択します。 f. [期間] で、アラームの期間を選択します。 g. [アラーム条件] で、選択したメトリクスを定義されたしきい値と比較する方法を選択します。 h. [メトリクスを比較するためのしきい値] と [アラームを開始する評価期間] に、アラームに使用するしきい値としきい値を評価する期間を入力します。	

使用するポリシータイプ	この操作を行います	
	i. [スケーリングアクション]で、次の手順を実行します。 • [アクション] には、サービスに対してタスクを追加する、削除する、または特定の必要数を設定するかを選択します。 • タスクの追加または削除を選択した場合は、スケーリングアクションの開始時に追加または削除するタスクの数 (または既存のタスクの割合) を [値] に入力します。希望の数を設定することを選択した場合は、タスクの数を入力します。[タイプ] で、[値] が整数であるか、既存の必要なカウントのパーセント値であるかを選択します。 • [下限] と [上限] に、ステップスケーリング調整の下限と上限を入力します。デフォルトでは、[ポリシーを追加] の下限はアラームしきい値であり、上限は正 (+)	

使用するポリシータイプ	この操作を行います	
	の無限大です。デフォルトでは、[ポリシーを削除] の上限はアラームしきい値であり、下限は負 (-) の無限大です。 • (オプション) さらにスケーリングオプションを追加します。[新しいスケーリングアクションの追加] を選択し、[アクションのスケーリング] の手順を繰り返します。 • [クールダウン期間] には、前回のスケーリングアクティビティが有効になるまで待機する時間を秒単位で入力します。追加ポリシーの場合、スケールアウトアクティビティの終了後、スケーリングポリシーによってスケールインアクティビティがブロックされ、一度にスケールアウトできるタスクの数が制限されます。削除ポリシーの場合、これはスケールインアクティビティの後	

使用するポリシータイプ	この操作を行います	
	、別のスケールインアクティビティが開始されるまでに経過する必要がある時間です。	

15. (オプション) Service Connect を使用するには、[Turn on Service Connect] (Service Connect をオンにする) を選択し、以下を指定します。

    a. [Service Connect configuration] (Service Connect 設定) で、クライアントモードを指定します。

    - サービスが名前空間内の他のサービスへの接続のみを必要とするネットワーククライアントアプリケーションを実行している場合は、[クライアント側のみ] を選択します。
    - サービスがネットワークまたは Web サービスアプリケーションを実行していて、このサービスにエンドポイントを提供し、名前空間内の他のサービスに接続する必要がある場合は、[Client and server] (クライアントとサーバー) を選択します。

    b. デフォルトのクラスター名前空間ではない名前空間を使用するには、[Namespace] (名前空間) でサービス名前空間を選択します。この場合、AWS アカウントの同じ AWS リージョンで個別に作成された名前空間、またはAWS Resource Access Manager (AWS RAM) を使用してアカウントと共有されている同じリージョンの名前空間のいずれかを選択できます。共有 AWS Cloud Map 名前空間の詳細については、「AWS Cloud Map デベロッパーガイド」の「クロスアカウント AWS Cloud Map 名前空間の共有」を参照してください。

    c. (オプション) ログ設定を指定します。[ログコレクションを使用] を選択します。デフォルトのオプションでは、コンテナログを CloudWatch Logs に送信します。その他のログドライバオプションは、AWS FireLens を使用して構成されます。詳細については、「Amazon ECS ログを AWS サービスまたは AWS Partner に送信する」を参照してください。

    以下では、各コンテナログの送信先について詳しく説明します。

    - Amazon CloudWatch — コンテナログを CloudWatch Logs に送信するようにタスクを設定します。デフォルトのログドライバーオプションが提供され、ユーザーに代わり

CloudWatch ロググループを作成します。別のロググループ名を指定するには、ドライバーオプションの値を変更します。

- [Amazon Data Firehose] — Firehose にコンテナログを送信するようタスクを設定します。Firehose 配信ストリームにログを送信するデフォルトのログドライバーオプションが提供されています。別の配信ストリーム名を指定するには、ドライバーオプションの値を変更します。
- Amazon Kinesis Data Streams — Kinesis Data Streams にコンテナログを送信するようタスクを設定します。Kinesis Data Streams のストリームにログを送信するデフォルトのログドライバーオプションが提供されています。別のストリーム名を指定するには、ドライバーオプションの値を変更します。
- Amazon OpenSearch Service — コンテナログを OpenSearch Service ドメインに送信するようタスクを設定します。ログドライバーオプションを提供する必要があります。
- Amazon S3 — Amazon S3 バケットにコンテナログを送信するようタスクを設定します。デフォルトのログドライバーオプションはデフォルトで設定されていますが、有効な Amazon S3 バケット名を指定する必要があります。

d. アクセスログを有効にするには、次の手順に従います。

i. [アクセスログ設定] を展開します。[形式] で、[JSON] または [TEXT] を選択します。

ii. アクセスログにクエリパラメータが含まれるようにするには、[クエリパラメータを含める] を選択します。

> **Note**
>
> アクセスログを無効にするには、[フォーマット] で [なし] を選択します。

16. デプロイ時の設定と互換性のあるデータボリュームをタスクで使用する場合は、[ボリューム] を拡張してボリュームを設定できます。

ボリューム名とボリュームタイプはタスク定義リビジョンを作成するときに設定され、サービスの更新時には変更できません。ボリューム名とタイプを更新するには、新しいタスク定義リビジョンを作成し、新しいリビジョンを使用してサービスを更新する必要があります。

このボリュームタイプを設定するには	この操作を行います
Amazon EBS	a. [EBS ボリュームタイプ] には、タスクにアタッチする EBS ボリュームのタイプを選択します。 b. [サイズ (GiB)] には、ボリュームサイズの有効な値をギビバイト (GiB) 単位で入力します。最小で 1 GiB、最大で 16,384 GiB のボリュームサイズを指定できます。スナップショット ID を指定しない限り、この値は必須です。 c. [IOPS] には、ボリュームが提供する入力/出力オペレーションの数 (IOPS) の最大数を入力します。この値は `io1`、`io2`、および `gp3` ボリュームタイプでのみ設定できます。 d. [スループット (MiB/秒)] には、ボリュームが提供すべきスループットをメビバイト/秒 (MiBps または MiB/s) で入力します。この値は `gp3` ボリュームタイプでのみ設定できます。 e. [スナップショット ID] には、既存の Amazon

このボリュームタイプを設定するには	この操作を行います	
	EBS ボリュームスナップショットを選択するか、スナップショットからボリュームを作成する場合はスナップショットの ARN を入力します。新しい空のボリュームは、スナップショット ID を選択または入力しないで作成することもできます。 f. スナップショット ID を指定すると、ボリューム初期化レート (MiB/秒) を指定できます。100 から 300 の値 (MiB/秒単位) を入力して、スナップショット ID を使用してボリュームを作成する際のスナップショットからのデータ読込み速度を指定します。 g. [ファイルシステムのタイプ] では、ボリューム上のデータストレージおよび取得に使用するファイルシステムの種類を選択します。オペレーティングシステムのデフォルトまたは特定のファイルシステムタイプのいずれかを選択できます。Linux のデフォルトは XFS で	

このボリュームタイプを設定するには	この操作を行います	
	す。スナップショットから作成されたボリュームでは、スナップショットの作成時にボリュームが使用していたのと同じファイルシステムタイプを指定する必要があります。ファイルシステムのタイプが一致しない場合、タスクは開始できません。 h. [インフラストラクチャロール] には、Amazon ECS がタスクの Amazon EBS ボリュームを管理できるようにするために必要な権限を持つ IAM ロールを選択します。`AmazonECSInfrastructureRolePolicyForVolumes` 管理ポリシーをロールにアタッチすることも、このポリシーをガイドとして使用して、特定のニーズを満たすアクセス権限を持つ独自のポリシーを作成してアタッチすることもできます。必要なアクセス権限の詳細については、「Amazon ECS インフラストラクチャ IAM	

このボリュームタイプを設定するには	この操作を行います	
	ロール」を参照してください。 i. デフォルト設定で Amazon EBS 暗号化を使用する場合は、[暗号化] で [デフォルト] を選択します。アカウントにデフォルトで暗号化が設定されている場合、ボリュームは設定で指定されている AWS Key Management Service (AWS KMS) キーで暗号化されます。[デフォルト] を選択し、Amazon EBS のデフォルト暗号化がオンになっていない場合、ボリュームは暗号化されません。 [カスタム] を選択した場合は、ボリューム暗号化に任意の AWS KMS key を指定できます。 [なし] を選択すると、デフォルトで暗号化が設定されている場合や、暗号化されたスナップショットからボリュームを作成した場合を除き、ボリュームは暗号化されません。	

このボリュームタイプを設定するには	この操作を行います	
	j. [暗号化] に [カスタム] を選択した場合は、使用する AWS KMS key を指定する必要があります。[KMS キー] には、AWS KMS key を選択するか、キーの ARN を入力します。対称型のカスタマー管理キーを使用してボリュームを暗号化する場合は、AWS KMS key ポリシーに適切な権限が定義されていることを確認してください。詳細については、「Amazon EBS ボリュームのデータ暗号化」を参照してください。 k. (オプション) [タグ] では、タスク定義またはサービスからタグを伝達するか、独自のタグを指定することで、Amazon EBS ボリュームにタグを追加できます。 タスク定義からタグを伝達する場合は、[タグの伝播元] で [タスク定義] を選択します。サービスからタグを伝達する場合は、[タグの伝播元] で [サービス] を選択します。[伝播しない] を選択	

このボリュームタイプを設定するには	この操作を行います	
	した場合、または値を選択しなかった場合、タグは伝播されません。 独自のタグを指定する場合は、[タグの追加] を選択し、追加する各タグのキーと値を指定します。 Amazon EBS ボリュームのタグ付けの詳細については、「Amazon EBS ボリュームのタグ付け」を参照してください。	

17. (オプション) サービスを識別しやすくするには、[Tags] (タグ) セクションを展開し、タグを設定します。
    - [タグの追加] [タグの追加] を選択して、以下を実行します。
        - [キー] にはキー名を入力します。
        - [値] にキー値を入力します。
    - [タグの削除] タグの横にある [タグの削除] を選択します。
18. [更新] を選択します。

AWS CLI

- `update-service` を実行します。コマンドの実行の詳細ついては、「AWS Command Line Interfaceリファレンス」の「update-service」を参照してください。

  次の `update-service` の例では、`my-http-service` サービスのタスクの数を 2 に更新します。

  `user-input` を独自の値に置き換えます。

```
aws ecs update-service \
```

```
    --cluster MyCluster \
    --service my-http-service \
    --desired-count 2
```

次のステップ

Amazon ECS サーキットブレーカーのサービスのデプロイを追跡し、サービス履歴を表示します。詳細については、「Amazon ECS サービスデプロイを使用してサービス履歴を表示する」を参照してください。

キャパシティプロバイダーを使用するように Amazon ECS サービスを更新する

Amazon EC2 または Fargate 起動タイプを使用する既存のサービスがあり、Amazon ECS マネージドインスタンスを使用する必要がある場合は、Amazon ECS マネージドインスタンスキャパシティプロバイダーを使用するようにサービスを更新する必要があります。

前提条件

Amazon ECS マネージドインスタンスのキャパシティプロバイダーを作成します。詳細については、「Amazon ECS マネージドインスタンスのキャパシティプロバイダーを作成する」を参照してください。

手順

Console

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
2. [クラスター] ページで、クラスターを選択します。
3. クラスターの詳細ページの [サービス] セクションで、サービスの横にあるチェックボックスを選択し、[更新] を選択します。
4. [新しいデプロイを強制する] を選択します。
5. [コンピューティングオプション] で、[キャパシティプロバイダー戦略] を選択します。続いて、次のいずれかを選択します。

    - Amazon ECS マネージドインスタンスキャパシティプロバイダーがデフォルトのキャパシティプロバイダーである場合は、[クラスターのデフォルトを使用する] を選択します。
    - Amazon ECS マネージドインスタンスキャパシティプロバイダーがデフォルトのキャパシティプロバイダーでない場合は、[カスタム (アドバンスト) を使用] を選択しま

す。Amazon ECS マネージドインスタンスのキャパシティプロバイダーを選択し、[重み] で [1] を選択します。

6. [更新] を選択します。

AWS CLI

- `update-service` を実行します。コマンドの実行の詳細ついては、「AWS Command Line Interfaceリファレンス」の「update-service」を参照してください。

  `user-input` を独自の値に置き換えます。

```
aws ecs update-service \
    --cluster my-cluster \
    --service my-service \
    --capacity-provider-strategy capacityProvider=my-managed-instance-capacity-provider,weight=1 \
    --force-new-deployment
```

コンソールを使用して Amazon ECS サービスの削除

サービスを削除する理由としては、次のようなものがあります。

- アプリケーションが不要になった
- サービスを新しい環境に移行する
- アプリケーションがアクティブに使用されていない
- アプリケーションが必要以上のリソースを使用しているため、コストを最適化しようとしている

削除する前に、このサービスは自動的に 0 までスケールダウンされます。サービスに関連付けられているロードバランサーリソース、またはサービス検出リソースは、サービスの削除による影響を受けません。Elastic Load Balancing リソースを削除するには、ロードバランサーのタイプに応じて、次のいずれかのトピックを参照してください。「Application Load Balancer の削除」または「Network Load Balancer の削除」。

サービスを削除すると、Amazon ECS はそのサービスのすべてのサービスデプロイとサービスリビジョンを削除します。

サービスを削除するときに、クリーンアップが必要なタスクがまだ実行中の場合、サービスは ACTIVE から DRAINING ステータスに移行し、そのサービスステータスはコンソールまたは ListServices API オペレーションで表示されなくなります。すべてのタスクが STOPPING または STOPPED ステータスに以降されたら、サービスステータスは DRAINING から INACTIVE になります。DRAINING または INACTIVE ステータスのサービスは、DescribeServices API オペレーションで引き続き表示できます。

> **Important**
>
> ACTIVE または DRAINING ステータスの既存のサービスと同じ名前で新しいサービスを作成しようとすると、エラーが表示されます。

手順

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. [Clusters] (クラスター) ページで、サービスに使用するクラスターを選択します。
3. [Clusters] (クラスター) ページで、クラスターを選択します。
4. [Cluster: ***name***] (クラスター: 名前) ページで、[Services] (サービス) タブを選択します。
5. サービスを選択して、[Delete] (削除) を選択します。
6. タスクがゼロになっていなくてもサービスを削除するには、[Force delete service] (サービスの強制削除) を選択します。
7. 確認プロンプトで、delete と入力し、[削除] を選択します。

Amazon ECS の短いサービス ARN を長い ARN に移行する

Amazon ECS は、各サービスに一意の Amazon リソースネーム (ARN) を割り当てます。2021 年以前に作成されたサービスは、短いARN 形式を使用しています。

`arn:aws:ecs:`*`region`*`:`*`aws_account_id`*`:service/`*`service-name`*

Amazon ECS では、クラスター名を含めるように ARN 形式を変更しました。これは長い ARN 形式です。

`arn:aws:ecs:`*`region`*`:`*`aws_account_id`*`:service/`*`cluster-name`*`/`*`service-name`*

サービスにタグを付けるには、長いARN形式を使用する必要があります。

サービスを再作成する必要なく、短い ARN 形式のサービスを長い ARN 形式のサービスへと移行できます。API、CLI、またはコンソールを使用できます。移行オペレーションを元に戻すことはできません。

移行プロセスはシームレスで、サービスのダウンタイムゼロを確保できます。移行中の詳細

- サービスの可用性: トラフィックや機能を中断することなく、サービスを正常に実行し続けます。
- タスクの実行: 既存のタスクは中断されることなく、実行し続けます。`taskLongArnFormat` アカウント設定が有効になっている場合、移行後に起動される新しいタスクには長い ARN 形式が使用されます。
- コンテナインスタンス: コンテナインスタンスはサービス ARN の移行の影響を受けることはなく、正常に動作し続けます。
- サービス設定: すべてのサービス設定が変更されません (タスク定義、ネットワーク、ロードバランサー設定など)。

CloudFormation を使用して短い ARN 形式でサービスにタグ付けする場合は、API、CLI、またはコンソールを使用してサービスを移行する必要があります。移行が完了したら、 CloudFormation を使用してサービスにタグを付けることができます。

Terraform を使用して短い ARN 形式でサービスにタグ付けする場合は、API、CLI、またはコンソールを使用してサービスを移行する必要があります。移行が完了したら、Terraform を使用してサービスにタグを付けることができます。

移行が完了すると、サービスには次の変更が行われます。

- 長い ARN 形式

  `arn:aws:ecs:`*`region`*`:`*`aws_account_id`*`:service/`*`cluster-name`*`/`*`service-name`*
- コンソールを使用して移行する場合、Amazon ECS はキーを「ecs:serviceArnMigratedAt」に設定し、値を移行タイムスタンプ (UTC 形式) に設定して、サービスにタグを追加します。

  このタグは、タグクォータにカウントされます。
- CloudFormation スタック内の `PhysicalResourceId` がサービス ARN を表す場合、値は変更されず、引き続き短いサービス ARN になります。

前提条件

サービス ARN を移行する前に、次のオペレーションを実行します。

1. 短い ARN 形式 があるかどうかを確認するには、Amazon ECS コンソールでサービスの詳細を表示するか (サービスが短い ARN 形式である場合は警告が表示されます)、または describe-services の serviceARN 戻りパラメータを表示します。ARN にクラスター名が含まれていない場合、短い ARN になります。短い ARN の形式は次のとおりです。

   arn:aws:ecs:*region*:*aws_account_id*:service/*service-name*

2. 作成日をメモします。
3. 短い ARN 形式を使用する IAM ポリシーがある場合は、長い ARN 形式に更新します。

   各*##############*を独自の情報に置き換えます。

   arn:aws:ecs:*region*:*aws_account_id*:service/*cluster-name*/*service-name*

   詳細については、「AWS Identity and Access Management ユーザーガイド」の「Editing IAM policies」(IAM ポリシーの編集) を参照してください。
4. 短い ARN 形式を使用するツールがある場合は、長い ARN 形式に更新してください。

   各*##############*を独自の情報に置き換えます。

   arn:aws:ecs:*region*:*aws_account_id*:service/*cluster-name*/*service-name*

5. サービスの長い ARN 形式を有効にします。serviceLongArnFormat オプションを enabled に設定して put-account-setting を実行します。詳細については、「Amazon Elastic Container Service API リファレンス」の「put-account-setting」を参照してください。

   サービスの createdAt 日付が不明な場合は、ルートユーザーとしてコマンドを実行します。

   ```
   aws ecs put-account-setting --name serviceLongArnFormat --value enabled
   ```

   出力の例

   ```
   {
       "setting": {
           "name": "serviceLongArnFormat",
           "value": "enabled",
           "principalArn": "arn:aws:iam::123456789012:role/your-role",
           "type": user
       }
   }
   ```

6. タスクの長い ARN 形式を有効にします。このアカウント設定により、サービス移行の完了後に起動される新しいタスクの ARN 形式が制御されます。`taskLongArnFormat` オプションを `enabled` に設定して `put-account-setting` を実行します。詳細については、「Amazon Elastic Container Service API リファレンス」の「put-account-setting」を参照してください。

   サービスの `createdAt` 日付が不明な場合は、ルートユーザーとしてコマンドを実行します。

   ```
   aws ecs put-account-setting --name taskLongArnFormat --value enabled
   ```

   出力の例

   ```
   {
       "setting": {
           "name": "taskLongArnFormat",
           "value": "enabled",
           "principalArn": "arn:aws:iam::123456789012:role/your-role",
           "type": user
       }
   }
   ```

   > **Note**
   >
   > `taskLongArnFormat` 設定では、既存のタスクは直接移行されません。設定が有効になった後に作成される新しいタスクの ARN 形式のみに影響を与えます。通常のサービスオペレーション (デプロイやスケーリングアクティビティなど) によって置き換えられるまで、既存の実行中のタスクは現在の ARN 形式を保持します。

手順

サービス ARN を移行するには、以下を使用します。

コンソール

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
2. [クラスター] ページで、クラスターを選択します。
3. [サービス] セクションで、ARN 列に警告があるサービスを選択します。

   [サービス詳細] ページが表示されます。

4. [長い ARN に移行する] を選択します。

   サービス移行ダイアログボックスが表示されます。

5. 移行 を選択します。

CLI

前提条件を完了したら、サービスにタグを付けることができます。次のコマンドを実行してください:

Amazon ECS は、短いARNを持つサービスに対する tag-resource APIリクエストで長い ARN 形式を渡すことを、そのサービスを長い ARN 形式を使用するよう移行するシグナルとみなします。

```
aws ecs tag-resource \
    --resource-arn arn:aws:ecs:region:aws_account_id:service/cluster-name/service-name
    --tags key=key1,value=value1
```

次の例では、キーが「TestService」に設定され、値が「WebServers」に設定されたタグで MyService にタグを付けます。

```
aws ecs tag-resource \
    --resource-arn arn:aws:ecs:us-east-1:123456789012:service/MyCluster/MyService
    --tags key=TestService1,value=WebServers
```

Terraform

前提条件を完了したら、サービスにタグを付けることができます。aws_ecs_service リソースを作成し、tags リファレンスを設定します。詳細については、Terraform ドキュメントの「Resource: aws_ecs_service」を参照してください。

```
resource "aws_ecs_service" "MyService" {
  name    = "example"
  cluster = aws_ecs_cluster.MyService.id

 tags = {
 "Name"  =  "MyService"
 "Environment"  =  "Production"
 "Department"  =  "QualityAssurance"
  }
}
```

次のステップ

サービスにタグを追加できます。詳細については、「Amazon ECS リソースにタグを追加する」を参照してください。

Amazon ECS でタスク定義またはサービスからタスクにタグを伝達する場合は、propagateTags パラメータを指定して update-service を実行します。詳細については、「 AWS Command Line Interface リファレンス」の「create-service」を参照してください。

トラブルシューティング

一部のユーザーの場合、短い ARN 形式から長い ARN 形式に移行すると以下のエラーが発生することがあります。

```
There was an error while migrating the ARN of service service-name. The
specified account does not have serviceLongArnFormat or taskLongArnFormat
account settings enabled. Add account settings in order to enable tagging.
```

serviceLongArnFormat アカウント設定を既に有効にしていてもこのエラーが発生する場合は、長い ARN 形式のアカウント設定が、最初にサービスを作成した特定の IAM プリンシパルに対して有効になっていない可能性があります。

1. サービスを作成したプリンシパルを特定します。

   1. コンソールでは、Amazon ECS コンソールのサービスの詳細ページの [設定とネットワーク] タブの [作成者] フィールドで情報を確認できます。
   2. AWS CLI の場合、次のコマンドを実行します。

      user-input を独自の値に置き換えます。

      ```
      aws ecs describe-services --cluster cluster-name --services service-name --query
       'services[0].{createdBy: createdBy}'
      ```

2. その特定のプリンシパルに必要なアカウント設定を有効にします。これは、次のいずれかの方法で行うことができます。

   a. そのプリンシパルの IAM ユーザーまたはロールを引き受けます。次に、put-account-setting を実行します。

   b. principal-arn で作成プリンシパルを指定しながら、ルートユーザーを使用してコマンドを実行します。

例。

[principal-arn] をステップ 1 で確認した値に置き換えます。

```
aws ecs put-account-setting --name serviceLongArnFormat --value enabled --
principal-arn arn:aws:iam::123456789012:role/jdoe
```

どちらの方法でも、サービスを作成したプリンシパルに必要な serviceLongArnFormat アカウント設定が有効になり、ARN 移行を続行できます。

Amazon ECS サービスのスロットルロジック

Amazon ECS サービススケジューラには、タスクが繰り返し起動に失敗した場合にタスク起動をスロットリングする保護ロジックが含まれています。これにより、不要なリソースの消費が防止され、コストを削減できるようになります。

サービス内のタスクが PENDING 状態から RUNNING 状態への移行に失敗し、代わりに STOPPED へと直接移行すると、スケジューラは次のようになります。

- 再起動の試行間隔を段階的に増やす
- 試行間隔の遅延を最大 27 分まで増加し続ける
- 問題を通知するサービスイベントメッセージを生成する

> **Note**
>
> 最大遅延時間の 27 分は、今後の更新で変更される場合があります。

スロットリングがアクティブ化されると、次のサービスイベントメッセージが表示されます。

```
(service service-name) is unable to consistently start tasks successfully.
```

スロットリングロジックの重要な特性

- サービスによって再試行が無期限に継続される
- 唯一の変更は、再起動間の時間が増加されること
- ユーザーが設定可能なパラメータがない

スロットリング問題の解決

スロットリングを解決するには、次の操作を行います。

- 新しいタスク定義を使用するようにサービスを更新すると、サービスは即時に通常のスロットリングされていないオペレーションに戻ります。詳細については、「Amazon ECS サービスを更新する」を参照してください。
- タスク失敗の根本的な原因に対処します。

スロットリングをトリガーするタスク障害によくある原因は、次のとおりです。

- 不十分なクラスターリソース (ポート、メモリ、CPU)
  - リソースサービスのイベントメッセージが不十分と表示されます
- コンテナイメージのプル失敗
  - 無効なイメージ名、タグ、不十分なアクセス許可が原因で発生する可能性があります
  - Amazon ECS の停止したタスクのエラーを表示する の `CannotPullContainerError` が発生する
- ディスク容量の不足
  - タスク停止エラーの `CannotCreateContainerError` が発生する
  - 解決手順については、「Amazon ECS の Docker API `error (500): devmapper` のトラブルシューティング」を参照してください。

> **Important**
>
> 次のシナリオでは、スロットルロジックはトリガーされません。
>
> - `RUNNING` 状態に達した後に停止するタスク
> - Elastic Load Balancing のヘルスチェックの失敗により、タスクが停止しました
> - `RUNNING` 状態に達した後に、コンテナコマンドがゼロ以外のコードで終了するタスク

Amazon ECS サービス定義パラメータ

サービス定義は、Amazon ECS サービスの実行方法を定義します。サービス定義では、次のパラメータを指定できます。

起動タイプ

launchType

タイプ: 文字列

有効な値: EC2 | FARGATE | EXTERNAL

必須: いいえ

サービスを実行する起動タイプ。起動タイプを指定しない場合は、デフォルトで capacityProviderStrategy が使用されます。

サービスを更新すると、このパラメータは新しいサービスのデプロイをトリガーします。

launchType を指定した場合、capacityProviderStrategy パラメータを省略する必要があります。

キャパシティープロバイダー戦略

capacityProviderStrategy

タイプ: オブジェクトの配列

必須: いいえ

サービスに使用するキャパシティープロバイダー戦略。

サービスを更新しても、このパラメータは新しいサービスのデプロイをトリガーしません。

キャパシティープロバイダー戦略は、1 つ以上のキャパシティープロバイダーと、それらに割り当てる base と weight で構成されます。キャパシティープロバイダーは、キャパシティープロバイダー戦略で使用するクラスターに関連付ける必要があります。PutClusterCapacityProviders API は、キャパシティープロバイダーをクラスターに関連付けるために使用されます。ACTIVE または UPDATING ステータスのキャパシティープロバイダーのみを使用できます。

capacityProviderStrategy を指定した場合、launchType パラメータを省略する必要があります。capacityProviderStrategy または launchType を指定しない場合は、クラスターの defaultCapacityProviderStrategy が使用されます。

Auto Scaling グループを使用するキャパシティープロバイダーを指定する場合は、キャパシティープロバイダーが既に作成されている必要があります。新しいキャパシティープロバイダーは、CreateCapacityProvider API オペレーションで作成できます。

AWS Fargate キャパシティープロバイダーを使用するには、FARGATE または FARGATE_SPOT キャパシティープロバイダーを指定します。AWS Fargate キャパシティープロバイダーはすべてのアカウントで使用でき、クラスターに関連付けるだけで使用できるようになります。

PutClusterCapacityProviders API オペレーションは、クラスターの作成後にクラスターで使用可能なキャパシティープロバイダーのリストを更新するために使用されます。

`capacityProvider`

タイプ: 文字列

必須: はい

キャパシティープロバイダーの短縮名または完全な Amazon リソースネーム (ARN)。

`weight`

タイプ: 整数

有効な範囲: 0 ~ 1,000 の整数。

必須: いいえ

ウエイト値は、指定したキャパシティープロバイダーを使用する起動済みタスクの総数に対する相対的な割合を示します。

例えば、2 つのキャパシティープロバイダーを含む戦略があり、両方の重みが 1 であるとします。ベースが満たされると、タスクは 2 つのキャパシティープロバイダー間で均等に分割されます。同じロジックを使用して、capacityProviderA に 1 の重みを指定し、capacityProviderB に 4 の重みを指定するとします。その後、capacityProviderA を使用して実行される 1 つのタスクごとに、4 つのタスクが capacityProviderB を使用します。

`base`

タイプ: 整数

有効な範囲: 0 ~ 100,000 の整数。

必須: いいえ

ベース値は、指定されたキャパシティープロバイダーで実行するタスクの最小限の数を指定します。キャパシティープロバイダー戦略では、ベースを定義できるキャパシティープロバイダーは 1 つだけです。

タスク定義

`taskDefinition`

タイプ: 文字列

必須: いいえ

`family` と `revision` (`family:revision`)、またはサービスで実行されるタスク定義の完全な Amazon リソースネーム (ARN) 。`revision` を指定しない場合、指定したファミリーの最新の `ACTIVE` リビジョンが使用されます。

サービスを更新すると、このパラメータは新しいサービスのデプロイをトリガーします。

ローリングアップデート (ECS) デプロイメントコントローラーを使用する場合は、タスク定義を指定する必要があります。

プラットフォームオペレーティングシステム

`platformFamily`

タイプ: 文字列

必須: 条件による

デフォルト: Linux

このパラメータは Fargate でホストされている Amazon ECS サービスに必要です。

このパラメータは、Amazon EC2 でホストされている Amazon ECS サービスでは無視されます。

サービスを実行するコンテナ上のオペレーティングシステム。有効な値は、`LINUX`、`WINDOWS_SERVER_2019_FULL`、`WINDOWS_SERVER_2019_CORE`、`WINDOWS_SERVER_2` および `WINDOWS_SERVER_2022_CORE` です。

サービスに指定するすべてのタスクの platformFamily 値は、サービス platformFamily 値と一致する必要があります。例えば、platformFamily を WINDOWS_SERVER_2019_FULL に設定すると、すべてのタスクの platformFamily 値は WINDOWS_SERVER_2019_FULL でなければなりません。

プラットフォームバージョン

`platformVersion`

タイプ: 文字列

必須: いいえ

サービス内のタスクが実行されているプラットフォームのバージョン。プラットフォームのバージョンは、Fargate 起動タイプを使用するタスクに対してのみ指定されています。指定されない場合、デフォルトで最新バージョン (LATEST) が使用されます。

サービスを更新すると、このパラメータは新しいサービスのデプロイをトリガーします。

AWS Fargate プラットフォームのバージョンを使って、Fargate タスクインフラストラクチャの特定のランタイム環境を参照できます。プラットフォームのバージョンに LATEST を指定してタスクを実行またはサービスを作成すると、プラットフォームの最新バージョンをタスクで利用できるようになります。サービスをスケールアップする場合は、これらのタスクには、サービスの最新のデプロイで指定されたプラットフォームのバージョンが提供されます。詳細については、「Amazon ECS 向け Fargate プラットフォームバージョン」を参照してください。

> **Note**
> プラットフォームのバージョンは、EC2 起動タイプを使用するタスクには指定されません。

クラスター

`cluster`

タイプ: 文字列

必須: いいえ

サービスを実行するクラスターの短い名前または完全な Amazon リソースネーム (ARN)。クラスターを指定しない場合は、default クラスターが使用されます。

サービス名

serviceName

タイプ: 文字列

必須: はい

サービスの名前。最大 255 文字の英字 (大文字と小文字)、数字、ハイフン、アンダースコアを使用できます。サービス名は同じクラスター内で一意になるようにしてください。ただし、リージョン内の複数のクラスター間や複数のリージョンにまたがるクラスター間では、同様の名前のサービスがあっても構いません。

スケジュール戦略

schedulingStrategy

タイプ: 文字列

有効な値: REPLICA | DAEMON

必須: いいえ

使用するスケジュール戦略。スケジュール戦略が指定されていない場合は、REPLICA 戦略が使用されます。詳細については、「Amazon ECS サービス」を参照してください。

利用できる 2 つのサービススケジューラ戦略があります。

- REPLICA - レプリカスケジュール戦略では、クラスター全体で必要数のタスクを配置して維持します。デフォルトでは、サービススケジューラによってタスクはアベイラビリティーゾーン間に分散されます。タスク配置の戦略と制約を使用すると、タスク配置の決定をカスタマイズできます。詳細については、「レプリカスケジューリング戦略」を参照してください。
- DAEMON - デーモンのスケジュール戦略では、指定したすべてのタスク配置制約を満たすクラスター内のアクティブなコンテナインスタンスごとに、1 つのタスクのみをデプロイします。この戦略を使用する場合、タスクの必要数や配置戦略、サービスの自動スケーリングポリシーを指定する必要はありません。詳細については、「デーモンスケジューリング戦略」を参照してください。

> **Note**
>
> Fargate タスクは DAEMON スケジュール戦略をサポートしていません。

必要数

`desiredCount`

タイプ: 整数

必須: いいえ

指定されたタスク定義のインスタンスをサービス内に配置し、実行し続ける数です。

サービスを更新しても、このパラメータは新しいサービスのデプロイをトリガーしません。

このパラメータは、REPLICA スケジュール戦略を使用する場合に必要です。サービスが DAEMON スケジュール戦略を使用する場合、このパラメータはオプションです。

サービスの自動スケーリングを使用する場合、現在実行中のサービスを、現在実行中のタスク数よりも少ない desiredCount で更新すると、サービスは指定された desiredCount までスケールダウンされます

デプロイ設定

`deploymentConfiguration`

タイプ: オブジェクト

必須: いいえ

デプロイ時に実行されるタスクの数と、タスクの停止および開始の順序を制御するオプションのデプロイパラメータ。

サービスを更新しても、このパラメータは新しいサービスのデプロイをトリガーしません。

`maximumPercent`

タイプ: 整数

必須: いいえ

サービスでローリング更新 (ECS) のデプロイタイプが使用されている場合、maximumPercent パラメータは、デプロイ時に RUNNING、STOPPING、または PENDING 状態で許可されるサービスのタスクの上限数を表します。これは、最も近い整数に切り捨てられた desiredCount のパーセンテージ (%) として表されます。このパラメータを使用して、デプロイのバッチサイズを定義できます。例えば、サービスで REPLICA サービススケジューラを使用して、desiredCount が 4 タスク、maximumPercent の値が 200% とすると、スケジューラは 4 つの古いタスクを停止する前に、4 つの新しいタスクを開始します。そのために必要なクラスターリソースを使用できることが前提です。maximumPercent サービススケジューラを使用するサービスのデフォルトの REPLICA 値は 200% です。

Amazon ECS スケジューラはこのパラメータを使用して、置き換えタスクを開始するためのクラスターリソースが使用可能である限り、最初に置き換えタスクを開始してから、異常なタスクを停止することで、異常なタスクを置き換えます。スケジューラが異常なタスクを置き換える仕組みの詳細については、「Amazon ECS サービス」を参照してください。

サービスで DAEMON サービススケジューラタイプを使用している場合、maximumPercent は 100% のままにする必要があります。これは、デフォルト値です。

デプロイ時のタスクの最大数は、desiredCount に maximumPercent/100 を乗算したもので、最も近い整数値に切り下げられます。

サービスで Blue/Green (CODE_DEPLOY) または EXTERNAL のいずれかのデプロイタイプと EC2 起動タイプを使用するタスクが使用されている場合、最大パーセントの値はデフォルト値に設定されます。この値は、コンテナインスタンスが DRAINING 状態にある間、サービス内で RUNNING 状態を保つタスクの数の上限を定義する目的で使用されます。

> **Note**
>
> ブルー/グリーン (CODE_DEPLOY) または EXTERNAL デプロイタイプのいずれかを使用し、EC2 を使用するタスクがあるサービスのカスタム maximumPercent 値を指定することはできません。

サービスがブルー/グリーン (CODE_DEPLOY) または EXTERNAL デプロイタイプのいずれかを使用し、サービス内のタスクが Fargate を使用する場合、最大パーセント値は使用されません。サービスを説明する際にも、この値は返されます。

`minimumHealthyPercent`

タイプ: 整数

必須: いいえ

サービスでローリング更新 (ECS) のデプロイタイプが使用されている場合、minimumHealthyPercent は、デプロイ時に RUNNING 状態に留まる必要があるサービスのタスクの下限数を表します。これは、最も近い整数に切り上げられた desiredCount のパーセンテージ (%) として表されます。このパラメータを使用して、追加のクラスターキャパシティーを使用せずにデプロイできます。

例えば、サービスで desiredCount が 4 タスク、minimumHealthyPercent が 50%、maximumPercent が 100% とすると、サービススケジューラは 2 つの新しいタスクを開始する前に、2 つの既存のタスクを停止してクラスターのキャパシティを解放できます。

いずれかのタスクに異常があり、maximumPercent が Amazon ECS スケジューラによる置き換えタスクの開始を許可しない場合、スケジューラは minimumHealthyPercent を制約として使用して、置き換えタスクを起動するためのキャパシティを解放するために、異常なタスクを 1 つずつ停止します。スケジューラが異常なタスクを置き換える仕組みの詳細については、「Amazon ECS サービス」を参照してください。

ロードバランサーを使用しないサービスの場合は、次のことを考慮してください。

- サービスのタスク内のすべての必須コンテナがヘルスチェックに合格すると、サービスは正常と見なされます。
- タスクにヘルスチェックが定義された必須コンテナがない場合、サービススケジューラは、タスクが RUNNING 状態に達した後 40 秒間待ってから、正常性の最小割合の合計にカウントします。
- タスクに、ヘルスチェックが定義された必須コンテナが 1 つ以上ある場合、サービススケジューラは、タスクが正常ステータスに達するのを待ってから、正常性の最小割合の合計にカウントします。タスク内のすべての必須コンテナがヘルスチェックに合格すると、タスクは正常と見なされます。サービススケジューラが待つことができる時間は、コンテナのヘルスチェックの設定によって決まります。詳細については、「ヘルスチェック」を参照してください。

ロードバランサーを使用するサービスについては、次のことを考慮してください。

- タスクにヘルスチェックが定義されている必須コンテナがない場合、サービススケジューラは、ロードバランサーターゲットグループのヘルスチェックが正常ステータスを返すのを待ってから、正常性の最小割合の合計にカウントします。
- タスクにヘルスチェックが定義されている必須コンテナがある場合、サービススケジューラは、タスクが正常なステータスになり、ロードバランサーターゲットグループのヘルス

チェックが正常ステータスを返すのを待ってから、タスクを正常性の最小割の合計にカウントします。

レプリカサービスの minimumHealthyPercent のデフォルト値は 100% です。DAEMON サービススケジュールを使用しているデフォルトの minimumHealthyPercent 値は、AWS CLI、AWS SDK、API では 0%、AWS マネジメントコンソール では 50% です。

デプロイ時の正常なタスクの最小数は、desiredCount に minimumHealthyPercent/100 を乗算したもので、最も近い整数値に切り上げられます。

サービスでブルー/グリーン (CODE_DEPLOY) または EXTERNAL のデプロイタイプが使用されており、EC2 を使用するタスクが実行されている場合、最小ヘルス率の値はデフォルト値に設定されます。この値は、コンテナインスタンスが DRAINING 状態にある間、サービス内で RUNNING 状態を保つタスクの数の下限を定義する目的で使用されます。

> **Note**
>
> ブルー/グリーン (CODE_DEPLOY) または EXTERNAL デプロイタイプのいずれかを使用し、EC2 を使用するタスクがあるサービスのカスタム maximumPercent 値を指定することはできません。

サービスがブルー/グリーン (CODE_DEPLOY) または EXTERNAL デプロイタイプのいずれかを使用しており、Fargate を使用するタスクを実行している場合、最小ヘルス率の値は使用されませんが、サービスについて説明する際に値が返されます。

earlySuccessCriteria

タイプ: オブジェクト

必須: いいえ

Amazon ECS がローリングデプロイを完了するタイミングを決定する早期成功基準。早期成功基準では、定義したタスクの割合がターゲットサービスリビジョンで実行中かつ正常である場合に、Amazon ECS はデプロイを完了します。Amazon ECS がデプロイを完了する前または後に、ソースサービスリビジョンのタスクをクリーンアップするかどうかを設定することもできます。詳細については、「早期成功基準を使用した Amazon ECS ローリングデプロイの早期完了」を参照してください。

enable

タイプ: ブール値

必須: いいえ

サービスで早期成功基準を有効にするかどうかを指定します。デフォルト値は false です。

healthyPercent

タイプ: 整数

必須: いいえ

Amazon ECS がデプロイを完了する前に、ターゲットサービスリビジョンで実行中かつ正常である必要がある必要数の割合を切り上げて指定します。値は、サービスの minimumHealthyPercent から 100 の間 (両端を含む) である必要があります。デフォルト値は 100 です。

sourceServiceRevisionCleanup

タイプ: 文字列

有効な値: BLOCKING | DEFERRED

必須: いいえ

Amazon ECS がソースサービスリビジョンのタスクをクリーンアップするタイミングを決定します。BLOCKING では、Amazon ECS はデプロイを完了する前にソースサービスリビジョンのタスクをクリーンアップします。DEFERRED では、Amazon ECS が最初にデプロイを完了し、デプロイの外部にあるソースサービスリビジョンのタスクをクリーンアップします。デフォルト値は BLOCKING です。

デプロイメントコントローラー

deploymentController

タイプ: オブジェクト

必須: いいえ

サービスで使用するデプロイコントローラータイプ。デプロイメントコントローラーが指定されていない場合は、ECS コントローラーが使用されます。詳細については、「Amazon ECS サービス」を参照してください。

サービスを更新しても、このパラメータは新しいサービスのデプロイをトリガーしません。

`type`

タイプ: 文字列

有効な値: `ECS` | `CODE_DEPLOY` | `EXTERNAL`

必須: はい

使用するデプロイコントローラータイプ。次の 3 つのデプロイコントローラータイプを使用できます。

`ECS`

Amazon ECS デプロイコントローラーは、ローリング更新、ブルー/グリーン、リニア、Canary の複数のデプロイ戦略をサポートしています。ローリング更新デプロイタイプでは、コンテナの現在実行されているバージョンが最新バージョンに置き換えられます。ブルー/グリーンデプロイでは、新しい環境が作成され、トラフィックがすべて一度に移行されます。リニアデプロイでは、トラフィックを同じ割合で徐々に移行します。カナリアデプロイでは、最初にトラフィックのごく一部を移行してから、残りのトラフィックを移行します。ローリング更新中に Amazon ECS がサービスに追加または削除するコンテナの数は、deploymentConfiguration で指定されるように、サービスのデプロイ中に許可される正常なタスクの最小数と最大数を調整することで制御されます。

`CODE_DEPLOY`

Blue/Green (`CODE_DEPLOY`) デプロイタイプは、CodeDeploy による ブルー/グリーンデプロイモデルを使用することにより、本稼働トラフィックを送信する前に、サービスの新しいデプロイを確認できます。

`EXTERNAL`

サードパーティーのデプロイコントローラーを使用して Amazon ECS サービスのデプロイプロセスを完全に制御することが必要な場合は、外部デプロイタイプを使用します。

タスクの配置

`placementConstraints`

タイプ: オブジェクトの配列

必須: いいえ

サービスのタスクに使用する、配置制約オブジェクトの配列。タスクごとに最大 10 個の制約を指定できます。この制限数には、タスク定義内の制約と、実行時に指定される制約が含まれます。Fargate を使用している場合、タスク配置の制約事項はサポートされません。

サービスを更新しても、このパラメータは新しいサービスのデプロイをトリガーしません。

`type`

タイプ: 文字列

必須: いいえ

制約のタイプ。特定のグループの各タスクが確実に別のコンテナインスタンスで実行されるようにするには、`distinctInstance` を使用します。選択対象を有効な候補グループに制約するには、`memberOf` を使用します。値 `distinctInstance` はタスク定義ではサポートされていません。

`expression`

タイプ: 文字列

必須: いいえ

制約に適用されるクラスタークエリ言語表現。制約タイプが `distinctInstance` である場合は、式を指定できません。詳細については、「Amazon ECS タスク用のコンテナインスタンスを定義する式を作成する」を参照してください。

`placementStrategy`

タイプ: オブジェクトの配列

必須: いいえ

サービスのタスクで使用する配置戦略オブジェクト。サービスごとに最大 4 つの戦略ルールを指定できます。

サービスを更新しても、このパラメータは新しいサービスのデプロイをトリガーしません。

`type`

タイプ: 文字列

有効な値: `random` | `spread` | `binpack`

必須: いいえ

配置戦略のタイプ。random 配置戦略は、タスクを利用可能な候補にランダムに配置します。spread 配置戦略は、field パラメータに基づいて、利用可能候補間で均等にタスクを分散して配置します。binpack 戦略は、field パラメータで指定したリソースの利用可能量が最も少ない利用可能候補にタスクを配置します。例えば、メモリの binpack 戦略を使用する場合、タスクは残りのメモリの量は最も少ないがタスクを実行するのに十分なインスタンスに配置されます。

`field`

タイプ: 文字列

必須: いいえ

配置戦略を適用するフィールド。spread 配置戦略では、有効な値は `instanceId` (または同じ効果を持つ `host`)、または `attribute:ecs.availability-zone` などのコンテナインスタンスに適用される任意のプラットフォームまたはカスタム属性です。binpack 配置戦略では、有効な値は `cpu` および `memory` です。random 配置戦略では、このフィールドは使用されません。

タグ

`tags`

タイプ: オブジェクトの配列

必須: いいえ

サービスに適用し、サービスの分類と整理に役立つメタデータ。各タグはキーとオプションの値で構成され、どちらもお客様側が定義します。サービスが削除されると、タグも削除されます。サービスには最大 50 個のタグを適用できます。詳細については、「Amazon ECS リソースにタグ付けする」を参照してください。

サービスを更新しても、このパラメータは新しいサービスのデプロイをトリガーしません。

`key`

タイプ: 文字列

長さの制限: 最小長は 1 です。最大長は 128 です。

必須: いいえ

タグを構成するキーと値のペアの一部。キーは、より具体的なタグ値のカテゴリのように動作する、一般的なラベルです。

`value`

タイプ: 文字列

長さの制約: 最小長は 0 です。最大長は 256 です。

必須: いいえ

タグを構成するキーと値のペアのオプションの一部。値はタグカテゴリ (キー) の記述子として機能します。

`enableECSManagedTags`

タイプ: ブール値

有効な値: `true` | `false`

必須: いいえ

サービスのタスクに Amazon ECS マネージドタグを使用するか否かを指定します。値を指定しない場合、デフォルトは `false` になります。詳細については、「請求にタグを使用する」を参照してください。

サービスを更新しても、このパラメータは新しいサービスのデプロイをトリガーしません。

`propagateTags`

タイプ: 文字列

有効な値: `TASK_DEFINITION` | `SERVICE`

必須: いいえ

タグをタスク定義またはサービスからサービスのタスクへコピーするかどうかを指定します。値を指定しない場合、タグはコピーされません。タグは、サービス作成中のサービス内のタスクにのみコピーすることができます。タグをサービス作成後またはタスク作成後のタスクに追加するには、`TagResource` API アクションを使用します。

サービスを更新しても、このパラメータは新しいサービスのデプロイをトリガーしません。

ネットワーク設定

`networkConfiguration`

タイプ: オブジェクト

必須: いいえ

サービスのネットワーク設定。このパラメータは awsvpc ネットワークモードを使用して独自の Elastic Network Interface を受け取るタスク定義の場合に必要です。その他のネットワークモードではサポートされていません。Fargate を使用している場合は、awsvpc ネットワークモードが必要です。EC2 のネットワークの詳細については、「EC2 の Amazon ECS タスクネットワークオプション」を参照してください。Fargate のネットワークの詳細については、「Fargate の Amazon ECS タスクのネットワークオプション」を参照してください。

サービスを更新すると、このパラメータは新しいサービスのデプロイをトリガーします。

`awsvpcConfiguration`

タイプ: オブジェクト

必須: いいえ

タスクまたはサービスのサブネットとセキュリティグループを表すオブジェクト。

`subnets`

型: 文字列の配列

必須: はい

タスクまたはサービスに関連付けられたサブネット。awsvpcConfiguration に従って指定できるサブネットは 16 個に制限されています。

`securityGroups`

型: 文字列の配列

必須: いいえ

タスクまたはサービスに関連付けられたセキュリティグループ。セキュリティグループを指定しないと、VPC のデフォルトのセキュリティグループが使用されます。awsvpcConfiguration に基づいて指定できるセキュリティグループは 5 つに制限されています。

assignPublicIP

タイプ: 文字列

有効な値: ENABLED | DISABLED

必須: いいえ

タスクの Elastic Network Interface がパブリック IP アドレスを受け取るかどうかを示します。値を指定しない場合、デフォルト値の DISABLED が使用されます。

healthCheckGracePeriodSeconds

タイプ: 整数

必須: いいえ

Amazon ECS サービススケジューラが、タスクが最初に開始された後で異常な Elastic Load Balancing、VPC Lattice、コンテナのヘルスチェックを無視する期間 (秒単位)。ヘルスチェックの猶予期間値を指定しない場合、デフォルト値 0 が使用されます。ヘルスチェックを使用しない場合、healthCheckGracePeriodSeconds は使用されません。

サービスのタスクが開始して応答するまでに時間がかかる場合は、ヘルスチェックの猶予期間として最大 2,147,483,647 秒 (約 69 年) まで指定できます。この間は、Amazon ECS サービススケジューラはヘルスチェックのステータスを無視します。この猶予期間により、サービススケジューラがタスクを異常とマークして時間より前に停止することがなくなります。

サービスを更新しても、このパラメータは新しいサービスのデプロイをトリガーしません。

loadBalancers

タイプ: オブジェクトの配列

必須: いいえ

サービスで使用するロードバランサーを表すロードバランサーオブジェクト。Application Load Balancer または Network Load Balancer を使,用するサービスの場合、6 個以上のターゲットグループはアタッチできません。

サービスを更新すると、このパラメータは新しいサービスのデプロイをトリガーします。

サービスの作成後にロードバランサーの設定を AWS マネジメントコンソール から変更することはできません。AWS Copilot、AWS CloudFormation、AWS CLI、SDK のいずれかを使用し

て、AWS CodeDeploy ブルー/グリーンまたは外部ではなく、ECS ローリングデプロイコントローラのみのロードバランサー設定を変更できます。ロードバランサー設定を追加、更新、削除すると、Amazon ECS は、更新された Elastic Load Balancing 設定で新しいデプロイを開始します。これにより、タスクがロードバランサーに登録およびロードバランサーから登録解除されます。Elastic Load Balancing 設定を更新する前に、テスト環境でこれを検証することをお勧めします。設定の変更方法については、「Amazon Elastic Containers Service API リファレンス」の「UpdateService」を参照してください。

Application Load Balancer および Network Load Balancers では、このオブジェクトには、ロードバランサーターゲットグループ ARN、コンテナ名 (コンテナの定義に表示)、ロードバランサーからアクセスされるコンテナポートが含まれている必要があります。このサービスのタスクがコンテナインスタンスに配置されると、コンテナインスタンスとポートの組み合わせは、指定したターゲットグループのターゲットとして登録されます。

`targetGroupArn`

タイプ: 文字列

必須: いいえ

サービスに関連付ける Elastic Load Balancing ターゲットグループの完全な Amazon リソースネーム (ARN)。

ターゲットグループ ARN は、Application Load Balancer または Network Load Balancer を使用する場合にのみ指定します。

`loadBalancerName`

タイプ: 文字列

必須: いいえ

サービスに関連付けるロードバランサーの名前。

Application Load Balancer または Network Load Balancer を使用している場合は、ロードバランサーの名前パラメータを省略します。

`containerName`

タイプ: 文字列

必須: いいえ

ロードバランサーに関連付けるコンテナの名前 (コンテナ定義に表示)。

`containerPort`

タイプ: 整数

必須: いいえ

ロードバランサーに関連付けるコンテナのポート。このポートは、サービスのタスクが使用しているタスク定義の `containerPort` に対応する必要があります。EC2 を使用するタスクの場合、コンテナインスタンスは、ポートマッピングの `hostPort` でインバウンドトラフィックを許可する必要があります。

`role`

タイプ: 文字列

必須: いいえ

Amazon ECS によるロードバランサーの呼び出しを許可する IAM ロールの短縮名または完全な ARN。このパラメーターは、サービスの単一のターゲットグループでロードバランサーを使用していて、タスク定義がawsvpcネットワークモードを使用していない場合にのみ許可されます。`role` パラメータを指定する場合、`loadBalancers` パラメータでロードバランサーのオブジェクトも指定する必要があります。

サービスを更新しても、このパラメータは新しいサービスのデプロイをトリガーしません。

指定したロールに / 以外のパスがある場合は、完全なロール ARN を指定するか (推奨)、ロール名の前にパスを付ける必要があります。例えば、ロールの名前が `bar` で、パスが `/foo/` の場合、ロール名として `/foo/bar` を指定します。詳細については、IAM ユーザーガイドの「わかりやすい名前とパス」を参照してください。

> **Important**
>
> アカウントが既に Amazon ECS サービスリンクロールを作成している場合は、ここでロールを指定しない限り、そのロールがサービスにデフォルトで使用されます。タスク定義で awsvpc ネットワークモードを使用している場合はサービスにリンクされたロールが必要です。詳細については、「Amazon ECS のサービスリンクロールの使用」を参照してください。

`serviceConnectConfiguration`

タイプ: オブジェクト

必須: いいえ

このサービスがサービスを検出して接続し、名前空間内の他のサービスによって検出され接続されるための設定。

サービスを更新すると、このパラメータは新しいサービスのデプロイをトリガーします。

詳細については、「Service Connect を使用して Amazon ECS サービスを短縮名で接続する」を参照してください。

`enabled`

タイプ: ブール値

必須: はい

このサービスで Service Connect を使用するかどうかを指定します。

`namespace`

タイプ: 文字列

必須: いいえ

Service Connect で使用する AWS Cloud Map 名前空間の短縮名または完全な Amazon リソースネーム (ARN)。名前空間は、Amazon ECS サービスおよびクラスターと同じ AWS リージョンにある必要があります。名前空間のタイプは Service Connect に影響しません。AWS Cloud Map の詳細については、「AWS Cloud Map デベロッパーガイド」の「サービスの使用」を参照してください。

`services`

タイプ: オブジェクトの配列

必須: いいえ

Service Connect サービスオブジェクトの配列。これらは、他の Amazon ECS サービスがこのサービスに接続するために使用する名前とエイリアスです (エンドポイントとも呼ばれます)。

名前空間のメンバーである「クライアント」 Amazon ECS サービスが名前空間内の他のサービスに接続するだけであれば、このフィールドは必要ありません。その一例が、サービスに接続されているロードバランサーまたは他の方法で受信リクエストを受け入れるフロントエンドアプリケーションです。

オブジェクトは、タスク定義からポートを選択し、AWS Cloud Map サービスの名前や、クライアントアプリケーションがこのサービスを参照するためのエイリアス (エンドポイントとも呼ばれます) とポートの配列を割り当てます。

portName

タイプ: 文字列

必須: はい

portName は、この Amazon ECS サービスのタスク定義内のすべてのコンテナからの、portMappings のいずれかの name と一致する必要があります。

discoveryName

タイプ: 文字列

必須: いいえ

discoveryName は、この Amazon ECS サービス用に Amazon ECS が作成する新しい AWS Cloud Map サービスの名前です。これは AWS Cloud Map 名前空間内で一意である必要があります。

このフィールドを指定しない場合、portName が使用されます。

clientAliases

タイプ: オブジェクトの配列

必須: いいえ

このサービス接続サービスのクライアントエイリアスのリストです。これらを使用して、クライアントアプリケーションで使用できる名前を割り当てます。このリストで使用できるクライアントエイリアスの最大数は 1 です。

各エイリアス (「エンドポイント」) は、他の Amazon ECS サービス (「クライアント」) がこのサービスに接続するために使用できる DNS 名とポート番号です。

名前とポートの組み合わせは、名前空間内で一意である必要があります。

これらの名前は、クライアントサービスの各タスク内で設定され、AWS Cloud Map では設定されません。これらの名前を解決するための DNS リクエストはタスクから離れることはなく、Elastic Network Interface ごとの 1 秒あたりの DNS リクエストの割り当てにもカウントされません。

port

タイプ: 整数

必須: はい

サービス接続プロキシのリスニングポート番号です。このポートは、同じ名前空間内のすべてのタスク内で使用できます。

クライアント Amazon ECS サービスのアプリケーションが変更されないようにするには、クライアントアプリケーションがデフォルトで使用するのと同じポートを設定します。

dnsName

タイプ: 文字列

必須: いいえ

dnsName は、このサービスに接続するためにクライアントタスクのアプリケーションで使用する名前です。この名前は有効な DNS ラベルでなければなりません。

このフィールドが指定されない場合、デフォルト値は discoveryName.namespace になります。discoveryName が指定されない場合、タスク定義から portName が使用されます。

クライアント Amazon ECS サービスのアプリケーションが変更されないようにするには、クライアントアプリケーションがデフォルトで使用するのと同じ名前を設定します。例えば、一般的な名前として database や db、または mysql や redis などのデータベース名の小文字表記が挙げられます。

ingressPortOverride

タイプ: 整数

必須: いいえ

(オプション) Service Connect プロキシがリッスンするポート番号です。

このフィールドの値を使用して、このアプリケーションのタスク定義で portMapping という名称で指定されているポート番号のトラフィックに対してプロキシをバイパスしてから、Amazon VPC セキュリティグループでそのポート番号を使用して、この Amazon ECS サービスのプロキシへのトラフィックを許可します。

awsvpc モードでは、このアプリケーションのタスク定義で portMapping という名称で指定されているコンテナポート番号がデフォルト値になります。bridge モードでは、Service Connect プロキシの動的エフェメラルポートがデフォルト値になります。

`logConfiguration`

タイプ: LogConfiguration オブジェクト

必須: いいえ

これにより、Service Connect プロキシログの公開場所が定義されます。このログは、予期しないイベント発生時のデバッグに使用します。この設定は、この Amazon ECS サービスの各タスクの Service Connect プロキシコンテナに `logConfiguration` パラメータを設定します。タスク定義でプロキシコンテナが指定されていません。

この Amazon ECS サービスのタスク定義のアプリケーションコンテナと同じログ設定を使用することをお勧めします。FireLens では、これはアプリケーションコンテナのログ設定です。`fluent-bit` または `fluentd` コンテナイメージを使用するのは FireLens ログルーターコンテナではありません。

`accessLogConfiguration`

タイプ: ServiceConnectAccessLogConfiguration オブジェクト

必須: いいえ

ログ形式やログにクエリ文字列を配置するかどうかなどの、Service Connect アクセスログ記録の設定。アクセスログは、リクエストパターン、レスポンスコード、タイミングデータなど、サービスに対して行われたリクエストに関する詳細情報をキャプチャします。アクセスログを有効にするには、`serviceConnectConfiguration` で `logConfiguration` も指定する必要があります。

`serviceRegistries`

タイプ: オブジェクトの配列

必須: いいえ

サービスのサービス検出設定の詳細。詳細については、「サービス検出を使用して Amazon ECS サービスを DNS 名で接続する」を参照してください。

サービスを更新すると、このパラメータは新しいサービスのデプロイをトリガーします。

`registryArn`

タイプ: 文字列

必須: いいえ

サービスレジストリの Amazon リソースネーム (ARN)。現在サポートされているサービスレジストリは AWS Cloud Map です。詳細については、AWS Cloud Map デベロッパーガイドの「サービスの使用」を参照してください。

`port`

タイプ: 整数

必須: いいえ

サービス検出サービスが SRV レコードを指定した場合に使用されるポート値。このフィールドは、`awsvpc` ネットワークモードと SRV レコードの両方が使用されている場合に必要です。

`containerName`

タイプ: 文字列

必須: いいえ

サービス検出サービスに使用されるコンテナ名の値。この値は、タスク定義で指定されます。サービスタスクが指定するタスク定義に `bridge` または `host` ネットワークモードが使用されている場合は、タスク定義からの `containerName` と `containerPort` の組み合わせを指定する必要があります。サービスタスクが指定するタスク定義に `awsvpc`ネットワークモードが使用され、SRV DNS タイプのレコードが使用されている場合は、`containerName` と `containerPort` の組み合わせを指定するか、`port` 値を指定する必要があります (両方は指定しないでください)。

`containerPort`

タイプ: 整数

必須: いいえ

サービス検出サービスに使用されるポートの値。この値は、タスク定義で指定されます。サービスタスクが指定するタスク定義に `bridge` または `host` ネットワークモードが使用されて

いる場合は、タスク定義からの containerName と containerPort の組み合わせを指定する必要があります。サービスタスクが指定するタスク定義に awsvpc ネットワークモードが使用され、SRV DNS タイプのレコードが使用されている場合は、containerName と containerPort の組み合わせを指定するか、port 値を指定する必要があります (両方は指定しないでください)。

クライアントトークン

clientToken

タイプ: 文字列

必須: いいえ

リクエストのべき等のために割り当てる一意の識別子 (大文字と小文字を区別)。最大 32 文字の ASCII 文字を使用できます。

アベイラビリティーゾーンの再調整

availabilityZoneRebalancing

タイプ: 文字列

必須: いいえ

サービスがアベイラビリティーゾーンの再調整を使用するかどうか示します。有効な値は ENABLED および DISABLED です。

サービスを更新しても、このパラメータは新しいサービスのデプロイをトリガーしません。

デフォルトの動作

- 新しいサービスの場合: 値が指定されていない場合、デフォルトは DISABLED です。
- 既存のサービスの場合: 値が指定されていない場合、Amazon ECS はデフォルトで既存の値を使用します。以前に値が設定されていない場合、Amazon ECS は値を DISABLED に設定します。

アベイラビリティーゾーンのリバランスの詳細については、「アベイラビリティーゾーン間での Amazon ECS サービスの調整」を参照してください。

ボリューム設定

`volumeConfigurations`

タイプ: オブジェクト

必須: いいえ

サービスによって管理されるタスク用のボリュームを作成するのに使用される設定。このオブジェクトを使用して設定できるのは、タスク定義で「`configuredAtLaunch`」とマークされているボリュームだけです。

サービスを更新すると、このパラメータは新しいサービスのデプロイをトリガーします。

このオブジェクトは、サービスによって管理されるタスクに Amazon EBS ボリュームをアタッチするのに必要です。詳細については、「Amazon ECS での Amazon EBS ボリュームの使用」を参照してください。

`name`

型: 文字列

必須: はい

サービスを作成または更新するときに設定されるボリュームの名前。最大 255 文字の英字 (大文字と小文字)、数字、アンダースコア (_)、ハイフン (-) が可能です。この値は、タスク定義で指定されたボリューム名と一致する必要があります。

`managedEBSVolume`

タイプ: オブジェクト

必須: いいえ

サービスの作成または更新時にそのサービスによって管理されるタスクにアタッチする Amazon EBS ボリュームを作成するためのボリューム設定。1 つのタスクごとに 1 つのボリュームがアタッチされます。

`encrypted`

タイプ: ブール値

必須: いいえ

有効な値: `true|false`

それぞれのAmazon EBS ボリュームを暗号化するかどうかを指定します。AWS アカウント の特定の AWS リージョン に対してデフォルトで Amazon EBS 暗号化を有効にし、同時にこのパラメータを `false` に設定すると、このパラメータはオーバーライドされ、ボリュームはデフォルトで暗号化用に指定された KMS キーにより暗号化されます。デフォルトでの Amazon EBS 暗号化の詳細については、「Amazon EBS ユーザーガイド」の「Amazon EBS 暗号化をデフォルトで有効にする」を参照してください。Amazon ECS タスクにアタッチされた Amazon EBS ボリュームの暗号化の詳細については、「Amazon ECS タスクにアタッチされた Amazon EBS ボリュームに保存されているデータの暗号化」を参照してください。

`kmsKeyId`

タイプ: 文字列

必須: いいえ

Amazon EBS 暗号化に使用する AWS Key Management Service (AWS KMS) キーの識別子。`kmsKeyId` を指定する場合、暗号化された状態は `true` である必要があります。

このパラメータを使用して指定されたキーは、指定した Amazon ECS マネージドストレージ暗号化の Amazon EBS デフォルトまたはクラスターレベルの KMS キーをオーバーライドします。詳細については、「Amazon ECS タスクにアタッチされた Amazon EBS ボリュームに保存されているデータの暗号化」を参照してください。

以下のいずれかを使用して KMS キーを指定できます。

- キー ID – `1234abcd-12ab-34cd-56ef-1234567890ab` など。
- キーエイリアス – `alias/ExampleAlias` など。
- キー ARN – `arn:aws:kms:us-east-1:012345678910:key/1234abcd-12ab-34cd-56ef-1234567890ab` など。
- エイリアス ARN – `arn:aws:kms:us-east-1:012345678910:alias/ExampleAlias` など。

> **Important**
>
> AWS では、KMS キーを非同期で認証します。したがって、無効な ID、エイリアス、または ARN を指定すると、アクションは正常に行われているように見える場合がありますが、最終的には失敗します。詳細については、「Troubleshooting Amazon EBS volume attachment issues」を参照してください。

`volumeType`

タイプ: 文字列

必須: いいえ

有効な値: `gp2|gp3|io1|io2|sc1|st1|standard`

Amazon EBS ボリュームの種類。ボリュームタイプの詳細については、「Amazon EBS ユーザーガイド」の「Amazon EBS ボリュームタイプ」を参照してください。デフォルトのボリュームタイプは `gp3` です。

> **Note**
>
> `standard` ボリュームタイプは Fargate タスクではサポートされていません。

`sizeInGiB`

タイプ: 整数

必須: いいえ

有効な範囲: 1 ~ 16,384 の整数。

EBS ボリュームのギビバイト (GiB) でのサイズ。アタッチするボリュームを設定するスナップショット ID を指定しない場合は、サイズ値を指定する必要があります。スナップショットを使用してボリュームをアタッチするように設定した場合、デフォルト値はスナップショットサイズです。スナップショットサイズ以上のサイズを指定できます。

`gp2` および `gp3` ボリュームタイプでは、有効範囲は 1 ~ 16,384 です。

`io1` および `io2` ボリュームタイプでは、有効範囲は 4 ~ 16,384 です。

`st1` および `sc1` ボリュームタイプでは、有効範囲は 125 ~ 16,384 です。

`standard` ボリュームタイプでは、有効範囲は 1 ~ 1,024 です。

`snapshotId`

タイプ: 文字列

必須: いいえ

Amazon ECS がアタッチメント用の新しいボリュームの作成に使用する既存の Amazon EBS ボリュームのスナップショットの ID。snapshotId または sizeInGiB のどちらかを指定する必要があります。

volumeInitializationRate

タイプ: 整数

必須: いいえ

既存の Amazon EBS ボリュームのスナップショットからデータを取得してアタッチメント用の新しいボリュームを作成する際の MiB/秒単位のデータ読込みレート。このプロパティは、snapshotId を指定する場合にのみ指定できます。サポートされている初期化レートの範囲など、このボリューム初期化レートの詳細については、「Amazon EBS ユーザーガイド」の「Amazon EBS ボリュームの初期化」を参照してください。

iops

タイプ: 整数

必須: いいえ

1 秒あたりの I/O 操作の数 (IOPS)。gp3、io1、io2 ボリュームの場合、これはボリュームでプロビジョニングされている IOPS の数を表します。gp2 ボリュームの場合、この値はボリュームのベースラインパフォーマンスと、バースト用の I/O クレジットがこのボリュームに蓄積されるレートを表します。このパラメータは、io1 と io2 のボリュームに必須です。このパラメータは、gp2、st1、sc1、standard ボリュームではサポートされていません。

gp3 ボリュームでは、値の有効範囲は 3,000 ~ 16,000 です。

io1 ボリュームでは、値の有効範囲は 100 ~ 64,000 です。

io2 ボリュームでは、値の有効範囲は 100 ~ 64,000 です。

throughput

タイプ: 整数

必須: いいえ

サービスで維持されるタスクにアタッチされたボリュームにプロビジョニングするためのスループット。

> **Important**
>
> このパラメータは、gp3 ボリュームのみでサポートされています。

`roleArn`

タイプ: 文字列

必須: はい

タスク用の Amazon EBS リソースを管理するための Amazon ECS アクセス許可を提供するインフラストラクチャ AWS Identity and Access Management (IAM) ロールの Amazon リソース ARN (ARN)。詳細については、「Amazon ECS インフラストラクチャ IAM ロール」を参照してください。

`tagSpecifications`

タイプ: オブジェクト

必須: いいえ

各 Amazon EBS ボリュームに適用されるタグの指定。

`resourceType`

タイプ: 文字列

必須: はい

有効な値: `volume`

作成時にタグ付けするリソースのタイプ。

`tags`

タイプ: オブジェクトの配列

必須: いいえ

分類と整理に役立つようにボリュームに適用するメタデータ。タグはそれぞれ、1 つのキーとオプションの 1 つの値で構成されており、どちらもユーザーが定義します。`AmazonECSCreated` と `AmazonECSManaged` は Amazon ECS がユーザーに代わって追加する予約済みのタグなので、独自のタグを最大 48 個指定できます。ボ

リュームが削除されると、タグも削除されます。詳細については、「Amazon ECS リソースにタグ付けする」を参照してください。

key

タイプ: 文字列

長さの制限: 最小長は 1 です。最大長は 128 です。

必須: いいえ

タグを構成するキーと値のペアの一部。キーは、より具体的なタグ値のカテゴリのように動作する、一般的なラベルです。

value

タイプ: 文字列

長さの制約: 最小長は 0 です。最大長は 256 です。

必須: いいえ

タグを構成するキーと値のペアのオプションの一部。値はタグカテゴリ (キー) の記述子として機能します。

propagateTags

タイプ: 文字列

有効な値: TASK_DEFINITION | SERVICE | NONE

必須: いいえ

タグをタスク定義またはサービスからボリュームにコピーするかどうかを指定します。NONE を指定した場合、または値を指定しない場合、タグはコピーされません。

fileSystemType

タイプ: 文字列

必須: いいえ

有効な値: xfs|ext3|ext4|NTFS

ボリューム上のファイルシステムのタイプ。ボリュームのファイルシステムタイプによって、ボリューム内でのデータの格納方法と取得方法が決まります。スナップショットから作成されたボリュームでは、スナップショットの作成時にボリュームが使用していたのと

同じファイルシステムタイプを指定する必要があります。ファイルシステムのタイプが一致しない場合、タスクは開始できません。

Linux の有効な値は xfs、ext3, and ext4 です。Linux タスクにアタッチされているボリュームのデフォルトは XFS です。

Windows の有効な値は NTFS です。TWindows タスクにアタッチされるボリュームのデフォルトは NTFS です。

サービス定義テンプレート

以下に、Amazon ECS サービス定義の JSON 表現を示します。

EC2

```
{
    "cluster": "",
    "serviceName": "",
    "taskDefinition": "",
    "loadBalancers": [
        {
            "targetGroupArn": "",
            "loadBalancerName": "",
            "containerName": "",
            "containerPort": 0
        }
    ],
    "serviceRegistries": [
        {
            "registryArn": "",
            "port": 0,
            "containerName": "",
            "containerPort": 0
        }
    ],
    "desiredCount": 0,
    "clientToken": "",
    "launchType": "EC2",
    "capacityProviderStrategy": [
        {
            "capacityProvider": "",
            "weight": 0,
```

```
            "base": 0
        }
    ],
    "platformVersion": "",
    "role": "",
    "deploymentConfiguration": {
        "deploymentCircuitBreaker": {
            "enable": true,
            "rollback": true
        },
        "maximumPercent": 0,
        "minimumHealthyPercent": 0,
        "alarms": {
            "alarmNames": [
                ""
            ],
            "enable": true,
            "rollback": true
        }
    },
    "placementConstraints": [
        {
            "type": "distinctInstance",
            "expression": ""
        }
    ],
    "placementStrategy": [
        {
            "type": "binpack",
            "field": ""
        }
    ],
    "networkConfiguration": {
        "awsvpcConfiguration": {
            "subnets": [
                ""
            ],
            "securityGroups": [
                ""
            ],
            "assignPublicIp": "DISABLED"
        }
    },
    "healthCheckGracePeriodSeconds": 0,
```

```
    "schedulingStrategy": "REPLICA",
    "deploymentController": {
        "type": "EXTERNAL"
    },
    "tags": [
        {
            "key": "",
            "value": ""
        }
    ],
    "enableECSManagedTags": true,
    "propagateTags": "TASK_DEFINITION",
    "enableExecuteCommand": true,
    "availabilityZoneRebalancing": "ENABLED",
    "serviceConnectConfiguration": {
        "enabled": true,
        "namespace": "",
        "services": [
            {
                "portName": "",
                "discoveryName": "",
                "clientAliases": [
                    {
                        "port": 0,
                        "dnsName": ""
                    }
                ],
                "ingressPortOverride": 0
            }
        ],
        "logConfiguration": {
            "logDriver": "journald",
            "options": {
                "KeyName": ""
            },
            "secretOptions": [
                {
                    "name": "",
                    "valueFrom": ""
                }
            ]
        }
    },
    "volumeConfigurations": [
```

```
        {
            "name": "",
            "managedEBSVolume": {
                "encrypted": true,
                "kmsKeyId": "",
                "volumeType": "",
                "sizeInGiB": 0,
                "snapshotId": "",
                "volumeInitializationRate": 0,
                "iops": 0,
                "throughput": 0,
                "tagSpecifications": [
                    {
                        "resourceType": "volume",
                        "tags": [
                            {
                                "key": "",
                                "value": ""
                            }
                        ],
                        "propagateTags": "NONE"
                    }
                ],
                "roleArn": "",
                "filesystemType": ""
            }
        }
    ]
}
```

Fargate

```
{
    "cluster": "",
    "serviceName": "",
    "taskDefinition": "",
    "loadBalancers": [
        {
            "targetGroupArn": "",
            "loadBalancerName": "",
            "containerName": "",
            "containerPort": 0
        }
```

```
    ],
    "serviceRegistries": [
        {
            "registryArn": "",
            "port": 0,
            "containerName": "",
            "containerPort": 0
        }
    ],
    "desiredCount": 0,
    "clientToken": "",
    "launchType": "FARGATE",
    "capacityProviderStrategy": [
        {
            "capacityProvider": "",
            "weight": 0,
            "base": 0
        }
    ],
    "platformVersion": "",
    "platformFamily": "",
    "role": "",
    "deploymentConfiguration": {
        "deploymentCircuitBreaker": {
            "enable": true,
            "rollback": true
        },
        "maximumPercent": 0,
        "minimumHealthyPercent": 0,
        "alarms": {
            "alarmNames": [
                ""
            ],
            "enable": true,
            "rollback": true
        }
    },
    "placementStrategy": [
        {
            "type": "binpack",
            "field": ""
        }
    ],
    "networkConfiguration": {
```

```
        "awsvpcConfiguration": {
            "subnets": [
                ""
            ],
            "securityGroups": [
                ""
            ],
            "assignPublicIp": "DISABLED"
        }
    },
    "healthCheckGracePeriodSeconds": 0,
    "schedulingStrategy": "REPLICA",
    "deploymentController": {
        "type": "EXTERNAL"
    },
    "tags": [
        {
            "key": "",
            "value": ""
        }
    ],
    "enableECSManagedTags": true,
    "propagateTags": "TASK_DEFINITION",
    "enableExecuteCommand": true,
    "availabilityZoneRebalancing": "ENABLED",
    "serviceConnectConfiguration": {
        "enabled": true,
        "namespace": "",
        "services": [
            {
                "portName": "",
                "discoveryName": "",
                "clientAliases": [
                    {
                        "port": 0,
                        "dnsName": ""
                    }
                ],
                "ingressPortOverride": 0
            }
        ],
        "logConfiguration": {
            "logDriver": "journald",
            "options": {
```

```
                "KeyName": ""
            },
            "secretOptions": [
                {
                    "name": "",
                    "valueFrom": ""
                }
            ]
        }
    },
    "volumeConfigurations": [
        {
            "name": "",
            "managedEBSVolume": {
                "encrypted": true,
                "kmsKeyId": "",
                "volumeType": "",
                "sizeInGiB": 0,
                "snapshotId": "",
                "volumeInitializationRate": 0,
                "iops": 0,
                "throughput": 0,
                "tagSpecifications": [
                    {
                        "resourceType": "volume",
                        "tags": [
                            {
                                "key": "",
                                "value": ""
                            }
                        ],
                        "propagateTags": "NONE"
                    }
                ],
                "roleArn": "",
                "filesystemType": ""
            }
        }
    ]
}
```

このサービス定義テンプレートは、次の AWS CLI コマンドを使用して作成できます。

```
aws ecs create-service --generate-cli-skeleton
```

Amazon ECS リソースにタグ付けする

Amazon ECS リソースを管理しやすくするために、タグを使用してオプションで各リソースに独自のメタデータを割り当てることができます。各タグは、キーおよび値 (オプション) で構成されます。

タグを使用すると、Amazon ECS リソースを目的、所有者、環境などの、さまざまな方法で分類することができます。これは、同じ種類のリソースが多い場合に役立ちます。リソースに割り当てたタグに基づいて、特定のリソースをすばやく識別できます。例えば、アカウントの Amazon ECS コンテナインスタンスの一連のタグを定義できます。これは、各インスタンスの所有者とスタックレベルを追跡するのに役立ちます。

コストと使用状況レポートでタグを使用できます。これらのレポートを使用して、Amazon ECS リソースのコストと使用状況を分析できます。詳細については、「the section called “使用状況レポート”」を参照してください。

> **Warning**
>
> タグキーとその値を返す API は多数あります。`DescribeTags` へのアクセスを拒否しても、他の API から返されるタグへのアクセスは自動的に拒否されません。ベストプラクティスとして、機密データをタグに含めないようお勧めします。

各リソースタイプのニーズを満たす一連のタグキーを考案することをお勧めします。リソースの管理を容易にするために、タグキーの一貫したセットを使用できます。追加したタグに基づいてリソースを検索およびフィルタリングできます。

タグには、Amazon ECS に関連する意味はなく、完全に文字列として解釈されます。タグのキーと値は編集でき、タグはリソースからいつでも削除できます。タグの値を空の文字列に設定することはできますが、タグの値を 空値 に設定することはできません。そのリソースの既存のタグと同じキーを持つタグを追加した場合、古い値は新しい値によって上書きされます。リソースを削除すると、リソースのタグも削除されます。

AWS Identity and Access Management (IAM) を使用すると、AWS アカウント内のどのユーザーがタグを管理するアクセス許可を持っているかを制御できます。

リソースのタグ付け方法

Amazon ECS タスク、サービス、タスク定義、およびクラスターへのタグ付けには、次のような複数の方法が存在します。

- AWS マネジメントコンソール、Amazon ECS API、AWS、または AWS CLI SDK を使用して、ユーザーが手動でリソースにタグ付けする。
- ユーザーが Amazon ECS マネージドタグのオプションを選択して、サービスを作成する、あるいはスタンドアロンのタスクを実行する。

  新しく起動されたすべてのタスクに対し、Amazon ECS が自動的なタグ付けを行う。詳細については、「the section called “Amazon ECS マネージドのタグ”」を参照してください。
- ユーザーがコンソールを使用してリソースを作成する。そのリソースに対し、コンソールが自動的にタグ付けを行います。

  これらのタグは、AWS CLI、および AWS SDK のレスポンスとして返され、コンソールに表示されます。これらのタグを、ユーザーが変更または削除することはできません。

  追加されたタグの詳細については、「Amazon ECS リソースのタグ付けのサポート」表内で、「コンソールによって自動的に追加されるタグ」列を参照してください。

リソースの作成時にタグを指定したにも関わらず、そのタグが適用されない場合、Amazon ECS は、対象の作成プロセスをロールバックします。これにより、リソースがタグ付きで作成されるか、まったく作成されないようになるため、タグ付けされていないリソースが存在することがなくなります。作成時にリソースにタグ付けすることで、リソース作成後にカスタムタグ付けスクリプトを実行する必要がなくなります。

以下の表に、タグ付けをサポートしている Amazon ECS リソースを示します。

リソース	タグをサポート	タグの伝播をサポート	コンソールによって自動的に追加されたタグ
Amazon ECS タスク	はい	はい、タスク定義からサポートします。	[Key] (キー): `aws:ecs:clusterName` 値: `cluster-name`

リソース	タグをサポート	タグの伝播をサポート	コンソールによって自動的に追加されたタグ
Amazon ECS サービス	はい	はい、タスク定義またはサービス内のタスクへのサービスのいずれかからサポートします。	[Key] (キー): `ecs:service:stackId` 値 `arn:aws:cloudformation: arn`
Amazon ECSの タスクセット	はい	なし	該当なし
Amazon ECS のタスク定義	はい	なし	[Key] (キー): `ecs:taskDefinition:createdFrom` 値: `ecs-console-v2`

リソース	タグをサポート	タグの伝播をサポート	コンソールによって自動的に追加されたタグ
Amazon ECS クラスター	はい	なし	[Key] (キー): `aws:cloudformation:logical-id` 値: `ECSCluster` [Key] (キー): `aws:cloudformation:stack-id` 値: `arn:aws:cloudformation:` *`arn`* [Key] (キー): `aws:cloudformation:stack-name` 値: `ECS-Console-V2-Cluster-` *`EXAMPLE`*
Amazon ECS コンテナインスタンス	はい	はい、Amazon EC2 インスタンスからサポートします。詳細については、「Amazon ECS の Amazon EC2 コンテナインスタンスにタグを追加する」を参照してください。	該当なし

リソース	タグをサポート	タグの伝播をサポート	コンソールによって自動的に追加されたタグ
Amazon ECS 外部インスタンス	はい	なし	該当なし
Amazon ECS キャパシティープロバイダー	はい。 事前定義された FARGATE および FARGATE_SPOT キャパシティープロバイダーに対し、タグ付けすることはできません。	はい。Amazon ECS マネージドインスタンスのキャパシティプロバイダーからのみ可能です。Amazon ECS マネージドインスタンスのキャパシティプロバイダーから、Amazon EC2 インスタンス、起動テンプレート、Elastic Network Interface、ボリュームなど、プロバイダーが管理するすべてのリソースにタグを伝播できます。	該当なし

作成時のリソースのタグ付け

次のリソースは、Amazon ECS API、AWS CLI、または AWS SDK を使用した作成時のタグ付けをサポートしています。

- Amazon ECS タスク
- Amazon ECS サービス
- Amazon ECS タスク定義
- Amazon ECSの タスクセット
- Amazon ECS クラスター

- Amazon ECS コンテナインスタンス
- Amazon ECS キャパシティープロバイダー

Amazon ECS には、リソースの作成にタグ付け認可を使用するオプションがあります。AWS アカウント がタグの承認用に設定されている場合、ユーザーには、リソースを作成するアクションの許可が必要です (`ecsCreateCluster` など)。リソース作成アクションでタグが指定されている場合、AWS は追加の認可を実行して、ユーザーまたはロールがタグを作成するための許可を持っているかどうかを確認します。したがって、`ecs:TagResource` アクションを使用するための明示的な許可を付与する必要があります。詳細については、「the section called “リソース作成時のタグ付け”」を参照してください。オプションを設定する方法については、「the section called “タグ付け認可”」を参照してください。

制限事項

タグには以下の制限があります。

- 1 つのリソースに対して最大 50 個のタグを関連付けることができます。
- 1 つのリソースに対してタグキーを繰り返すことはできません。各タグキーは一意である必要があり、それぞれに使用できる値は 1 つのみです。
- キーの値には最大 128 UTF-8 文字を使用できます。
- 値の最大長は 256 UTF-8 文字です。
- 複数の AWS のサービスおよびリソースがタグ付けスキーマを使用する場合、使用する文字の種類を制限します。一部のサービスでは、使用できる文字に制限がある場合があります。通常、使用できる文字は、英字、数字、スペース、および特殊文字 **+**、**-**、**=**、**.**、**_**、**:**、**/**、**@** です。
- タグのキーと値では、大文字と小文字が区別されます。
- キーまたは値のプレフィックスとして、`aws:`、`AWS:`、またはこれら大文字または小文字の任意の組み合わせを使用することはできません。これらは AWS でのみ使用するように予約されています。このプレフィックスを持つタグのキーや値を編集または削除することはできません。このプレフィックスを持つタグは、リソースあたりのタグ数の制限にはカウントされません。

Amazon ECS マネージドのタグ

Amazon ECS マネージド タグを使用すると、Amazon ECS は、クラスター情報とユーザーが追加したタスク定義タグまたはサービスタグのいずれかを使用して、新しく起動されたすべてのタスクとタ

スクにアタッチされた Amazon EBS ボリュームに自動的にタグ付けします。以下は、追加されたタグについて説明しています。

- スタンドアロンタスク – キーには `aws:ecs:clusterName` を、値にはクラスター名を設定したタグ。ユーザーによって追加されたすべてのタスク定義タグです。スタンドアロンタスクにアタッチされた Amazon EBS ボリュームは、キーを `aws:ecs:clusterName` として、値がクラスター名に設定されたタグを受け取ります。Amazon EBS ボリュームのタグ付けの詳細については、「Amazon EBS ボリュームのタグ付け」を参照してください。
- サービスの一部であるタスク – キーを `aws:ecs:clusterName` として、値をクラスター名として設定したタグ。キーに `aws:ecs:serviceName` を、値にサービス名を設定したタグです。以下のリソースのいずれかからのタグです。
  - タスク定義 – ユーザーによって追加されたすべてのタスク定義タグ。
  - サービス – ユーザーによって追加されたすべてのサービスタグ。

    サービスの一部であるタスクにアタッチされた Amazon EBS ボリュームは、キーを `aws:ecs:clusterName` として、値をクラスター名に設定されたタグと、キーを `aws:ecs:serviceName` として値をサービス名に設定されたタグを受け取ります。Amazon EBS ボリュームのタグ付けの詳細については、「Amazon EBS ボリュームのタグ付け」を参照してください。

この機能には、以下のオプションが必要です。

- 新しい Amazon リソースネーム (ARN) とリソース識別子 (ID) 形式にオプトインする必要があります。詳細については、「Amazon リソースネーム (ARN) と ID」を参照してください。
- API を使用してサービスを作成したり、タスクを実行したりするときは、`run-task` と `create-service` の `enableECSManagedTags` を `true` に設定する必要があります。詳細については、「AWS Command Line Interface API リファレンス」の「create–service」および「run–task」を参照してください。
- Amazon ECS はマネージドタグを使用して、クラスターの Auto Scaling などの一部の機能をいつ有効にするかを判断します。Amazon ECS が機能を効果的に管理できるように、タグを手動で変更しないことをお勧めします。

請求にタグを使用する

AWS は、Amazon ECS リソースのコストおよび使用量を分析するために使用できる、Cost Explorer と呼ばれるレポートツールを提供します。

Cost Explorer を使用して、使用状況とコストのグラフを表示できます。過去 13 か月からデータを表示でき、また次の 3 か月間にどのくらい使用する可能性があるかを予測します。Cost Explorer を使用すると、時間の経過とともに AWS リソースに費やす金額のパターンを確認できます。例えば、Cost Explorer を使用して、さらに調べる必要がある分野を特定し、コストを把握するために使用できる傾向を確認できます。データの時間範囲を指定したり、時間データを日または月ごとに表示することもできます。

コストと使用状況レポートには、Amazon ECS マネージドのタグ、またはユーザーにより追加されたタグを使用できます。詳細については、「Amazon ECS 使用状況レポート」を参照してください。

リソースを組み合わせたコストを確認するには同じタグキー値を持つリソースに基づいて、請求情報を整理します。例えば、複数のリソースに特定のアプリケーション名のタグを付け、請求情報を整理することで、複数のサービスを利用しているアプリケーションの合計コストを確認することができます。タグによるコスト配分レポートの設定の詳細については、AWS Billing ユーザーガイドのコスト配分月次レポートを参照してください。

さらに、[コスト配分データの分割] をオンにして、コストと使用状況レポートでタスクレベルの CPU とメモリの使用状況データを取得できます。詳細については、「タスクレベルのコストと使用状況レポート」を参照してください。

> **Note**
>
> レポートをオンにしている場合、当月のデータを表示できるようになるまでに最大 24 時間かかることがあります。

Amazon ECS リソースにタグを追加する

新規または既存のタスク、サービス、タスク定義、またはクラスターにタグ付けできます。コンテナインスタンスのタグ付けについては、「Amazon ECS の Amazon EC2 コンテナインスタンスにタグを追加する」を参照してください。

> **Warning**
>
> 個人情報 (PII) などの機密情報や秘匿性の高い情報はタグに追加しないようにします。タグは、多くの AWS のサービス (請求など) からアクセスできます。タグは、プライベートデータや機密データに使用することを意図していません。

リソースの作成時に、次のリソースを使用してタグを指定できます。

タスク	コンソール	AWS CLI	API アクション
1 つまたは複数のタスクを実行する	Amazon ECS タスクとしてのアプリケーションの実行	実行タスク	RunTask
サービスを作成します。	Amazon ECS のローリング更新デプロイの作成	サービスの作成	CreateService
タスクセットを作成します。	サードパーティーのコントローラーを使用して Amazon ECS サービスをデプロイする	タスクセットを作成	CreateTaskSet
タスク定義を登録する	the section called “コンソールを使用したタスク定義の作成”	タスク定義を登録	RegisterTaskDefinition
クラスターを作成する。	Fargate ワークロード用の Amazon ECS クラスターを作成する	クラスター作成	CreateCluster
1 つまたは複数のコンテナインスタンスを実行する	Amazon ECS Linux コンテナインスタンスの起動	実行インスタンス	RunInstances

タスク	コンソール	AWS CLI	API アクション
Amazon ECS マネージドインスタンスのキャパシティプロバイダーを作成します。	Amazon ECS マネージドインスタンスのキャパシティプロバイダーを作成する	create-capacity-provider	CreateCapacityProvider

既存のリソースにタグを追加する (Amazon ECS コンソール)

Amazon ECS では、クラスター、サービス、タスク定義に関連付けられているタグを、そのリソースのページから直接追加または削除できます。

個々のリソースのタグを変更するには

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションバーから、使用する AWS リージョン を選択します。
3. ナビゲーションペインでリソースタイプ (例:Clusters (クラスター) を選択します
4. リソースリストからリソースを選択し、[Tags] (タグ) タブを選択してから、[Manage tags] (タグの管理) を選択します。
5. タグを設定します。

   [タグを追加] [Add tag] (タグを追加) を選択し、以下を実行します。

   - [キー] にはキー名を入力します。
   - [値] にキー値を入力します。
6. [保存] を選択します。

既存のリソースにタグを追加する (AWS CLI)

AWS CLI または API を使用して、1 つ以上のタグを追加または上書きできます。

> **Note**
>
> デュアルスタックサービスエンドポイントを使用することで、IPv4 と IPv6 の両方を介して AWS CLI、SDK、および Amazon ECS API から Amazon ECS とやり取りできます。詳細については、「Amazon ECS デュアルスタックエンドポイントの使用」を参照してください。

- AWS CLI - tag-resource
- API アクション - TagResource

Amazon ECS マネージドインスタンスのタグ

Amazon ECS マネージドインスタンスは、コスト最適化に使用できるカスタムタグと AWS によって追加されたタグの組み合わせをサポートします。請求におけるタグの使用の詳細については、「請求にタグを使用する」を参照してください。

AWS によって追加されたタグ

AWS は、キャパシティプロバイダーによって作成された各 Amazon ECS マネージドインスタンスに次のタグを追加します。

- Amazon ECS は、予約済みタグ `AmazonECSCreated` と `AmazonECSManaged`を Amazon ECS マネージドインスタンスに自動的に追加します。
- Amazon ECS は、各インスタンスに次のシステムタグを追加します。
  - キーに `aws:ecs:clusterName` を、値にクラスター名を設定したタグ。
  - キーに `aws:ecs:capacityProviderName` を、値にキャパシティプロバイダー名を設定したタグ。
  - キーに `aws:ecs:containerInstanceId` を、値に Amazon ECS マネージドインスタンスのコンテナインスタンス ID を設定したタグ。
- Amazon EC2 は、値 `ecs-managed-instances` を持つシステムタグ `aws:ec2:managed-launch` を追加します。また Amazon EC2 は、マネージドインスタンスの作成に使用された起動テンプレートと、マネージドインスタンスが属する Amazon EC2 Fleet を示すシステムタグを追加します。

カスタムタグ

キャパシティプロバイダーにタグを追加し、propagateTags プロパティを使用してタグ伝播を有効にすることで、Amazon ECS マネージドインスタンスにカスタムタグを追加できます。キャパシティプロバイダーのタグは、キャパシティプロバイダーが起動するマネージドインスタンスに伝播されます。

次のキャパシティプロバイダー定義の例では、propagateTags に CAPACITY_PROVIDER の値を使用してキャパシティプロバイダーを作成する場合、キャパシティプロバイダーからタグを指定して伝播する方法を示しています。

```
{
    "name": "my-cluster-managed-instances-cp",
    "cluster": "my-cluster",
     "tags": [
                {
                "key":"tag_key",
                "value":"tag_value"
                }
            ],
    "managedInstancesProvider": {
        "infrastructureRoleArn": "arn:aws:iam::123456789012:role/
ecsInfrastructureRole",
        "propagateTags": "CAPACITY_PROVIDER",
        "instanceLaunchTemplate": {
            "ec2InstanceProfileArn": "arn:aws:iam::123456789012:instance-profile/
ecsInstanceProfile",
            "networkConfiguration": {
                "subnets": [
                    "subnet-abcdef01234567",
                    "subnet-bcdefa98765432"
                ],
                "securityGroups": [
                    "sg-0123456789abcdef"
                ]
            }
        }
    }
}
```

> **Note**
>
> - キャパシティプロバイダーに新しいタグを追加した場合、新しく追加されたタグは既存のインスタンスには伝播されませんが、新しく作成されたインスタンスには伝播されます。
> - お客様が EC2 インスタンスにタグを直接追加することも可能ですが、それらのタグはキャパシティプロバイダーに伝播されません。
> - 既存のリソースにタグを追加する (AWS CLI) または 既存のリソースにタグを追加する (Amazon ECS コンソール) を使用して、キャパシティプロバイダーにタグ付けする、またはタグを削除することもできます。

Amazon ECS マネージドインスタンスキャパシティプロバイダーの詳細については、「Amazon ECS マネージドインスタンスのキャパシティプロバイダー」を参照してください。

Amazon ECS の Amazon EC2 コンテナインスタンスにタグを追加する

以下のいずれかの方法で、Amazon ECS の Amazon EC2 コンテナインスタンスにタグを関連付けることができます。

- 方法 1 – Amazon EC2、API、CLI、またはコンソールを使用してコンテナインスタンスを作成するときに、コンテナエージェントの設定パラメータ `ECS_CONTAINER_INSTANCE_TAGS` を使用してインスタンスにユーザーデータを渡すことで、タグを指定します。これにより、Amazon ECS のコンテナインスタンスにのみ関連付けられているタグが作成され、Amazon EC2 API では一覧表示できなくなります。詳細については、「Amazon ECS Linux コンテナインスタンスをブートストラップしてデータを渡す」を参照してください。

  > **Important**
  >
  > Amazon EC2 Auto Scaling グループを使用してコンテナインスタンスを起動する場合は、ECS_CONTAINER_INSTANCE_TAGS エージェント設定パラメータを使用してタグを追加する必要があります。これは、Auto Scaling グループを使用して起動される Amazon EC2 インスタンスにタグを追加する方法によるものです。

  コンテナインスタンスにタグを関連付けるユーザーデータスクリプトの例を次に示します。

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_CONTAINER_INSTANCE_TAGS={"tag_key": "tag_value"}
EOF
```

- 方法 2 – Amazon EC2 API、CLI、またはコンソールを使用してコンテナインスタンスを作成する場合、最初に TagSpecification.N パラメータを使用してタグを指定します。その後、コンテナエージェント設定パラメータ ECS_CONTAINER_INSTANCE_PROPAGATE_TAGS_FROM を使用して、ユーザーデータをインスタンスに渡します。これにより、Amazon EC2 から Amazon ECS に伝播されます。

  以下に、Amazon EC2 インスタンスに関連付けられたタグを伝播し、さらに MyCluster という名前のクラスターでインスタンスを登録するユーザーデータスクリプトの例を示します。

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_CONTAINER_INSTANCE_PROPAGATE_TAGS_FROM=ec2_instance
EOF
```

  コンテナインスタンスのタグが Amazon EC2 から Amazon ECS に伝達されることを許可するには、以下のアクセス許可を コンテナインスタンスの IAM ロールにインラインポリシーとして手動で追加します。詳細については、「IAM ポリシーの追加と削除」を参照してください。

  - ec2:DescribeTags

  これらのアクセス許可を追加するために使用される、ポリシーの例を次に示します。

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
             "ec2:DescribeTags"
            ],
            "Resource": "*"
        }
```

```
    ]
}
```

Amazon ECS の外部コンテナインスタンスにタグを追加する

次のいずれかの方法を使用して、Amazon ECS の外部コンテナインスタンスに関連するタグを付けることができます。

- 方法 1 — インストールスクリプトを実行して外部インスタンスをクラスターに登録する前に、Amazon ECS コンテナエージェント設定ファイル (`/etc/ecs/ecs.config`を追加し`ECS_CONTAINER_INSTANCE_TAGS`コンテナエージェント設定パラメーター。これにより、外部インスタンスに関連付けられているタグが作成されます。

  構文の例を次に示します。

  ```
  ECS_CONTAINER_INSTANCE_TAGS={"tag_key": "tag_value"}
  ```

- 方法 2 — 外部インスタンスがクラスターに登録された後、AWS マネジメントコンソール を使用して、タグを追加できます。詳細については、「既存のリソースにタグを追加する (Amazon ECS コンソール)」を参照してください。

Amazon ECS 使用状況レポート

AWS は、Amazon ECS リソースのコストおよび使用量を分析するために使用できる、Cost Explorer と呼ばれるレポートツールを提供します。

Cost Explorer を使用して、使用状況とコストのグラフを表示できます。過去 13 か月からデータを表示でき、また次の 3 か月間にどのくらい使用する可能性があるかを予測します。Cost Explorer を使用すると、時間の経過とともに AWS リソースに費やす金額のパターンを確認できます。例えば、Cost Explorer を使用して、さらに調べる必要がある分野を特定し、コストを把握するために使用できる傾向を確認できます。データの時間範囲を指定したり、時間データを日または月ごとに表示することもできます。

コストと使用状況レポートの計測データには、すべての Amazon ECS タスクの使用状況が示されます。計測データには、実行された各タスクの CPU 使用量が `vCPU-Hours` として、メモリ使用量が `GB-Hours` として含まれます。データの表示方法は、タスクのコンピューティングオプションによって異なります。

Fargate を使用するタスクの場合、lineItem/Operation 列には FargateTask が表示され、各タスクに関連するコストを確認できます。

EC2 を使用するタスクの場合、lineItem/Operation 列には ECSTask-EC2 が表示されます。該当のタスクには、直接関連するコストはありません。メモリの使用状況など、レポートに表示される計測データは、指定された請求期間にタスクが予約したリソースの合計を表します。このデータを使用して、Amazon EC2 インスタンスの基盤となるクラスターのコストを決定できます。Amazon EC2 インスタンスのコストと使用状況データは、Amazon EC2 サービスの下に個別に示されます。

Amazon ECS マネージドタグを使用して各タスクが属するサービスやクラスターを識別することもできます。詳細については、「請求にタグを使用する」を参照してください。

> **Important**
>
> 計測データは、2018 年 11 月 16 日以降に起動したタスクでのみ表示されます。この日付より前に起動したタスクには、計測データが表示されません。

Cost Explorer でコスト配分データを並べ替えるために使用できるフィールドの例を次に示します。

- クラスター名
- サービス名
- リソースタグ
- 起動タイプ
- AWS リージョン
- 使用タイプ

AWSのコストと使用状況レポートの作成の詳細については、AWS BillingユーザーガイドのAWSのコストと使用状況レポートを参照してください。

タスクレベルのコストと使用状況レポート

AWS Cost Management は、Fargate のタスクや EC2 のタスクを含む、Amazon ECS の各タスクの AWS Cost and Usage Report における CPU とメモリの使用状況データを提供できます。このデータはコスト配分データの分割と呼ばれます。このデータを使用して、アプリケーションのコストと使用状況を分析できます。さらに、コスト配分タグとコストカテゴリを使用して、個々のビジネスユニットやチームにコストを分割して割り当てることができます。コスト配分データの分割の詳細について

は、「AWS Cost and Usage Report ユーザーガイド」の「分割コスト配分データを理解する」を参照してください。

AWS Cost Management Console で、アカウントのためにタスクレベルのコスト配分データの分割をオプトインできます。管理 (支払者) アカウントをお持ちの場合は、支払者アカウントからオプトインして、すべてのリンクされたアカウントにこの設定を適用できます。

コスト配分データの分割を設定すると、レポートの splitLineItem ヘッダーの下に追加の列が表示されます。詳細については、「AWS Cost and Usage Report ユーザーガイド」の「明細の分割の詳細」を参照してください。

EC2 のタスクの場合、このデータは、リソースの使用状況または予約、およびインスタンスの残りのリソースに基づいて、EC2 インスタンスのコストを分割します。

前提条件は次のとおりです。

- `ECS_DISABLE_METRICS` Amazon ECS エージェント設定パラメータを `false` に設定します。

  この設定が `false` である場合、Amazon ECS エージェントはメトリクスを Amazon CloudWatch に送信します。Linux では、この設定はデフォルトで `false` であり、メトリクスは CloudWatch に送信されます。Windows では、この設定はデフォルトで `true` であるため、メトリクスを CloudWatch に送信して AWS Cost Management が使用できるようにするためには、設定を `false` に変更する必要があります。ECS エージェント設定の詳細については、「Amazon ECS コンテナエージェントの設定」を参照してください。
- 信頼できるメトリクスの最小 Docker バージョンは、Amazon ECS 最適化 AMI 20220607 以降に含まれる Docker バージョン v20.10.13 以降です。

コスト配分データの分割を使用するには、レポートを作成し、[コスト配分データの分割] を選択する必要があります。詳細については、「AWS Cost and Usage Report ユーザーガイド」の「コストと使用状況レポートを作成する」を参照してください。

AWS Cost Management は、タスクの CPU とメモリの使用状況を使用してコスト配分データの分割を計算します。使用状況が利用できない場合、AWS Cost Management は、使用状況の代わりにタスクの CPU とメモリの予約を使用できます。CUR が予約を使用していることがわかった場合は、コンテナインスタンスが前提条件を満たしていること、およびタスクリソースの使用状況メトリクスが CloudWatch に表示されることを確認します。

Amazon ECS のモニタリング

モニタリングは、Amazon ECS および AWS ソリューションの信頼性、可用性、パフォーマンスを維持する上で重要な部分です。マルチポイント障害が発生した場合は、その障害をより簡単にデバッグできるように、AWS ソリューションのすべての部分からモニタリングデータを収集する必要があります。Amazon ECS のモニタリングを開始する前に、以下の質問に対する回答を盛り込んだモニタリング計画を作成する必要があります。

- どのような目的でモニタリングしますか?
- どのリソースをモニタリングしますか?
- どのくらいの頻度でこれらのリソースをモニタリングしますか?
- どのモニタリングツールを利用しますか?
- 誰がモニタリングタスクを実行しますか?
- 問題が発生したときに誰が通知を受け取りますか?

使用可能になるメトリクスは、クラスター内のタスクとサービスのコンピューティングオプションに応じて異なります。サービスに Fargate を使用している場合、サービスの監視を支援するための CPU とメモリの使用状況メトリクスが提供されます。EC2 の場合、基盤となるインフラストラクチャを構成する EC2 インスタンスを所有し、それらのインスタンスをモニタリングする必要があります。CPU とメモリの予約率と使用率の追加のメトリクスは、クラスター、サービス、およびタスクで使用できます。

次のステップでは、さまざまなタイミングと負荷条件でパフォーマンスを測定することにより、お客様の環境で通常の Amazon ECS のパフォーマンスのベースラインを確定します。Amazon ECS のモニタリングでは、過去のモニタリングデータを保存し、現在のパフォーマンスデータと比較することで、パフォーマンスの通常パターンと異常パターンを特定し、問題に対処する方法を考案できます。

ベースラインを確立するには、少なくとも、次の項目をモニタリングする必要があります。

- Amazon ECS クラスターの CPU とメモリの予約率および使用率に関するメトリクス
- Amazon ECS サービスの CPU とメモリの使用率メトリクス

詳細については、「Amazon ECS メトリクスの表示」を参照してください。

Amazon ECS モニタリングのベストプラクティス

Amazon ECS をモニタリングするには、次のベストプラクティスを使用します。

- モニタリングに優先順位を設定し、小さな問題が大きな問題に発展する前に阻止する
- 以下の質問に対する回答を盛り込んだモニタリング計画を作成する
  - どのような目的でモニタリングしますか?
  - どのリソースをモニタリングしますか?
  - どのくらいの頻度でこれらのリソースをモニタリングしますか?
  - どのモニタリングツールを利用しますか?
  - 誰がモニタリングタスクを実行しますか?
  - 問題が発生したときに誰が通知を受け取りますか?
- モニタリングは可能な限り自動化します。
- Amazon ECS ログファイルを確認します。詳細については、「Amazon ECS コンテナエージェントログの表示」を参照してください。
- Runtime Monitoring を使用すると、 AWS 環境内のアカウント、コンテナ、ワークロード、およびデータを保護することができます。詳細については、「ランタイムモニタリングを使用して不正な動作を特定する」を参照してください。

Amazon ECS のモニタリングツール

AWS は、Amazon ECS のモニタリングに使用できるさまざまなツールを提供します。これらのツールの一部はモニタリングを行うように設定できますが、一部のツールは手動による介入が必要です。モニタリングタスクをできるだけ自動化することをお勧めします。

自動モニタリングツール

以下の自動化されたモニタリングツールを使用して、Amazon ECS を監視し、問題が発生したときにレポートできます。

- Amazon CloudWatch のアラーム – 単一のメトリクスを指定した期間モニタリングし、特定のしきい値に対する複数の期間にわたるメトリクスの値に基づいて、1 つ以上のアクションを実行します。アクションは、Amazon Simple Notification Service (Amazon SNS) のトピックまたは Amazon EC2 Auto Scaling のポリシーに送信される通知です。CloudWatch アラームは、特定の状態にあるという理由だけでアクションを呼び出すことはありません。状態が変更され、指定された期間維持

されている必要があります。詳細については、「CloudWatch を使用して Amazon ECS をモニタリングする」を参照してください。

Fargate を使用するタスクがあるサービスでは、CloudWatch アラームを使用して、CPU やメモリの使用率などの CloudWatch メトリクスに基づいてサービス内のタスクをスケールインおよびスケールアウトできます。詳細については、「Amazon ECS サービスを自動的にスケールする」を参照してください。

EC2 を使用するタスクまたはサービスがあるクラスターでは、CloudWatch アラームを使用して、クラスターのメモリ使用率などの CloudWatch メトリクスに基づいてコンテナインスタンスをスケールインおよびスケールアウトできます。

Amazon ECS に最適化された Amazon Linux AMI で起動されたコンテナインスタンスの場合は、CloudWatch Logs を使用してコンテナインスタンスのさまざまなログを 1 つの便利な場所で表示できます。CloudWatch エージェントをコンテナインスタンスにインストールする必要があります。詳細については、Amazon CloudWatch ユーザーガイドの「コマンドラインを使用して CloudWatch エージェントをダウンロードおよび設定する」を参照してください。また、`ecsInstanceRole` ロールに `ECS-CloudWatchLogs` ポリシーを追加する必要があります。詳細については、「コンテナインスタンスのモニタリングに必要なアクセス許可」を参照してください。

- Amazon CloudWatch Logs – タスク定義で`awslogs` ログドライバーを指定することで、Amazon ECS タスクのコンテナからのログファイルをモニタリング、保存、およびアクセスできます。詳細については、「Amazon ECS ログを CloudWatch に送信する」を参照してください。

  Amazon ECS コンテナインスタンスからのオペレーティングシステムおよび Amazon ECS コンテナエージェントのログファイルをモニタリング、保存、アクセスすることもできます。このログへのアクセス方法は、EC2 を使用するコンテナの場合に使うことができます。

- Amazon CloudWatch Events - イベントに一致したものを 1 つ以上のターゲットの関数またはストリームに渡して、変更、状態の情報の収集、是正措置を行います。詳細については、このガイドの「EventBridge を使用して Amazon ECS エラーへの対応を自動化する」および「Amazon EventBridge ユーザーガイド」の「EventBridge is the evolution of Amazon CloudWatch Events」を参照してください。

- Container Insights – コンテナ化されたアプリケーションとマイクロサービスからメトリクスとログを収集、集計、要約します。Container Insights は、埋め込みメトリクス形式を使用して、パフォーマンスログイベントとしてデータを収集します。このパフォーマンスログイベントは、高濃度データを取り込み、大規模に格納することが可能な構造化された JSON スキーマを使用するエントリです。CloudWatch は、このデータから、クラスター、タスク、サービスのレベルで

CloudWatch メトリクスとして集計されたメトリクスを作成します。Container Insights が収集するメトリクスは、CloudWatch 自動ダッシュボードで使用でき、CloudWatch コンソールの [メトリクス] セクションでも表示できます。

- AWS CloudTrail のログのモニタリング – アカウント間でログファイルを共有し、CloudTrail のログファイルを CloudWatch Logs に送信してリアルタイムでモニタリングします。また、ログを処理するアプリケーションを Java で作成し、CloudTrail からの提供後にログファイルが変更されていないことを確認します。詳細については、「AWS CloudTrail を使用して Amazon ECS API コールを ログに記録する」と、AWS CloudTrail ユーザーガイドの「CloudTrail ログファイルの操作」を参照してください。
- ランタイムモニタリング – AWS 環境内のクラスターとコンテナの脅威を検出します。Runtime Monitoring では、ファイルアクセス、プロセス実行、ネットワーク接続などの個々の Amazon ECS ワークロードを実行時に可視化する、GuardDuty セキュリティエージェントを使用します。

手動モニタリングツール

Amazon ECS のモニタリングでもう 1 つ重要な点は、CloudWatch のアラームの対象外の項目を手動でモニタリングすることです。Trusted Advisor、CloudWatch、その他の AWS コンソールのダッシュボードには、AWS 環境の状態が一目でわかるように表示されます。コンテナインスタンスおよびタスクのコンテナのログファイルも確認することをお勧めします。

- Amazon ECS コンソール:
  - EC2 のクラスターメトリクス
  - サービスメトリクス
  - サービスのヘルスステータス
  - サービスデプロイイベント
- CloudWatch のホームページ:
  - 現在のアラームとステータス
  - アラームとリソースのグラフ
  - サービスのヘルスステータス

  また、CloudWatch を使用して以下のことを行えます。
  - 重要なサービスをモニタリングするために カスタマイズされたダッシュボード を作成する。
  - メトリクスデータをグラフ化して、問題のトラブルシューティングを行い、傾向を確認する。
  - AWS リソースのすべてのメトリクスを検索およびブラウズする。

  - 問題があることを通知するアラームを作成/編集する。
- コンテナのヘルスチェック - コンテナ上でローカルに実行され、アプリケーションのヘルスと可用性を検証するコマンドです。これは、タスク定義でコンテナごとに構成します。
- AWS Trusted Advisor は、AWS リソースのパフォーマンス、信頼性、セキュリティ、費用効率を向上するためのモニタリングに役立ちます。すべてのユーザーは、4 つの Trusted Advisor; チェックを利用できます。ビジネスまたはエンタープライズサポートプランのユーザーは、50 以上のチェックを利用できます。詳細については、 を参照してください。AWS Trusted Advisor

  Trusted Advisor には Amazon ECS に関連する以下のチェックがあります。

  - 1 つのアベイラビリティーゾーンでサービスが実行されていることを示す耐障害性。
  - 複数のアベイラビリティーゾーンでスプレッド配置戦略を使用していないことを示す耐障害性。
- AWS Compute Optimizer は、AWS リソースの設定と使用率のメトリクスを分析するサービスです。Compute Optimizer は、リソースが最適かどうかを報告し、最適化に関するレコメンデーションを生成してコストを削減およびワークロードのパフォーマンスを改善します。

  詳細については、「AWS Compute OptimizerAmazon ECS に関する 推奨事項」を参照してください。
- Amazon ECS MCP サーバー – AI アシスタントを使用して、自然言語を用いたデプロイのモニタリング、コンテナ問題のトラブルシューティング、設定の検査を行います。MCP サーバーは、サービスの正常性をリアルタイムで可視化し、タスクの失敗を分析して、問題解決を加速します。詳細については、「Amazon ECS MCP サーバー」を参照してください。

CloudWatch を使用して Amazon ECS をモニタリングする

Amazon CloudWatch を使用して Amazon ECS リソースモニタリングすることで、Amazon ECSからraw データを収集して、リアルタイムに近い読み取り可能なメトリクスに加工することができます。これらの統計情報は 2 週間単位で記録されるため、履歴情報にアクセスしてクラスターやサービスの動作をより的確に把握できます。Amazon ECS のメトリクスデータは 1 分間隔で自動的に CloudWatch に送信されます。CloudWatch の詳細については、「Amazon CloudWatch ユーザーガイド」を参照してください。

Amazon ECS には、クラスターとサービスの無料メトリクスがあります。追加料金で、CPU、メモリ、EBS ファイルシステムの使用率など、タスクごとのメトリクスについて、クラスターに対して Amazon ECS CloudWatch Container Insights を有効にすることができます。コンテナインサイトの詳細については、オブザーバビリティが強化された Container Insights を使用し、Amazon ECS コンテナを監視する を参照してください。

考慮事項

Amazon ECS CloudWatch メトリクスを使用する際には、以下の点を考慮する必要があります。

- Fargate でホストされるどの Amazon ECS サービスでも、CloudWatch の CPU とメモリの使用率メトリクスが自動的に備わっているため、手動の手順を踏む必要はありません。

- Amazon EC2 インスタンスでホストされているどの Amazon ECS タスクまたはサービスでも、CloudWatch メトリクスを生成するために、Amazon EC2 インスタンスにコンテナエージェントのバージョン `1.4.0` 以降 (Linux) または `1.0.0` 以降 (Windows) が必要です。ただし、最新のコンテナエージェントのバージョンを使用することをお勧めします。エージェントのバージョンの確認と最新バージョンへの更新については、「Amazon ECS コンテナエージェントをアップデートする」を参照してください。
- 信頼性の高い CloudWatch メトリクスの最低 Docker バージョンは Docker `20.10.13` バージョン以降です。
- Amazon EC2 インスタンスには、IAM ロールに Amazon EC2 インスタンスを起動するのに使用する `ecs:StartTelemetrySession` アクセス許可も必要です。CloudWatch メトリクスが Amazon ECS で使用可能になる前に Amazon ECS コンテナインスタンス IAM ロールを作成した場合は、このアクセス許可の追加が必要になる場合があります。コンテナインスタンス IAM ロールとコンテナインスタンスのマネージド IAM ポリシーのアタッチについては、「Amazon ECS コンテナインスタンスの IAM ロール」を参照してください。
- Amazon ECS コンテナエージェント設定で `ECS_DISABLE_METRICS=true` を設定することで、Amazon EC2 インスタンスでの CloudWatch メトリクス収集を無効にできます。詳細については、「Amazon ECS コンテナエージェントの設定」を参照してください。

推奨メトリクス

Amazon ECS には、リソースをモニタリングするのに使用できる無料 CloudWatch メトリクスが用意されています。このメトリクスを使用して、クラスター全体の CPU とメモリの予約率と CPU、メモリ、EBS ファイルシステムの使用率、およびクラスター内のサービスの CPU、メモリ、EBS ファイルシステムの使用率を測定できます。GPU のワークロードについては、クラスター全体の GPU 予約率を測定できます。

Amazon ECS タスクがクラスターでホストされているインフラストラクチャによって、利用可能なメトリクスが決まります。Fargate インフラストラクチャでホストされているタスクでは、サービスのモニタリングを支援するために、Amazon ECS より CPU、メモリ、EBS ファイルシステムの使用

率メトリクスが提供されます。EC2 インスタンスでホストされるタスクでは、CPU、メモリ、GPU の予約率メトリクス、および CPU とメモリの使用率メトリクスが、クラスターおよびサービスレベルで Amazon ECS により提供されます。基盤となるインフラストラクチャを構成する Amazon EC2 インスタンスを個別にモニタリングする必要があります。Amazon EC2 インスタンスのモニタリングの詳細については、「Amazon EC2 ユーザーガイド」の「Amazon EC2 のモニタリング」を参照してください。

Amazon ECS で使用する推奨アラームについては、「Amazon CloudWatch Logs ユーザーガイド」にある以下のいずれかを参照してください。

- Amazon ECS
- Container Insights を使用する Amazon ECS

Amazon ECS メトリクスの表示

クラスターでリソースを実行したら、Amazon ECS と CloudWatch のコンソールでメトリクスを表示できます。Amazon ECS コンソールでは、クラスターとサービスのメトリクスの 24 時間の最大、最小、平均が表示されます。CloudWatch コンソールでは、リソースを詳細でカスタマイズ可能な表示で示します。また、サービスの実行中タスク数も表示します。

Amazon ECS コンソール

Amazon ECS サービスの CPU とメモリの使用率メトリクスを Amazon ECS サービス コンソールで使用できます。サービスメトリクスについてのビューには、過去 24 時間の平均値、最小値、最大値が、5 分ごとのデータポイントとともに表示されます。詳細については、「Amazon ECS サービスの使用率メトリクス」を参照してください。

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. メトリクスを表示するクラスターを選択します。
3. 表示するメトリクスを決定します。

メトリクスの表示元	Steps
クラスター	クラスターの詳細のページで、[メトリクス] タブを選択します。CloudWatch コンソールには、有効になってい

メトリクスの表示元	Steps
	る場合に CloudWatch コンテナインサイトのメトリクスを表示できるリンクも用意されています。
サービス	クラスターの詳細のページの [サービス] タブで、目的のサービスを選択します。このメトリクスは、[正常性とメトリクス] タブで利用可能になります。

CloudWatch コンソール

Fargate では、Amazon ECS サービスのメトリクスも CloudWatch コンソールに表示できます。コンソールで Amazon ECS メトリクスの最も詳細なビューを表示できます。必要に応じてこのビューをカスタマイズできます。サービス使用率とサービス実行中のタスク数を表示できます。

EC2 キャパシティプロバイダーでは、Amazon ECS クラスターとサービスのメトリクスも CloudWatch コンソールに表示できます。コンソールで Amazon ECS メトリクスの最も詳細なビューを表示できます。必要に応じてこのビューをカスタマイズできます。

メトリクスの表示方法の詳細については、「Amazon CloudWatch ユーザーガイド」の「使用可能なメトリクスの表示」を参照してください。

Amazon ECS CloudWatch メトリクス

CloudWatch 使用状況メトリクスを使用して、アカウントのリソースの使用状況を把握できます。これらのメトリクスを使用して、CloudWatch グラフやダッシュボードで現在のサービスの使用状況を可視化できます。

Amazon ECS は、CloudWatch に 1 分間隔でメトリクスを送信します。これらのメトリクスは、`RUNNING` 状態のタスクを持つリソースに対して収集されます。クラスター、サービス、またはその他のリソースに実行中のタスクがない場合、その期間中、そのリソースのメトリクスは報告されません。例えば、クラスターに 1 つのサービスがあるが、このサービスに `RUNNING` 状態のタスクがない場合、CloudWatch に送信されるメトリクスはありません。同様に、2 つのサービスがあり、1

つに実行中のタスクがあるが、別の 1 つに実行中のタスクがない場合、実行中のタスクがあるサービスのメトリクスのみが送信されます。

メトリクス	説明	有効なディメンション	有用な統計	Unit
CPUReservation	クラスターまたはサービスで予約されている CPU ユニットの割合。 CPU の予約率 (`ClusterName` でフィルタリング) は、クラスター上の Amazon ECS タスクによって予約されている CPU ユニットの総数を、クラスターに登録されているすべての Amazon EC2 インスタンスの CPU ユニットの総数で割った数で測定されます。`ACTIVE` または `DRAINING` ステータスの Amazon EC2 インスタンスのみが、CPU 予約率メトリクスに影響します。	`ClusterName`	Average、Minimum、Maximum	割合 (%)

メトリクス	説明	有効なディメンション	有用な統計	Unit
	このメトリクスは、Amazon EC2 インスタンスでホストされるタスクでのみサポートされています。			

メトリクス	説明	有効なディメンション	有用な統計	Unit
CPUUtilization	クラスターまたはサービスまたはマネージドデーモンで使用されている CPU ユニットの割合。 クラスターレベルの CPU 使用率 (ClusterName でフィルタリング) は、クラスター上の Amazon ECS タスクによって使用されている CPU ユニットの総数を、クラスターに登録されているすべての Amazon EC2 インスタンスの CPU ユニットの総数で割った値で測定されます。ACTIVE または DRAINING ステータスの Amazon EC2 インスタンスのみが、CPU 予約率メトリクスに影響します。	ClusterName , ServiceName	Average、Minimum、Maximum	割合 (%)

メトリクス	説明	有効なディメンション	有用な統計	Unit
	クラスターレベルのメトリクスは、Amazon EC2 インスタンスでホストされるタスクでのみサポートされています。 サービスレベルの CPU 使用率 (`ClusterName` 、`ServiceName` でフィルタリング) は、サービスに属するタスクによって使用されている CPU ユニットの総数を、サービスに属するタスク用に予約されている CPU ユニットの総数で割った値で測定されます。サービスレベルのメトリクスは、Amazon EC2 インスタンスと Fargate でホストされているタスクでサポートされています。			

メトリクス	説明	有効なディメンション	有用な統計	Unit
	デーモンレベルの CPU 使用率 (`ClusterName` 、`ServiceName` でフィルタリング) は、マネージドデーモンに属するタスクによって使用されている CPU ユニットの総数を、マネージドデーモンに属するタスク用に予約されている CPU ユニットの総数で割った値で測定されます。			

メトリクス	説明	有効なディメンション	有用な統計	Unit
MemoryReservation	クラスターでタスクを実行することで予約されているメモリの割合。 クラスターのメモリ予約率は、クラスター上の Amazon ECS タスクによって予約されているメモリの合計を、クラスターに登録されているすべての Amazon EC2 インスタンスのメモリの合計で割った値で測定されます。このメトリクスは ClusterName でのみフィルタリング可能です。ACTIVE または DRAINING ステータスの Amazon EC2 インスタンスのみが、メモリ予約率メトリクスに影響します。クラスター	ClusterName .	Average、Minimum、Maximum	割合 (%)

メトリクス	説明	有効なディメンション	有用な統計	Unit
	レベルのメモリ予約率メトリクスは、Amazon EC2 インスタンスでホストされるタスクでのみサポートされています。 Note メモリ使用量率を計算する際、MemoryReservation が指定されると、合計メモリの代わりにその値が計算に使用されます。			

メトリクス	説明	有効なディメンション	有用な統計	Unit
MemoryUtilization	クラスターまたはサービスまたはマネージドデーモンで使用されているメモリの割合。 クラスターレベルのメモリ使用率 (ClusterName でフィルタリング) は、クラスター上の Amazon ECS タスクによって使用されているメモリの合計を、クラスターに登録されているすべての Amazon EC2 インスタンスメモリの合計で割った値で測定されます。ACTIVE または DRAINING ステータスの Amazon EC2 インスタンスのみが、メモリ使用率メトリクスに影響します。クラスターレベルのメトリク	ClusterName , ServiceName	Average、Minimum、Maximum	割合 (%)

メトリクス	説明	有効なディメンション	有用な統計	Unit
	スは、Amazon EC2 インスタンスでホストされるタスクでのみサポートされています。 サービスレベルのメモリ使用率は (`ClusterName` 、`ServiceName` でフィルタリング)、サービスに属するタスクによって使用されているメモリの合計を、サービスに属するタスク用に予約されているメモリの合計で割った値で測定されます。サービスレベルのメトリクスは、Amazon EC2 インスタンスと Fargate でホストされているタスクでサポートされています。			

メトリクス	説明	有効なディメンション	有用な統計	Unit
	デーモンレベルのメモリ使用率は (`ClusterName` 、`ServiceName` でフィルタリング)、マネージドデーモンに属するタスクによって使用されているメモリの合計を、マネージドデーモンに属するタスク用に予約されているメモリの合計で割った値で測定されます。			

メトリクス	説明	有効なディメンション	有用な統計	Unit
EBSFilesystemUtilization	サービス内のタスクによって使用されている Amazon EBS ファイルシステムの割合。 サービスレベルの EBS ファイルシステム使用率メトリクス (ClusterName 、ServiceName でフィルタリング) は、サービスに属するタスクによって使用されている EBS ファイルシステムの合計を、サービスに属するすべてのタスクに割り当てられている EBS ファイルシステムストレージの合計で割った値で測定されます。サービスレベルの EBS ファイルシステム使用率メトリクスは、EBS ボリュームがア	ClusterName , ServiceName	Average、Minimum、Maximum	割合 (%)

メトリクス	説明	有効なディメンション	有用な統計	Unit
	タッチされている Amazon EC2 インスタンス (コンテナエージェントバージョン `1.79.0`) と Fargate (プラットフォームバージョン `1.4.0`) でホストされているタスクでのみ使用できます。 Note Fargate でホストされているタスクでは、Fargate によってのみ使用されるディスク上のスペースがあります。Fargate が使用するスペースにコス			

メトリクス	説明	有効なディメンション	有用な統計	Unit
	トは発生しませんが、df のようなツールで使用されるこの追加ストレージが表示されます。			

メトリクス	説明	有効なディメンション	有用な統計	Unit
GPUReservation	使用可能な GPUs の合計に対する、クラスター内の実行中のタスクによって予約されている GPU の割合。 クラスターレベルの GPU 予約率メトリクスは、クラスター上の Amazon ECS タスクによって予約されている GPU の数を、クラスターに登録された GPU を備えたすべての Amazon EC2 インスタンスで使用可能な GPU の総数で割った値で測定されます。ACTIVE または DRAINING ステータスの Amazon EC2 インスタンスのみが、GPU 予約率メトリクスに影響します。	ClusterName	Average、Minimum、Maximum	割合 (%)

メトリクス	説明	有効なディメンション	有用な統計	Unit
ActiveConnectionCount	選択した DiscoveryName を共有するタスクで実行される、クライアントから Amazon ECS Service Connect プロキシへの、アクティブな同時接続の総数。 このメトリクスは、Amazon ECS Service Connect を設定した場合にのみ使用できます。 有効なディメンション: DiscoveryName および DiscoveryName, ServiceName, ClusterName 。	DiscoveryName と DiscoveryName 、ServiceName 、ClusterName	Average、Minimum、Maximum、Sum	カウント

メトリクス	説明	有効なディメンション	有用な統計	Unit
NewConnectionCount	選択した DiscoveryName を共有するタスクで実行される、クライアントから Amazon ECS Service Connect プロキシに確立された新しい接続の総数。 このメトリクスは、Amazon ECS Service Connect を設定した場合にのみ使用できます。	DiscoveryName および DiscoveryName, ServiceName, ClusterName	Average、Minimum、Maximum、Sum	カウント
ProcessedBytes	Service Connect プロキシによって処理されたインバウンドトラフィックの総バイト数。 このメトリクスは、Amazon ECS Service Connect を設定した場合にのみ使用できます。	DiscoveryName および DiscoveryName, ServiceName, ClusterName	Average、Minimum、Maximum、Sum	バイト

メトリクス	説明	有効なディメンション	有用な統計	Unit
RequestCount	Service Connect プロキシによって処理されたインバウンドトラフィックリクエストの数。 このメトリクスは、Amazon ECS Service Connect を設定した場合にのみ使用できます。 また、タスク定義のポートマッピングで appProtocol を設定する必要があります。	DiscoveryName および DiscoveryName, ServiceName, ClusterName	Average、Minimum、Maximum、Sum	カウント

メトリクス	説明	有効なディメンション	有用な統計	Unit
GrpcRequestCount	Service Connect プロキシによって処理された gRPC インバウンドトラフィックリクエストの数。 このメトリクスは、Amazon ECS Service Connect を設定していて、タスク定義のポートマッピングで appProtocol が GRPC である場合にのみ使用できます。	DiscoveryName および DiscoveryName, ServiceName, ClusterName	Average、Minimum、Maximum、Sum	カウント

メトリクス	説明	有効なディメンション	有用な統計	Unit
HTTPCode_Target_2XX_Count	これらのタスクによってアプリケーションによって生成された 200 から 299 の数字の HTTP レスポンスコードの数。これらのタスクがターゲットです。このメトリクスは、これらのタスクでアプリケーションによって Service Connect プロキシに送信された応答のみをカウントし、直接送信された応答はカウントしません。 このメトリクスは、Amazon ECS Service Connect を設定していて、タスク定義のポートマッピングで appProtocol が HTTP または HTTP2 である場	TargetDiscoveryName および TargetDiscoveryName, ServiceName, ClusterName	Average、Minimum、Maximum、Sum	カウント

メトリクス	説明	有効なディメンション	有用な統計	Unit
	合にのみ使用できます。 有効なディメンション:			

メトリクス	説明	有効なディメンション	有用な統計	Unit
HTTPCode_Target_3XX_Count	これらのタスクによってアプリケーションによって生成された 300 から 399 の数字の HTTP レスポンスコードの数。これらのタスクがターゲットです。このメトリクスは、これらのタスクでアプリケーションによって Service Connect プロキシに送信された応答のみをカウントし、直接送信された応答はカウントしません。 このメトリクスは、Amazon ECS Service Connect を設定していて、タスク定義のポートマッピングで appProtocol が HTTP または HTTP2 である場	TargetDiscoveryName および TargetDiscoveryName, ServiceName, ClusterName	Average、Minimum、Maximum、Sum	カウント

メトリクス	説明	有効なディメンション	有用な統計	Unit
	合にのみ使用できます。			

メトリクス	説明	有効なディメンション	有用な統計	Unit
HTTPCode_Target_4XX_Count	これらのタスクによってアプリケーションによって生成された 400 から 499 の数字の HTTP レスポンスコードの数。これらのタスクがターゲットです。このメトリクスは、これらのタスクでアプリケーションによって Service Connect プロキシに送信された応答のみをカウントし、直接送信された応答はカウントしません。 このメトリクスは、Amazon ECS Service Connect を設定していて、タスク定義のポートマッピングで appProtocol が HTTP または HTTP2 である場	TargetDiscoveryName および TargetDiscoveryName, ServiceName, ClusterName	Average、Minimum、Maximum、Sum	カウント

メトリクス	説明	有効なディメンション	有用な統計	Unit
	合にのみ使用できます。			

メトリクス	説明	有効なディメンション	有用な統計	Unit
HTTPCode_Target_5XX_Count	これらのタスクによってアプリケーションによって生成された 500 から 599 の数字の HTTP レスポンスコードの数。これらのタスクがターゲットです。このメトリクスは、これらのタスクでアプリケーションによって Service Connect プロキシに送信された応答のみをカウントし、直接送信された応答はカウントしません。 このメトリクスは、Amazon ECS Service Connect を設定していて、タスク定義のポートマッピングで appProtocol が HTTP または HTTP2 である場	TargetDiscoveryName および TargetDiscoveryName, ServiceName, ClusterName	Average、Minimum、Maximum、Sum	カウント

メトリクス	説明	有効なディメンション	有用な統計	Unit
	合にのみ使用できます。			
RequestCountPerTarget	選択した DiscoveryName を共有する各ターゲットによって受信されたリクエストの平均数。 このメトリクスは、Amazon ECS Service Connect を設定した場合にのみ使用できます。	TargetDiscoveryName および TargetDiscoveryName, ServiceName, ClusterName	平均	カウント
TargetProcessedBytes	Service Connect プロキシによって処理された総バイト数。 このメトリクスは、Amazon ECS Service Connect を設定した場合にのみ使用できます。	TargetDiscoveryName および TargetDiscoveryName, ServiceName, ClusterName	Average、Minimum、Maximum、Sum	バイト

メトリクス	説明	有効なディメンション	有用な統計	Unit
TargetResponseTime	アプリケーションリクエスト処理のレイテンシー。リクエストがターゲットタスクの Service Connect プロキシに到達してから、ターゲットアプリケーションからの応答がプロキシに受信されるまでの経過時間 (ミリ秒)。 このメトリクスは、Amazon ECS Service Connect を設定した場合にのみ使用できます。	TargetDiscoveryName および TargetDiscoveryName, ServiceName, ClusterName	Average、Minimum、Maximum	ミリ秒

メトリクス	説明	有効なディメンション	有用な統計	Unit
ClientTLSNegotiationErrorCount	TLS 接続に失敗した合計回数。このメトリクスは TLS が有効になっている場合にのみ使用されます。 このメトリクスは、Amazon ECS Service Connect を設定した場合にのみ使用できます。	DiscoveryName と DiscoveryName 、ServiceName 、ClusterName	Average、Minimum、Maximum、Sum	カウント

メトリクス	説明	有効なディメンション	有用な統計	Unit
TargetTLSNegotiationErrorCount	クライアント証明書の欠落、AWS Private CA 検証の失敗、または SAN 検証の失敗により TLS 接続が失敗した合計回数。このメトリクスは TLS が有効になっている場合にのみ使用されます。 このメトリクスは、Amazon ECS Service Connect を設定した場合にのみ使用できます。	ServiceName 、ClusterName 、TargetDiscoveryName および TargetDiscoveryName	Average、Minimum、Maximum、Sum	カウント

メトリクス	説明	有効なディメンション	有用な統計	Unit
LiveTaskCount	サービスの ACTIVATING 、RUNNING、DEACTIVATING ライフサイクル状態にあるタスクの数。 ACTIVATING 状態のタスクは、トラフィックを処理する準備をしています (ロードバランサーターゲットグループへの登録やサービス検出など)。RUNNING 状態のタスクは実行されており、トラフィックの受信とリクエストの処理に使用できます。DEACTIVATING 状態のタスクは、停止する前に接続をドレインし、トラフィック供給リソースから登録解除しています。	ClusterName , ServiceName	Average、Minimum、Maximum	カウント

Amazon ECS メトリクスのディメンション

Amazon ECS メトリックスは AWS/ECS 名前空間を使用し、以下のディメンションのメトリックスを提供しています。Amazon ECS は、タスクが RUNNING 状態にあるリソースのメトリクスのみを送信します。例えば、クラスターに 1 つのサービスがあるが、このサービスに RUNNING 状態のタスクがない場合、CloudWatch に送信されるメトリクスはありません。2 つのサービスがあり、1 つに実行中のタスクがあるが、別の 1 つに実行中のタスクがない場合、実行中のタスクがあるサービスのメトリクスのみが送信されます。

ディメンション	定義
ClusterName	このディメンションにより、指定したクラスター内のすべてのリソースから、リクエストしたデータがフィルタ処理されます。すべての Amazon ECS メトリクスは ClusterName でフィルタ処理されます。
ServiceName	このディメンションにより、指定したクラスター内の特定のサービスまたはデーモンのすべてのリソースから、リクエストしたデータがフィルタ処理されます。デーモンメトリクスでは、ServiceName ディメンション値にはデーモン名が続くプレフィックス daemon: があります。例えば、my-daemon という名前のデーモンには、daemon:my-daemon の ServiceName ディメンション値があります。
DiscoveryName	このディメンションは、トラフィックメトリクスをリクエストするデータを、すべての Amazon ECS クラスター全体の指定された Service Connect 検出名にフィルタリングします。 実行中のコンテナ内の特定のポートには、複数の検出名を指定できることに注意してください。

ディメンション	定義
`DiscoveryName, ServiceName, ClusterName`	このディメンションは、トラフィックメトリクスをリクエストするデータを、このクラスター内のこのサービスによって作成された、この検出名を持つタスク全体の指定された Service Connect 検出名にフィルタリングします。 異なる名前空間の複数のサービスで同じ検出名を再利用した場合、このディメンションを使用して特定のサービスのインバウンドトラフィックメトリクスを確認できます。 実行中のコンテナ内の特定のポートには、複数の検出名を指定できることに注意してください。
`TargetDiscoveryName`	このディメンションは、トラフィックメトリクスをリクエストするデータを、すべての Amazon ECS クラスター全体の指定された Service Connect 検出名にフィルタリングします。 `DiscoveryName` とは異なり、このトラフィックメトリクスは、この名前空間に Service Connect 設定がある他の Amazon ECS タスクから送信される、この `Discovery Name` へのインバウンドトラフィックのみを測定します。これには、クライアントのみ、またはクライアントとサーバーの Service Connect 設定を使用したサービスによって実行されるタスクが含まれます。 実行中のコンテナ内の特定のポートには、複数の検出名を指定できることに注意してください。

ディメンション	定義
`TargetDiscoveryName, ServiceName, ClusterName`	このディメンションは、トラフィックメトリクスにリクエストしたデータを指定された Service Connect 検出名にフィルタリングしますが、このクラスター内のこのサービスによって作成されたタスクからのトラフィックのみをカウントします。 このディメンションを使用して、別のサービスの特定のクライアントからのインバウンドトラフィックメトリクスを確認できます。 `DiscoveryName, ServiceName, ClusterName` とは異なり、このトラフィックメトリクスは、この名前空間に Service Connect 設定がある他の Amazon ECS タスクから送信される、この `DiscoveryName` へのインバウンドトラフィックのみを測定します。これには、クライアントのみ、またはクライアントとサーバーの Service Connect 設定を使用したサービスによって実行されるタスクが含まれます。 実行中のコンテナ内の特定のポートには、複数の検出名を指定できることに注意してください。

AWS Fargate 使用状況メトリクス

CloudWatch 使用状況メトリクスを使用して、アカウントのリソースの使用状況を把握できます。これらのメトリクスを使用して、CloudWatch グラフやダッシュボードで現在のサービスの使用状況を可視化できます。

AWS Fargate 使用状況メトリクスは、AWS のサービスクォータに対応しています。使用量がサービスクォータに近づいたときに警告するアラームを設定することもできます。Fargate サービスクォー

タに関する詳細については、「Amazon Web Services 全般のリファレンス」の「Amazon ECS のエンドポイントとクォータ」を参照してください。

AWS Fargate は、AWS/Usage 名前空間に以下のメトリクスを公開します。

メトリクス	説明
ResourceCount	アカウントで実行されている指定されたリソースの合計数。リソースは、メトリクスに関連付けられたディメンションによって定義されます。

以下のディメンションは、AWS Fargate によって発行される使用状況メトリクスを絞り込むために使用されます。

ディメンション	説明
Service	リソースを含む AWS のサービスの名前。AWS Fargate 使用状況メトリクスの場合、このディメンションの値は Fargate です。
Type	レポートされるエンティティのタイプ。現在、AWS Fargate 使用状況メトリクスの有効な値は Resource のみです。
Resource	実行中のリソースタイプ。実行中のリソースタイプ。現在、AWS Fargate 使用状況メトリクスの有効な値は vCPU のみです。これは、実行中のインスタンスに関する情報を返します。
Class	追跡されているリソースのクラス。追跡されているリソースのクラス。Resource ディメンションの値として vCPU を使用する AWS Fargate 使用状況メトリクスの場合、有効な値は Standard/OnDemand と Standard/Spot です。

Service Quotas コンソールを使用することで、使用状況をグラフで可視化したり、AWS Fargate 使用量がサービスクォータに近づくと警告するアラームを設定したりできます。クォータ値のしきい値に近づいたときに通知する CloudWatch アラームを作成する方法については、「サービスクォータユーザーガイド」の「Service Quotas と Amazon CloudWatch アラーム」を参照してください。

.

Amazon ECS クラスターの予約率メトリクス

クラスターの予約メトリクスは、クラスター内のアクティブな各コンテナインスタンスに登録された CPU、メモリ、GPUs に対する、クラスター内のすべての Amazon ECS タスクによって予約されている CPU、メモリ、GPUs の割合として測定されます。ステータスが ACTIVE または DRAINING であるコンテナインスタンスのみが、クラスター予約メトリクスに影響します。このメトリクスは、EC2 インスタンスでホストされるタスクまたはサービスを含むクラスターでのみサポートされています。AWS Fargate でタスクがホストされているクラスターではサポートされていません。

```
                              (Total CPU units reserved by tasks in cluster) x 100
Cluster CPU reservation =
 ---------------------------------------------------------------
                           (Total CPU units registered by container instances in
 cluster)
```

```
                                 (Total MiB of memory reserved by tasks in cluster x
 100)
Cluster memory reservation =
 ------------------------------------------------------------------
                             (Total MiB of memory registered by container instances in
 cluster)
```

```
                                (Total GPUs reserved by tasks in cluster x 100)
Cluster GPU reservation =
 ------------------------------------------------------------------
                             (Total GPUs registered by container instances in cluster)
```

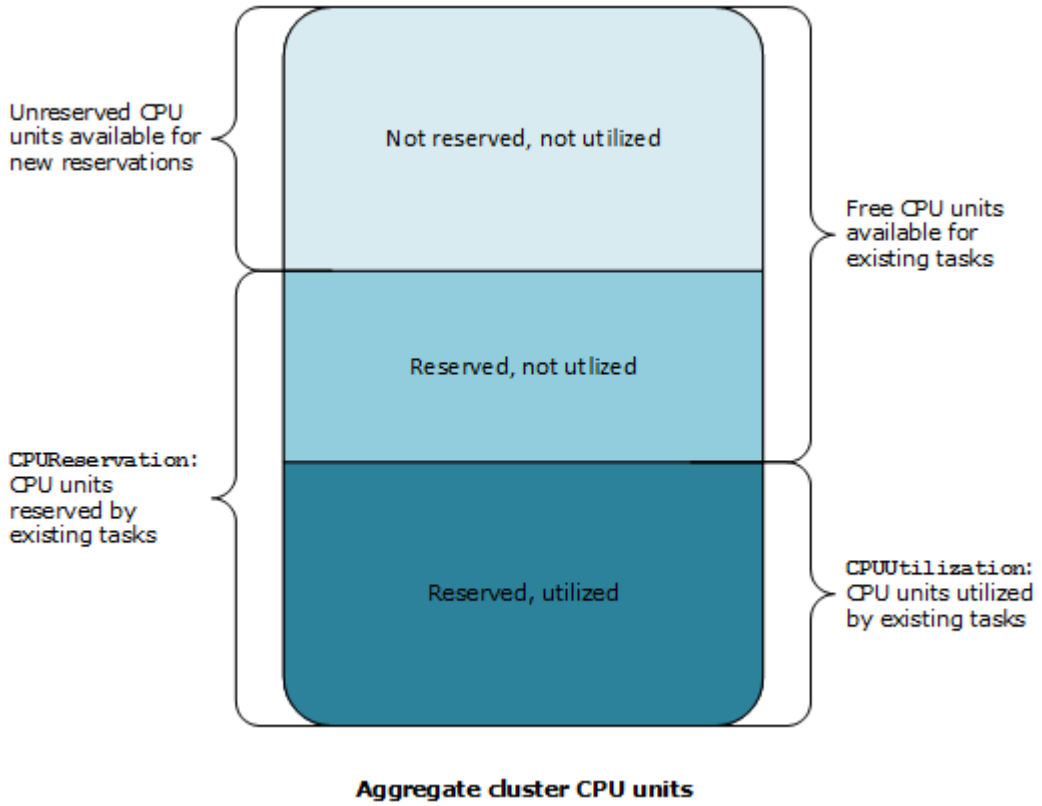
クラスターでタスクを実行すると、Amazon ECS はタスク定義を解析し、そのコンテナの定義で指定されている CPU ユニット、メモリ (MiB)、GPUs の合計を予約します。Amazon ECS は毎分、クラスター内で実行中の各タスクによって現在予約されている CPU ユニットの数、メモリ (MiB)、GPU の数を計算します。クラスター内で実行中のすべてのタスクによって予約されている CPU、メモリ、GPUs の合計が計算され、その数字がクラスターに登録されているリソースの合計に対する割合として CloudWatch にレポートされます。タスク定義でソフト制限 (`memoryReservation`) を指定する場合、予約メモリの容量を計算するためにその制限が使用されます。それ以外の場合は、ハード制限 (`memory`) が使用されます。クラスター内のタスクによって予約されるメモリの合計 MiB には、一時ファイルシステム (`tmpfs`) のボリュームサイズが含まれま

す。タスク定義で定義されている場合、sharedMemorySize も含まれます。ハード制限とソフト制限、共有メモリサイズ、tmpfs ボリュームサイズの詳細については、「タスク定義パラメータ」を参照してください。

例えば、クラスターに 2 つのアクティブなコンテナインスタンス、c4.4xlarge インスタンスおよび c4.large インスタンスが登録されているとします。c4.4xlarge インスタンスは、CPU ユニット 16,384、メモリ 30,158 MiB でクラスターに登録されています。c4.large インスタンスは、CPU ユニット 2,048、メモリ 3,768 MiB で登録されています。このクラスターの合計リソースは、CPU ユニット 18,432、メモリ 33,926 MiB です。

このタスク定義により、1,024 CPU ユニットと 2,048 MiB のメモリが予約され、このクラスター内で 10 個のタスクが開始されている (その他のタスクが現在実行されていない) 場合、合計 10,240 CPU ユニットと 20,480 MiB のメモリが予約されています。この場合、クラスターについて CPU 予約率 55%、メモリ予約率 60% として CloudWatch にレポートされます。

以下の図では、クラスターに登録されている CPU ユニットの合計と、それらの予約率および使用率が既存のタスクおよび新規タスク配置にどのように影響するかを示しています。下段 (予約済み、使用中) および中段 (予約済み、未使用) ブロックは、クラスターで実行中の既存タスクで予約されている CPU ユニットの合計、または CPUReservation CloudWatch メトリクスを表しています。下段ブロックは、実行中のタスクがクラスターで実際に使用している予約 CPU ユニット、または CPUUtilization CloudWatch メトリクスを表しています。上段ブロックは既存のタスクによって予約されていない CPU ユニットを表します。これらの CPU ユニットを新しいタスクの配置に利用できます。既存タスクの CPU リソースを増やす必要がある場合に、これらの予約されていない CPU ユニットを使用することもできます。詳細については、「cpu」タスク定義パラメータのドキュメントを参照してください。



Amazon ECS クラスターの使用率メトリクス

クラスター使用率メトリクスは CPU、メモリのほか、タスクに EBS ボリュームがアタッチされている場合は EBS ファイルシステムの使用率にも使用できます。このメトリクスは、Amazon EC2 インスタンスでホストされるタスクまたはサービスを含むクラスターでのみサポートされています。AWS Fargate でタスクがホストされているクラスターではサポートされていません。

Amazon ECS クラスターレベルの CPU とメモリの使用率メトリクス

CPU とクラスターの使用率は、クラスターに登録されたアクティブな Amazon EC2 インスタンスのそれぞれに登録された CPU とメモリの合計に対する、クラスター上のすべてのタスクによって使用されている CPU とメモリの割合で測定されます。`ACTIVE` または `DRAINING` ステータスの Amazon EC2 インスタンスのみが、クラスター使用率メトリクスに影響します。

```
                                (Total CPU units used by tasks in cluster) x 100
Cluster CPU utilization =
 ------------------------------------------------------------------
                           (Total CPU units registered by container instances in
 cluster)
```

```
                                    (Total MiB of memory used by tasks in cluster x
 100)
Cluster memory utilization =
 ------------------------------------------------------------------
                              (Total MiB of memory registered by container instances in
 cluster)
```

各 Amazon EC2 コンテナインスタンスの Amazon ECS エージェントは毎分、Amazon EC2 インスタンスで実行中の各タスクで現在使用されている CPU ユニット数とメモリの MiB を計算し、この情報は Amazon ECS にレポートされます。クラスターで実行中のすべてのタスクで使用されている CPU とメモリの合計が計算され、その数字がクラスターの予約リソースの合計に対する比率として CloudWatch にレポートされます。

例えば、クラスターに 2 つのアクティブな Amazon EC2 インスタンス、`c4.4xlarge` インスタンスと `c4.large` インスタンスが登録されているとします。`c4.4xlarge` インスタンスは、CPU ユニット数 `16,384`、メモリ `30,158` MiB でクラスターに登録されています。`c4.large` インスタンスは、CPU ユニット数 `2,048`、メモリ `3,768` MiB で登録されています。このクラスターの合計リソースは、CPU ユニット数 `18,432`、メモリ `33,926` MiB です。

このクラスターで 10 個のタスクが実行中で、各タスクが `1,024` CPU ユニットとメモリ `2,048` MiB を消費する場合、クラスターでは合計で `10,240` CPU ユニットとメモリ `20,480` MiB が使用されます。この場合、クラスターについて CPU 使用率 55%、メモリ使用率 60% として CloudWatch にレポートされます。

Amazon ECS クラスターレベルの Amazon EBS ファイルシステムの使用率

クラスターレベルの EBS ファイルシステム使用率メトリクスは、クラスターで実行されているタスクによって使用されている EBS ファイルシステムの合計を、クラスター内のすべてのタスクに割り当てられた EBS ファイルシステムストレージの合計で割った値で測定されます。

```
                                       (Total GB of EBS filesystem used by tasks in
 cluster x 100)
Cluster EBS filesystem utilization =
 ------------------------------------------------------------------
                                       (Total GB of EBS filesystem allocated to tasks
 in cluster)
```

Amazon ECS サービスの使用率メトリクス

サービス使用率メトリクスは CPU、メモリのほか、タスクに EBS ボリュームがアタッチされている場合は EBS ファイルシステムの使用率にも使用できます。サービスレベルのメトリクスは、Amazon EC2 インスタンスと Fargate の両方でタスクをホストするサービスでサポートされています。

サービスレベルの CPU とメモリの使用率

CPU とメモリの使用率は、サービスのタスク定義で指定された CPU とメモリに対する、クラスターのサービスに属する Amazon ECS タスクによって使用されている CPU とメモリの割合で測定されます。

CloudWatch でこれらのメトリクスを表示する場合、次のさまざまな統計を選択できます。

- Average: サービス内のすべてのタスクの平均使用率です。値は、次の式を使用して計算されます。
- Minimum: サービス内のリソース使用量が最も少ないタスクの使用率です。これは、タスク定義で指定されたものと比較して、最もリソースを消費しないタスクで使用される CPU またはメモリのパーセンテージを表します。
- Maximum: サービス内のリソース使用量が最も多いタスクの使用率です。これは、タスク定義で指定されたものと比較して、最もリソースを消費するタスクで使用される CPU またはメモリのパーセンテージを表します。

次の式は、統計 Average の計算方法を示しています。

```
                                  (Total CPU units used by tasks in service) x 100
Service CPU utilization =
 ------------------------------------------------------------------
                           (Total CPU units specified in the task definition) x (Number
 of tasks in the service)
```

```
                                     (Total MiB of memory used by tasks in service x
 100)
Service memory utilization =
 ---------------------------------------------------------------------
                              (Total MiB of memory specified in the task definition)  x
 (Number of tasks in the service)
```

> **Note**
>
> これらの式は、平均統計にのみ適用されます。統計 Minimum と Maximum は、すべてのタスクの集計計算ではなく、リソース使用率が最も低い個々のタスクと最も高い個々のタスクを表します。

Amazon ECS は 20 秒ごとにメトリクスを収集します。Amazon ECS コンテナエージェントは毎分、サービスが所有する実行タスクのそれぞれで現在使用されている CPU ユニット数とメモリの MiB を計算します。この情報は Amazon ECS に報告されます。クラスターで実行中のサービスが所有するすべてのタスクで使用されている CPU とメモリの合計が計算され、その数字がサービスのタスク定義でサービス用に指定されたリソースの合計に対する比率として CloudWatch にレポートされます。最小値と最大値は、20 秒のメトリクスの最小値と最大値です。平均は 3 つの値の集計です。

ソフト制限 (`memoryReservation`) を指定した場合、予約メモリの容量を計算するためにその制限が使用されます。それ以外の場合は、ハード制限 (`memory`) が使用されます。ハードリミットとソフトリミットの詳細については、「タスクサイズ」を参照してください。

例えば、サービスのタスク定義で 512 CPU ユニットおよび 1,024 MiB のメモリを (ハードリミットの `memory` パラメータとともに) コンテナ全体で指定しているとします。サービスの実行タスクの必要数は 1 であり、1 つの `c4.large` コンテナインスタンス (2,048 CPU ユニット、3,768 MiB の合計メモリ) を使用してクラスターで実行されます。また、クラスターで実行されている他のタスクはありません。タスクは 512 CPU ユニットを指定していますが、2,048 CPU ユニットを搭載したコンテナインスタンスで唯一実行されているタスクであるため、指定した量の 4 倍 (2,048/512) まで使用できます。ただし、指定した 1,024 MiB のメモリはハード制限であり、それを超えることはできません。したがってこの場合、サービスのメモリ使用率は 100% を超えることはできません。

前の例で、ハードリミットの `memoryReservation` パラメータの代わりにソフト制限の `memory` を使用した場合、サービスのタスクでは、必要に応じて指定された 1,024 MiB を超えるメモリを使用できます。この場合、サービスのメモリ使用率が 100% を超える可能性があります。

アプリケーションのメモリ使用率が短時間で突然急上昇した場合、Amazon ECS は毎分複数のデータポイントを収集し、CloudWatch に送信される 1 つのデータポイントに集約するため、サービスメモリ使用率の増加を見ることがありません。

このタスクが一定期間中に CPU 負荷の高い作業を実行し、使用可能な 2,048 CPU ユニットすべてと 512 MiB のメモリを使用している場合、サービスからは CPU 使用率 400%、メモリ使用率 50% としてレポートされます。タスクがアイドル状態で 128 CPU ユニットおよび 128 MiB のメモリを

使用している場合、サービスからは CPU 使用率 25%、メモリ使用率 12.5% としてレポートされます。

> **Note**
>
> この例では、CPU 単位がコンテナレベルで定義されている場合に、CPU 使用率が 100% を超えるだけです。タスク・レベルで CPU ユニットを定義した場合、使用率は定義されたタスク・レベルの制限を超えません。

サービスレベルの EBS ファイルシステム使用率

サービスレベルの EBS ファイルシステムの使用率は、サービスに属するタスクによって使用されている EBS ファイルシステムの合計を、サービスに属するすべてのタスクに割り当てられている EBS ファイルシステムストレージの合計で割った値で測定されます。

サービスの **RUNNING** タスク数

CloudWatch メトリクスを使用して、RUNNING 状態のサービス内のタスク数を表示できます。例えば、このメトリクスに CloudWatch アラームを設定して、サービスで実行中のタスクの数が指定された値を下回った場合にアラートを送信できます。

Amazon ECS CloudWatch Container Insights のサービス **RUNNING** タスク数

Amazon ECS CloudWatch Container Insights を使用すると、[Number of Running Tasks] (実行中のタスクの数) (`RunningTaskCount`) メトリクスがクラスターごとおよびサービスごとに利用できます。Container Insights は、`containerInsights` アカウント設定にオプトインして作成された新しいすべてのクラスターで、またはクラスター作成時にクラスター設定を有効にすることで個々のクラスターで、または UpdateClusterSettings API を使用して既存のクラスターで、使用できます。CloudWatch Container インサイトによって収集されたメトリクスは、カスタムメトリクスとして課金されます。CloudWatch の料金の詳細については、 CloudWatch の料金をご覧ください。

このメトリクスを表示するには、「Amazon CloudWatch ユーザーガイド」の「Amazon ECS Container Insights のメトリクス」を参照してください。

高解像度のサービスメトリクス

デフォルトでは、Amazon ECS は `CPUUtilization` および `MemoryUtilization` サービスメトリクスを 60 秒の解像度で CloudWatch に発行します。サービスでモニタリング設定を行うことで、20

秒単位の解像度を設定できます。これにより、リソース使用率の変化をより迅速に検出でき、サービスの自動スケーリングを高速化するために活用できます。

高解像度メトリクスを有効にするには、サービスを作成または更新するときに `20` を `resolutionSeconds` に設定してモニタリング設定を指定します。高解像度スケーリングポリシーの設定方法を含む完全なセットアップ手順については、「高解像度メトリクスによる高速自動スケーリング」を参照してください。

Amazon ECS サービス使用率メトリクスのユースケース

次のリストは、Amazon ECS メトリクスを使用するタイミングに関する情報を示しています。

- リソース使用率のモニタリング: 平均統計を使用して、CPU とメモリの消費パターンをモニタリングし、パフォーマンスベースラインを確立して、パフォーマンスが段階的に低下する傾向を検出します。
- コスト最適化: 平均統計を使用して、リソースの使用状況のモニタリングし、コンテナを適切なサイズに設定し、使用状況パターンに基づく予約を調整して、スケジュールされたスケーリングを実装します。
- パフォーマンスベンチマーク: 平均統計を使用して、サービスリビジョン間のメトリクスを比較し、パフォーマンス KPI を確立し、最適化の改善を検証します。
- リソースフロア検出: 平均統計を使用して、休止期間中の最小リソースの最小ニーズを特定し、適切な予約を設定し、異常なドロップを検出します。
- 異常検出: 最小統計を使用して、初期化の失敗や予期しない休止期間などの潜在的な問題を示す異常なリソース使用率の低下を特定します。
- スケーリングポリシーの改善: 最小統計を使用して、実行可能な最小使用率に基づいて最適なスケールインしきい値を確立し、積極的なスケーリングを防ぎます。
- キャパシティプランニング: 最大統計を使用して、適切なタスクサイズを設定し、トラフィックの急増に備えて十分な余裕のあるインフラストラクチャを計画します。
- パフォーマンスのボトルネックの特定: 最大統計を使用して、リソースの飽和ポイントを検出し、ボトルネックを特定し、タスクサイズを拡大するタイミングを判断します。
- スケーリングポリシー設定: 最大統計を使用して、ピークパターンに基づいて最適なスケールアウトしきい値を設定し、バーストキャパシティを設定します。
- SLA コンプライアンスモニタリング: 最大統計を使用して、ピーク時の応答時間とエラー率を追跡し、サービスパフォーマンスが定義された SLA を満たしていることを確認します。

関連するメトリクス情報

Container Insights の詳細については、「CloudWatch ユーザーガイド」の「Container Insights use cases」を参照してください。

EventBridge を使用して Amazon ECS エラーへの対応を自動化する

Amazon EventBridge を使用すると、AWS のサービスを自動化して、アプリケーションの可用性の問題やリソース変更といったシステムイベントに自動的に対応できます。AWS サービスからのイベントは、ほぼリアルタイムに EventBridge に提供されます。どのイベントに興味があるのか、イベントがルールに一致した場合にどのように自動的に実行するアクションをとるのか簡単なルールを指定して書き込みすることができます。自動的に設定できるオペレーションには、以下が含まれます。

- CloudWatch Logs のロググループへのイベントの追加
- AWS Lambda 関数の呼び出し
- Amazon EC2 Run Command の呼び出し
- Amazon Kinesis Data Streams へのイベントの中継
- AWS Step Functions ステートマシンのアクティブ化
- Amazon SNS トピックまたは Amazon Simple Queue Service (Amazon SQS) キューに通知する

詳細については、「Amazon EventBridge ユーザーガイド」の「Getting started with Amazon EventBridge」を参照してください。

Amazon ECS の Eventbridge イベントを使用して、Amazon ECS クラスターの現在の状態に関するほぼリアルタイムの通知を受け取ることができます。タスクで EC2 を使用している場合は、コンテナインスタンスと、それらのコンテナインスタンスで実行中のすべてのタスクの現在の状態の両方の状態を確認できます。タスクで Fargate を使用している場合、コンテナインスタンスの状態を確認できます。

Eventbridge を使用して Amazon ECS 上にカスタムスケジューラを構築し、クラスター間のタスクを調整するとともに、クラスターの状態をほぼリアルタイムにモニタリングできます。Amazon ECS サービス に対して Amazon ECS 状態の変更を絶え間なくポーリングするスケジューリングおよびモニタリングのコードをなくし、代わりに、Eventbridge ターゲットを使用して非同期的に状態の変更に対応できます。ターゲットには、AWS Lambda、Amazon Simple Queue Service、Amazon Simple Notification Service、Amazon Kinesis Data Streams

Amazon ECS イベントストリームは、イベントごとに少なくとも 1 回必ず送信されます。重複したイベントが送信された場合、イベントには重複を識別できるだけの十分な情報が備わっています。詳細については、「Amazon ECS イベントの処理」を参照してください。

イベントは相対的な順番になっているため、特定のイベントがいつ発生したかは、他のイベントとの前後関係から簡単に判断できます。

トピック

- Amazon ECS イベント
- Amazon ECS イベントの処理

Amazon ECS イベント

Amazon ECS は、各タスクとサービスの状態を追跡します。タスクやサービスの状態が変わると、イベントが生成され、Amazon EventBridge に送信されます。これらのイベントは、タスク状態変更イベントおよびサービスアクションイベントとして分類されます。各イベントとその考えられる原因については、以下のセクションで詳しく説明します。

Amazon ECS は、次のタイプのイベントを生成して EventBridge に送信します。

- コンテナインスタンス状態変更
- タスク状態変更
- デプロイの状態変更
- フック状態変更
- サービスアクション
- コンテナインスタンスのヘルス変更
- デーモンデプロイ状態の変更
- デーモンサービスアクション

> **Note**
>
> 今後、Amazon ECS に他の種類のイベント、ソース、詳細が追加される場合があります。コードのイベント JSON データを逆シリアル化する場合は、不明なプロパティが追加されたときに問題が発生しないように、アプリケーションが不明なプロパティに対応できるようにしてください。

同じアクティビティに対して複数のイベントが生成される場合もあります。例えば、コンテナインスタンスでタスクが開始されると、この新しいタスクに対してタスク状態変更イベントが生成されます。コンテナインスタンスの使用可能なリソース (CPU、メモリ、使用可能なポートなど) の変更に対応するため、コンテナインスタンス状態変更イベントがアカウントに生成されます。同様に、コンテナインスタンスが終了すると、コンテナインスタンス、コンテナエージェント接続ステータス、およびコンテナインスタンスで実行されている各タスクに対してイベントが生成されます。

コンテナ状態変更イベントとタスク状態変更イベントには 2 つの `version` フィールドがあります。1 つはイベントの本体で、もう 1 つはイベントの `detail` オブジェクトです。次に、これら 2 つのフィールドの違いについて説明します。

- イベントの本文の `version` フィールドは、すべてのイベントで `0` に設定されています。EventBridge パラメータの詳細については、「Amazon EventBridge ユーザーガイド」の「AWS サービスイベントメタデータ」を参照してください。
- イベントの `version` オブジェクトの `detail` フィールドは、関連付けられているリソースのバージョンについて説明します。リソースの状態が変わるたびに、このバージョンはインクリメントされます。イベントは複数回送信できるため、このフィールドで重複するイベントを識別できます。重複するイベントには、`detail` オブジェクト内で同じバージョンがあります。EventBridge で Amazon ECS コンテナインスタンスおよびタスク状態をレプリケートする場合は、Amazon ECS API によって報告されるリソースのバージョンとそのリソースについて EventBridge で報告されるバージョン (`detail` オブジェクト内) を比較して、イベントストリームでのバージョンが最新であることを確認できます。

サービスアクションイベントには、本体の `version` フィールドのみが含まれます。

サービスアクションイベントは、次の 2 つの異なるフィールドでサービスを指定します。

- `create-service` が生成したイベントの場合、サービスは `serviceName` フィールドに設定されます。
- `update-service` が生成したイベントの場合、サービスは `service` フィールドに設定されます。

サービスイベントを自動化ツールで処理する場合、両方のフィールドをコーディングする必要があります。

サービスアクションイベントのルールを作成する方法については、「Amazon ECS サービスアクションイベント」を参照してください。

Amazon ECS と EventBridge を統合する方法に関する追加情報については、「Amazon EventBridge と Amazon ECS の統合」を参照してください。

Amazon ECS コンテナインスタンス状態変更イベント

以下の場合は、コンテナインスタンス状態変更イベントが発生します。

ユーザーが StartTask、RunTask、StopTask API オペレーションを直接または AWS マネジメントコンソール や SDKs 経由で呼び出します。

コンテナインスタンスでタスクを配置または停止すると、コンテナインスタンスの使用可能なリソース (CPU、メモリ、使用可能なポートなど) が変更されます。

Amazon ECS サービススケジューラがタスクを開始または停止する。

コンテナインスタンスでタスクを配置または停止すると、コンテナインスタンスの使用可能なリソース (CPU、メモリ、使用可能なポートなど) が変更されます。

SubmitTaskStateChangeAmazon ECSコンテナエージェントは、目的のRUNNINGステータスがタスクのSTOPPEDステータスでAPIオペレーションを呼び出します。

Amazon ECS コンテナエージェントは、コンテナインスタンスでタスクの状態をモニタリングし、状態に変更があるとレポートします。RUNNING であるべきタスクが STOPPED に変わると、エージェントは停止したタスクに割り当てられているリソース (CPU、メモリ、使用可能なポートなど) を解放します。

ユーザーが DeregisterContainerInstance API オペレーションを使用してコンテナインスタンスを直接または AWS マネジメントコンソール や SDKs 経由で登録解除します。

コンテナインスタンスを登録解除すると、コンテナインスタンスのステータスと Amazon ECS コンテナエージェントの接続ステータスが変更されます。

EC2 インスタンスの停止に伴ってタスクが停止されました。

コンテナインスタンスを停止すると、このコンテナインスタンスで実行されているタスクのステータスは STOPPED に変わります。

Amazon ECS コンテナエージェントがコンテナインスタンスを初めて登録します。

Amazon ECS コンテナエージェントで初めてコンテナインスタンスを (起動時または手動による初回起動時に) 登録するときに、インスタンスの状態変更イベントが作成されます。

Amazon ECS コンテナエージェントが、Amazon ECS と接続または接続解除する。

Amazon ECS コンテナエージェントが Amazon ECS バックエンドと接続または接続解除すると、コンテナインスタンスの agentConnected ステータスが変わります。

> **Note**
>
> Amazon ECS コンテナエージェントは、通常の操作の一環として、1 時間に数回切断して再接続するため、エージェントの接続イベントが予期されます。これらのイベントは、コンテナエージェントまたはコンテナインスタンスに問題があることを示すものではありません。

インスタンスの Amazon ECS コンテナエージェントをアップグレードします。

コンテナインスタンスの詳細には、コンテナエージェントバージョンのオブジェクトが含まれています。エージェントをアップグレードすると、このバージョン情報が変わり、イベントが発生します。

Exampleコンテナインスタンス状態変更イベント

コンテナインスタンスの状態変更イベントは、次の形式で配信されます。`detail` セクションは、「Amazon Elastic Container Service API リファレンス」の DescribeContainerInstances API オペレーションから返される ContainerInstance オブジェクトに似ています。EventBridge パラメータの詳細については、「Amazon EventBridge ユーザーガイド」の「AWS サービスイベントメタデータ」を参照してください。

```
{
  "version": "0",
  "id": "8952ba83-7be2-4ab5-9c32-6687532d15a2",
  "detail-type": "ECS Container Instance State Change",
  "source": "aws.ecs",
  "account": "111122223333",
  "time": "2016-12-06T16:41:06Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:ecs:us-east-1:111122223333:container-instance/
b54a2a04-046f-4331-9d74-3f6d7f6ca315"
  ],
  "detail": {
    "agentConnected": true,
    "attributes": [
      {
        "name": "com.amazonaws.ecs.capability.logging-driver.syslog"
      },
```

```
      {
        "name": "com.amazonaws.ecs.capability.task-iam-role-network-host"
      },
      {
        "name": "com.amazonaws.ecs.capability.logging-driver.awslogs"
      },
      {
        "name": "com.amazonaws.ecs.capability.logging-driver.json-file"
      },
      {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.17"
      },
      {
        "name": "com.amazonaws.ecs.capability.privileged-container"
      },
      {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.18"
      },
      {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.19"
      },
      {
        "name": "com.amazonaws.ecs.capability.ecr-auth"
      },
      {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.20"
      },
      {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.21"
      },
      {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.22"
      },
      {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.23"
      },
      {
        "name": "com.amazonaws.ecs.capability.task-iam-role"
      }
    ],
    "clusterArn": "arn:aws:ecs:us-east-1:111122223333:cluster/default",
    "containerInstanceArn": "arn:aws:ecs:us-east-1:111122223333:container-instance/
b54a2a04-046f-4331-9d74-3f6d7f6ca315",
    "ec2InstanceId": "i-f3a8506b",
```

```
    "registeredResources": [
      {
        "name": "CPU",
        "type": "INTEGER",
        "integerValue": 2048
      },
      {
        "name": "MEMORY",
        "type": "INTEGER",
        "integerValue": 3767
      },
      {
        "name": "PORTS",
        "type": "STRINGSET",
        "stringSetValue": [
          "22",
          "2376",
          "2375",
          "51678",
          "51679"
        ]
      },
      {
        "name": "PORTS_UDP",
        "type": "STRINGSET",
        "stringSetValue": []
      }
    ],
    "remainingResources": [
      {
        "name": "CPU",
        "type": "INTEGER",
        "integerValue": 1988
      },
      {
        "name": "MEMORY",
        "type": "INTEGER",
        "integerValue": 767
      },
      {
        "name": "PORTS",
        "type": "STRINGSET",
        "stringSetValue": [
          "22",
```

```
          "2376",
          "2375",
          "51678",
          "51679"
        ]
      },
      {
        "name": "PORTS_UDP",
        "type": "STRINGSET",
        "stringSetValue": []
      }
    ],
    "status": "ACTIVE",
    "version": 14801,
    "versionInfo": {
      "agentHash": "aebcbca",
      "agentVersion": "1.13.0",
      "dockerVersion": "DockerVersion: 1.11.2"
    },
    "updatedAt": "2016-12-06T16:41:06.991Z"
  }
}
```

Amazon ECS タスク状態変更イベント

以下の場合は、タスク状態変更イベントが発生します。

ユーザーが StartTask、RunTask、StopTask API オペレーションを直接または AWS マネジメントコンソール、AWS CLI、SDKs 経由で呼び出します。

タスクを開始または停止すると、新しいタスクリソースが作成されるか、既存のタスクリソースの状態が変更されます。

Amazon ECS サービススケジューラがタスクを開始または停止する。

タスクを開始または停止すると、新しいタスクリソースが作成されるか、既存のタスクリソースの状態が変更されます。

Amazon ECS コンテナエージェントが SubmitTaskStateChange API オペレーションを呼び出す。

EC2 では、Amazon ECS コンテナエージェントは、コンテナインスタンス上のタスクの状態をモニタリングします。Amazon ECS コンテナエージェントは、状態に変更があるとレポートしま

す。状態の変更には、PENDING から RUNNING または RUNNING から STOPPED への変更が含まれます。

ユーザーが、DeregisterContainerInstance API オペレーションと force フラグを直接または AWS マネジメントコンソール や SDKs. 経由で使用して、基盤となるコンテナインスタンスの登録解除を強制します。

コンテナインスタンスを登録解除すると、コンテナインスタンスのステータスと Amazon ECS コンテナエージェントの接続ステータスが変更されます。コンテナインスタンスでタスクが実行されている場合、登録解除を許可するには force フラグを設定する必要があります。これにより、インスタンスのすべてのタスクが停止します。

基盤となるコンテナインスタンスが停止または終了する。

コンテナインスタンスを停止または終了すると、このコンテナインスタンスで実行されているタスクのステータスは STOPPED に変わります。

タスクのコンテナの状態が変わる。

Amazon ECS コンテナエージェントは、タスク内のコンテナの状態をモニタリングします。例えば、タスク内で実行されているコンテナが停止すると、このコンテナの状態変更に伴ってイベントが生成されます。

Fargate Spot キャパシティープロバイダーを使用するタスクは、終了通知を受け取ります。

タスクが FARGATE_SPOT キャパシティープロバイダーを使用していて、スポットの中断により停止すると、タスク状態変更イベントが生成されます。

Exampleタスク状態変更イベント

タスク状態変更イベントは、次の形式で配信されます。フィールドについて、以下の点に注意してください。

- イベントのヘルスステータス情報は、タスク状態変更イベントでは提供されません。タスクのヘルスステータスが必要な場合は、describe-tasks を実行します。
- コンテナが Amazon ECR でホストされているイメージを使用している場合は、imageDigest フィールドが返されます。
- createdAt、connectivityAt、pullStartedAt、startedAt、pullStoppedAt、updatedAt のフィールドの値は ISO 形式のタイムスタンプです。
- detail-type の値は「ECS タスク状態変更」です。

- 停止したタスクに対してイベントが生成されると、タスクが停止した理由 (「ユーザー開始」など) に関する追加情報がフィールド `stoppedReason`と `stopCode` に表示されます。

EventBridge パラメータの詳細については、「Amazon EventBridge イベントリファレンス」の「AWS サービスイベントメタデータ」を参照してください。

必須コンテナのいずれかが終了したためにタスクの実行が停止した場合に限りタスクイベントをキャプチャするように Amazon EventBridge イベントルールを設定する方法の詳細については、「Amazon ECS タスク停止イベントに関する Amazon Simple Notification Service アラートの送信」を参照してください。


```
{
    "version": "0",
    "id": "105f6bb1-4da6-c630-4965-35383018cbca",
    "detail-type": "ECS Task State Change",
    "source": "aws.ecs",
    "account": "123456789012",
    "time": "2025-05-06T11:02:34Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:ecs:us-east-1:123456789012:task/example-cluster/a1173316d40a45dea9"
    ],
    "detail": {
        "attachments": [
            {
                "id": "fe3a9a46-6a47-40ee-afd9-7952ae90a75a",
                "type": "eni",
                "status": "ATTACHED",
                "details": [
                    {
                        "name": "subnetId",
                        "value": "subnet-0d0eab1bb38d5ca64"
                    },
                    {
                        "name": "networkInterfaceId",
                        "value": "eni-0103a2f01bad57d71"
                    },
                    {
                        "name": "macAddress",
                        "value": "0e:50:d1:c1:77:81"
                    },
                    {
```

```
                        "name": "privateDnsName",
                        "value": "ip-10-0-1-163.ec2.internal"
                    },
                    {
                        "name": "privateIPv4Address",
                        "value": "10.0.1.163"
                    }
                ]
            }
        ],
        "attributes": [
            {
                "name": "ecs.cpu-architecture",
                "value": "x86_64"
            }
        ],
        "availabilityZone": "us-east-1b",
        "capacityProviderName": "FARGATE",
        "clusterArn": "arn:aws:ecs:us-east-1:123456789012:cluster/example-cluster",
        "connectivity": "CONNECTED",
        "connectivityAt": "2025-05-06T11:02:17.19Z",
        "containers": [
            {
                "containerArn": "arn:aws:ecs:us-east-1:123456789012:container/example-
cluster/a1173316d40a45dea9/a0a99b87-baa8-4bf6-b9f1-a9a95917a635",
                "lastStatus": "RUNNING",
                "name": "web",
                "image": "nginx",
                "imageDigest":
 "sha256:c15da6c91de8d2f436196f3a768483ad32c258ed4e1beb3d367a27ed67253e66",
                "runtimeId": "a1173316d40a45dea9-0265927825",
                "taskArn": "arn:aws:ecs:us-east-1:123456789012:task/example-cluster/
a1173316d40a45dea9",
                "networkInterfaces": [
                    {
                        "attachmentId": "fe3a9a46-6a47-40ee-afd9-7952ae90a75a",
                        "privateIpv4Address": "10.0.1.163"
                    }
                ],
                "cpu": "99",
                "memory": "100"
            },
            {
```

```
                "containerArn": "arn:aws:ecs:us-east-1:123456789012:container/example-
cluster/a1173316d40a45dea9/a2010e2d-ba7c-4135-8b79-e0290ff3cd8c",
                "lastStatus": "RUNNING",
                "name": "aws-guardduty-agent-nm40lC",
                "imageDigest":
 "sha256:bf9197abdf853607e5fa392b4f97ccdd6ca56dd179be3ce8849e552d96582ac8",
                "runtimeId": "a1173316d40a45dea9-2098416933",
                "taskArn": "arn:aws:ecs:us-east-1:123456789012:task/example-cluster/
a1173316d40a45dea9",
                "networkInterfaces": [
                    {
                        "attachmentId": "fe3a9a46-6a47-40ee-afd9-7952ae90a75a",
                        "privateIpv4Address": "10.0.1.163"
                    }
                ],
                "cpu": "null"
            },
            {
                "containerArn": "arn:aws:ecs:us-east-1:123456789012:container/example-
cluster/a1173316d40a45dea9/dccf0ca2-d929-471f-a5c3-98006fd4379e",
                "lastStatus": "RUNNING",
                "name": "aws-otel-collector",
                "image": "public.ecr.aws/aws-observability/aws-otel-collector:v0.32.0",
                "imageDigest":
 "sha256:7a1b3560655071bcacd66902c20ebe9a69470d5691fe3bd36baace7c2f3c4640",
                "runtimeId": "a1173316d40a45dea9-4027662657",
                "taskArn": "arn:aws:ecs:us-east-1:123456789012:task/example-cluster/
a1173316d40a45dea9",
                "networkInterfaces": [
                    {
                        "attachmentId": "fe3a9a46-6a47-40ee-afd9-7952ae90a75a",
                        "privateIpv4Address": "10.0.1.163"
                    }
                ],
                "cpu": "0"
            }
        ],
        "cpu": "256",
        "createdAt": "2025-05-06T11:02:13.877Z",
        "desiredStatus": "RUNNING",
        "enableExecuteCommand": false,
        "ephemeralStorage": {
            "sizeInGiB": 20
        },
```

```
        "group": "family:webserver",
        "launchType": "FARGATE",
        "lastStatus": "RUNNING",
        "memory": "512",
        "overrides": {
            "containerOverrides": [
                {
                    "name": "web"
                },
                {
                    "environment": [
                        {
                            "name": "CLUSTER_NAME",
                            "value": "example-cluster"
                        },
                        {
                            "name": "REGION",
                            "value": "us-east-1"
                        },
                        {
                            "name": "HOST_PROC",
                            "value": "/host_proc"
                        },
                        {
                            "name": "AGENT_RUNTIME_ENVIRONMENT",
                            "value": "ecsfargate"
                        },
                        {
                            "name": "STAGE",
                            "value": "prod"
                        }
                    ],
                    "memory": 128,
                    "name": "aws-guardduty-agent-nm40lC"
                },
                {
                    "name": "aws-otel-collector"
                }
            ]
        },
        "platformVersion": "1.4.0",
        "pullStartedAt": "2025-05-06T11:02:24.162Z",
        "pullStoppedAt": "2025-05-06T11:02:33.493Z",
        "startedAt": "2025-05-06T11:02:34.325Z",
```

```
        "taskArn": "arn:aws:ecs:us-east-1:123456789012:task/example-cluster/
a1173316d40a45dea9",
        "taskDefinitionArn": "arn:aws:ecs:us-east-1:123456789012:task-definition/
webserver:5",
        "updatedAt": "2025-05-06T11:02:34.325Z",
        "version": 3
    }
}
```

Example

EC2 のタスク状態変更イベントの例を次に示します。

```
{
    "version": "0",
    "id": "a65cf262-f104-0dd5-ceda-4b09ba71a441",
    "detail-type": "ECS Task State Change",
    "source": "aws.ecs",
    "account": "123456789012",
    "time": "2025-05-12T13:12:06Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:ecs:us-east-1:123456789012:task/example/c1ffa94f19a540ed8d9f7e1d2a5d"
    ],
    "detail": {
        "attachments": [
            {
                "id": "52333e3b-b812-41a8-b057-9ed184bbe5e1",
                "type": "eni",
                "status": "ATTACHED",
                "details": [
                    {
                        "name": "subnetId",
                        "value": "subnet-0d0eab1bb38d5ca64"
                    },
                    {
                        "name": "networkInterfaceId",
                        "value": "eni-0ea90f746500773a4"
                    },
                    {
                        "name": "macAddress",
                        "value": "0e:d5:9b:ce:49:fb"
                    },
                    {
```

```
                        "name": "privateDnsName",
                        "value": "ip-10-0-1-37.ec2.internal"
                    },
                    {
                        "name": "privateIPv4Address",
                        "value": "10.0.1.37"
                    }
                ]
            }
        ],
        "attributes": [
            {
                "name": "ecs.cpu-architecture",
                "value": "x86_64"
            }
        ],
        "availabilityZone": "us-east-1b",
        "capacityProviderName": "Infra-ECS-Cluster-example-fa84e0cc-
AsgCapacityProvider-OseQJU9pizmp",
        "clusterArn": "arn:aws:ecs:us-east-1:123456789012:cluster/example",
        "connectivity": "CONNECTED",
        "connectivityAt": "2025-05-12T13:11:44.98Z",
        "containerInstanceArn": "arn:aws:ecs:us-east-1:123456789012:container-instance/
example/d1d84798400f49f3b21cb61610c1e",
        "containers": [
            {
                "containerArn": "arn:aws:ecs:us-east-1:123456789012:container/example/
c1ffa94f19a540ed8d9f7e1d2a5d3626/197d0994-5367-4a6d-9f9a-f075e4a6",
                "lastStatus": "RUNNING",
                "name": "aws-otel-collector",
                "image": "public.ecr.aws/aws-observability/aws-otel-collector:v0.32.0",
                "imageDigest":
 "sha256:7a1b3560655071bcacd66902c20ebe9a69470d5691fe3bd36baace7c2f3c4640",
                "runtimeId":
 "8e926f0ccd8fe2b459926f49584ba6d33a3d9f61398dbabe944ee6a13a8ff3a1",
                "taskArn": "arn:aws:ecs:us-east-1:123456789012:task/example/
c1ffa94f19a540ed8d9f7e1d2a5d",
                "networkInterfaces": [
                    {
                        "attachmentId": "52333e3b-b812-41a8-b057-9ed184bbe5e1",
                        "privateIpv4Address": "10.0.1.37"
                    }
                ],
                "cpu": "0"
```

```
            },
            {
                "containerArn": "arn:aws:ecs:us-east-1:123456789012:container/example/
c1ffa94f19a540ed8d9f7e1d2a5d3626/cab39ef0-9c50-459d-844b-b9d51d73d",
                "lastStatus": "RUNNING",
                "name": "web",
                "image": "nginx",
                "imageDigest":
 "sha256:c15da6c91de8d2f436196f3a768483ad32c258ed4e1beb3d367a27ed67253e66",
                "runtimeId":
 "9f1c73f0094f051541d9e5c2ab1e172d83c4eb5171bcc857c4504b02770ff3b8",
                "taskArn": "arn:aws:ecs:us-east-1:123456789012:task/example/
c1ffa94f19a540ed8d9f7e1d2a5d",
                "networkInterfaces": [
                    {
                        "attachmentId": "52333e3b-b812-41a8-b057-9ed184bbe5e1",
                        "privateIpv4Address": "10.0.1.37"
                    }
                ],
                "cpu": "99",
                "memory": "100"
            }
        ],
        "cpu": "256",
        "createdAt": "2025-05-12T13:11:44.98Z",
        "desiredStatus": "RUNNING",
        "enableExecuteCommand": false,
        "group": "family:webserver",
        "launchType": "EC2",
        "lastStatus": "RUNNING",
        "memory": "512",
        "overrides": {
            "containerOverrides": [
                {
                    "name": "aws-otel-collector"
                },
                {
                    "name": "web"
                }
            ]
        },
        "pullStartedAt": "2025-05-12T13:11:59.491Z",
        "pullStoppedAt": "2025-05-12T13:12:05.896Z",
        "startedAt": "2025-05-12T13:12:06.053Z",
```

```
        "taskArn": "arn:aws:ecs:us-east-1:123456789012:task/example/
c1ffa94f19a540ed8d9f7e1d2a5d",
        "taskDefinitionArn": "arn:aws:ecs:us-east-1:123456789012:task-definition/
webserver",
        "updatedAt": "2025-05-12T13:12:06.053Z",
        "version": 4
    }
}
```

Amazon ECS サービスアクションイベント

Amazon ECS は、詳細タイプ ECS サービスアクションのサービスアクションイベントを送信します。コンテナインスタンスおよびタスク状態変更イベントとは異なり、サービスアクションイベントの `details` レスポンスフィールドにバージョン番号は含まれません。次に、Amazon ECS サービス アクションイベントの EventBridge ルールを作成するために使用されるイベントパターンを示します。詳細については、「Amazon EventBridge ユーザーガイド」の「Getting started with EventBridge」を参照してください。

```
{
    "source": [
        "aws.ecs"
    ],
    "detail-type": [
        "ECS Service Action"
    ]
}
```

Amazon ECS は、`INFO`、`WARN`、`ERROR` イベントタイプのイベントを送信します。以下はサービスアクションイベントです。

`INFO` イベントタイプのサービスアクションイベント

`SERVICE_STEADY_STATE`

サービスは正常であり、目的のタスクの数であるため、安定した状態に達します。サービススケジューラは定期的にステータスを報告するため、このメッセージを複数回受信する場合があります。

`TASKSET_STEADY_STATE`

タスクセットは正常で、目的のタスクの数であるため、定常状態に達します。

CAPACITY_PROVIDER_STEADY_STATE

サービスに関連付けられたキャパシティープロバイダーが定常状態に達します。

SERVICE_DESIRED_COUNT_UPDATED

サービススケジューラが、サービスまたはタスクセットに対して計算された目的のカウントを更新するとき。このイベントは、目的のカウントがユーザーによって手動で更新されるときには送信されません。

TASKS_STOPPED

サービスが実行中のタスクを停止しました。

SERVICE_DEPLOYMENT_IN_PROGRESS

サービスデプロイが進行中です。サービスデプロイは、ロールバックまたは新しいサービスリビジョンのいずれかです。

SERVICE_DEPLOYMENT_COMPLETED

サービスデプロイが定常状態であり、完了しています。サービスデプロイは、ロールバック、または更新されたサービスリビジョンのデプロイのいずれかです。

WARN イベントタイプのサービスアクションイベント

SERVICE_TASK_START_IMPAIRED

サービスは一貫してタスクを正常に起動することができません。

SERVICE_DISCOVERY_INSTANCE_UNHEALTHY

サービス検出を使用するサービスに、異常なタスクが含まれています。サービススケジューラは、サービスレジストリ内のタスクが正常でないことを検出します。

VPC_LATTICE_TARGET_UNHEALTHY

VPC Lattice を使用するサービスが、VPC Lattice のターゲットの 1 つに異常を検出しました。

ERROR イベントタイプのサービスアクションイベント

SERVICE_DAEMON_PLACEMENT_CONSTRAINT_VIOLATED

DAEMON サービススケジューラ戦略を使用するサービス内のタスクは、サービスの配置制約戦略を満たさなくなりました。

`ECS_OPERATION_THROTTLED`

Amazon ECS API スロットルの制限により、サービススケジューラが調整されました。

`SERVICE_DISCOVERY_OPERATION_THROTTLED`

AWS Cloud Map API スロットルの制限により、サービススケジューラが調整されました。これは、サービス検出を使用するように設定されたサービスで発生する可能性があります。

`SERVICE_TASK_PLACEMENT_FAILURE`

サービススケジューラがタスクを配置できません。原因は、`reason` フィールドに説明されます。

このサービスイベントが生成される一般的な原因は、タスクを配置するためのクラスターでリソースが不足しているためです。例えば、使用可能なコンテナインスタンスに CPU またはメモリ容量が不足しているか、使用可能なコンテナインスタンスがない場合などです。もう 1 つの一般的な原因は、Amazon ECS コンテナエージェントがコンテナインスタンスで切断され、スケジューラがタスクを配置できない場合です。

`SERVICE_TASK_CONFIGURATION_FAILURE`

設定エラーのため、サービススケジューラがタスクを配置できません。原因は、`reason` フィールドに説明されます。

このサービスイベントが生成される一般的な原因は、タグがサービスに適用されてはいるが、ユーザーまたはロールがリージョンで新しい Amazon リソースネーム (ARN) 形式にオプトインしていないためです。詳細については、「Amazon リソースネーム (ARN) と ID」を参照してください。もう 1 つの一般的な原因は、Amazon ECS が提供されたタスク IAM ロールを継承できなかったことです。

`SERVICE_HEALTH_UNKNOWN`

サービスがタスクのヘルスデータを記述できませんでした。

`SERVICE_DEPLOYMENT_FAILED`

サービスデプロイが定常状態に達しませんでした。これは、CloudWatch がトリガーされたとき、またはサーキットブレーカーがサービスデプロイの失敗を検出したときに発生します。

Exampleサービス定常状態イベント

サービス定常状態イベントは、次の形式で配信されます。EventBridge パラメータの詳細については、「Amazon EventBridge ユーザーガイド」の「EventBridge イベント」を参照してください。

```
{
    "version": "0",
    "id": "af3c496d-f4a8-65d1-70f4-a69d52e9b584",
    "detail-type": "ECS Service Action",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2019-11-19T19:27:22Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
        "eventType": "INFO",
        "eventName": "SERVICE_STEADY_STATE",
        "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/default",
        "createdAt": "2019-11-19T19:27:22.695Z"
    }
}
```

Exampleキャパシティープロバイダー定常状態イベント

キャパシティプロバイダーの定常状態イベントは、次の形式で配信されます。

```
{
    "version": "0",
    "id": "b9baa007-2f33-0eb1-5760-0d02a572d81f",
    "detail-type": "ECS Service Action",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2019-11-19T19:37:00Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
        "eventType": "INFO",
        "eventName": "CAPACITY_PROVIDER_STEADY_STATE",
        "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/default",
        "capacityProviderArns": [
            "arn:aws:ecs:us-west-2:111122223333:capacity-provider/ASG-tutorial-
capacity-provider"
        ],
        "createdAt": "2019-11-19T19:37:00.807Z"
```

```
    }
}
```

Exampleサービスタスク開始障害イベント

サービスタスク開始障害イベントは、次の形式で配信されます。

```
{
    "version": "0",
    "id": "57c9506e-9d21-294c-d2fe-e8738da7e67d",
    "detail-type": "ECS Service Action",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2019-11-19T19:55:38Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
        "eventType": "WARN",
        "eventName": "SERVICE_TASK_START_IMPAIRED",
        "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/default",
        "createdAt": "2019-11-19T19:55:38.725Z"
    }
}
```

Exampleサービスタスク配置失敗イベント

サービスタスク配置失敗イベントは、次の形式で配信されます。詳細については、「Amazon EventBridge ユーザーガイド」の「EventBridge イベント」を参照してください。

次の例では、タスクは FARGATE_SPOT キャパシティープロバイダーを使用しようとしましたが、サービススケジューラは Fargate Spot キャパシティーを取得できませんでした。

```
{
    "version": "0",
    "id": "ddca6449-b258-46c0-8653-e0e3a6d0468b",
    "detail-type": "ECS Service Action",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2019-11-19T19:55:38Z",
    "region": "us-west-2",
```

```
    "resources": [
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
        "eventType": "ERROR",
        "eventName": "SERVICE_TASK_PLACEMENT_FAILURE",
        "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/default",
        "capacityProviderArns": [
            "arn:aws:ecs:us-west-2:111122223333:capacity-provider/FARGATE_SPOT"
        ],
        "reason": "RESOURCE:FARGATE",
        "createdAt": "2019-11-06T19:09:33.087Z"
    }
}
```

次の EC2 の例では、コンテナインスタンス 2dd1b186f39845a584488d2ef155c131 でタスクを起動しようとしたが、CPU が不十分なため、サービススケジューラがタスクを実行できませんでした。

```
{
  "version": "0",
  "id": "ddca6449-b258-46c0-8653-e0e3a6d0468b",
  "detail-type": "ECS Service Action",
  "source": "aws.ecs",
  "account": "111122223333",
  "time": "2019-11-19T19:55:38Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
  ],
  "detail": {
    "eventType": "ERROR",
    "eventName": "SERVICE_TASK_PLACEMENT_FAILURE",
    "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/default",
    "containerInstanceArns": [
    "arn:aws:ecs:us-west-2:111122223333:container-instance/
default/2dd1b186f39845a584488d2ef155c131"
    ],
    "reason": "RESOURCE:CPU",
    "createdAt": "2019-11-06T19:09:33.087Z"
  }
}
```

Amazon ECS サービスデプロイ状態変更イベント

Amazon ECS は、詳細タイプの [ECS Deployment State Change] を使用してサービスデプロイ状態変更イベントを送信します。以下は、Amazon ECS サービスデプロイ状態変更イベントの Eventbridgeルールを作成するために使用されるイベントパターンです。EventBridge ルールの作成の詳細については、「Amazon EventBridge ユーザーガイド」の「Amazon EventBridge の開始方法」を参照してください。

```
{
    "source": [
        "aws.ecs"
    ],
    "detail-type": [
        "ECS Deployment State Change"
    ]
}
```

Amazon ECS は、INFO、と ERROR イベントタイプのイベントを送信します。詳細については、Amazon ECS サービスアクションイベントを参照してください。

次に、サービスデプロイ状態の変更イベントを示します。

SERVICE_DEPLOYMENT_IN_PROGRESS

サービスデプロイは進行中です。このイベントは、初期デプロイとロールバックデプロイの両方で送信されます。

SERVICE_DEPLOYMENT_COMPLETED

サービスのデプロイが完了しました。このイベントは、デプロイ後にサービスが定常状態になると、送信されます。

SERVICE_DEPLOYMENT_FAILED

サービスのデプロイに失敗しました。このイベントは、デプロイサーキットブレーカーロジックが有効になっているサービスに対して送信されます。

Exampleサービスデプロイ進行中イベント

サービスデプロイ進行中イベントは、初期デプロイとロールバックデプロイの両方の開始時に配信されます。両者の違いは reason フィールドにあります。EventBridge パラメータの詳細については、

「Amazon EventBridge ユーザーガイド」の「AWS サービスイベントメタデータ」を参照してください。

次に、初期デプロイが開始された場合の出力例を示します。

```
{
   "version": "0",
   "id": "ddca6449-b258-46c0-8653-e0e3a6EXAMPLE",
   "detail-type": "ECS Deployment State Change",
   "source": "aws.ecs",
   "account": "111122223333",
   "time": "2020-05-23T12:31:14Z",
   "region": "us-west-2",
   "resources": [
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
   ],
   "detail": {
        "eventType": "INFO",
        "eventName": "SERVICE_DEPLOYMENT_IN_PROGRESS",
        "deploymentId": "ecs-svc/123",
        "updatedAt": "2020-05-23T11:11:11Z",
        "reason": "ECS deployment deploymentId in progress."
   }
}
```

次に、ロールバックデプロイが開始された場合の出力例を示します。reason フィールドには、サービスがロールバックされるデプロイの ID を指定します。

```
{
   "version": "0",
   "id": "ddca6449-b258-46c0-8653-e0e3aEXAMPLE",
   "detail-type": "ECS Deployment State Change",
   "source": "aws.ecs",
   "account": "111122223333",
   "time": "2020-05-23T12:31:14Z",
   "region": "us-west-2",
   "resources": [
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
   ],
   "detail": {
        "eventType": "INFO",
        "eventName": "SERVICE_DEPLOYMENT_IN_PROGRESS",
        "deploymentId": "ecs-svc/123",
```

```
        "updatedAt": "2020-05-23T11:11:11Z",
        "reason": "ECS deployment circuit breaker: rolling back to
 deploymentId deploymentID."
    }
}
```

Exampleサービスデプロイ完了イベント

サービスデプロイ完了状態イベントは、次の形式で配信されます。詳細については、「タスクを置き換えて Amazon ECS サービスをデプロイする」を参照してください。

```
{
   "version": "0",
   "id": "ddca6449-b258-46c0-8653-e0e3aEXAMPLE",
   "detail-type": "ECS Deployment State Change",
   "source": "aws.ecs",
   "account": "111122223333",
   "time": "2020-05-23T12:31:14Z",
   "region": "us-west-2",
   "resources": [
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
   ],
   "detail": {
        "eventType": "INFO",
        "eventName": "SERVICE_DEPLOYMENT_COMPLETED",
        "deploymentId": "ecs-svc/123",
        "updatedAt": "2020-05-23T11:11:11Z",
        "reason": "ECS deployment deploymentID completed."
   }
}
```

Exampleサービスデプロイ失敗イベント

サービスデプロイ失敗イベントは、次の形式で配信されます。サービスデプロイ失敗状態イベントは、デプロイサーキットブレーカーロジックが有効になっているサービスに対してのみ送信されます。詳細については、「タスクを置き換えて Amazon ECS サービスをデプロイする」を参照してください。

```
{
   "version": "0",
   "id": "ddca6449-b258-46c0-8653-e0e3aEXAMPLE",
   "detail-type": "ECS Deployment State Change",
```

```
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2020-05-23T12:31:14Z",
    "region": "us-west-2",
    "resources": [
         "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
         "eventType": "ERROR",
         "eventName": "SERVICE_DEPLOYMENT_FAILED",
         "deploymentId": "ecs-svc/123",
         "updatedAt": "2020-05-23T11:11:11Z",
         "reason": "ECS deployment circuit breaker: task failed to start."
    }
}
```

Amazon ECS サービスデプロイフック状態変更イベント

Amazon ECS は、詳細タイプ [ECS フック状態変更] を使用してフック状態変更イベントを送信します。これらのイベントは、デプロイライフサイクルフックのステータスが変更されたときに出力されます。Amazon ECS フック状態変更イベントの EventBridge ルールを作成するために使用されるイベントパターンを次に示します。

```
{
    "source": [
        "aws.ecs"
    ],
    "detail-type": [
        "ECS Hook State Change"
    ]
}
```

フック状態変更イベントを次に示します。

`HOOK_IN_PROGRESS`

フックが進行中です。

`HOOK_AWAITING_ACTION`

一時停止フックが開始され、完了するにはアクションが必要です。`ContinueServiceDeployment` を呼び出してデプロイを続行またはロールバックします。

HOOK_SUCCEEDED

フックは正しく完了しています。一時停止フックの場合、これは CONTINUE アクションで ContinueServiceDeployment が呼び出されたことを意味します。

HOOK_FAILED

フックが失敗しました。一時停止フックの場合、これは ROLLBACK アクションで ContinueServiceDeployment が呼び出されたことを意味します。

HOOK_TIMED_OUT

完了することなくフックがタイムアウトしました。

Exampleアクションイベントを待機中のフック

一時停止フックが起動し、アクションを待機しているときのイベントの例を次に示します。

```
{
    "version": "0",
    "id": "3329f79b-3dca-07f8-b1c2-5fe99f0b5e87",
    "detail-type": "ECS Hook State Change",
    "source": "aws.ecs",
    "account": "123456789012",
    "time": "2026-03-05T15:54:41Z",
    "region": "us-west-2",
    "resources": ["arn:aws:ecs:us-west-2:123456789012:service-deployment/my-cluster/my-
service/0EYSiB0qap8xf0N76FsbE"],
    "detail": {
        "eventType": "INFO",
        "eventName": "HOOK_AWAITING_ACTION",
        "hookId": "ecs-pause-e7tK9G_WRJqNF_EOMjztDXfKenlJuEUVjsNStf4WLKw",
        "hookType": "PAUSE",
        "expiresAt": "2026-03-05T16:04:41Z",
        "lifecycleStage": "POST_TEST_TRAFFIC_SHIFT",
        "clusterArn": "arn:aws:ecs:us-west-2:123456789012:cluster/my-cluster",
        "serviceArn": "arn:aws:ecs:us-west-2:123456789012:service/my-cluster/my-
service",
        "serviceDeploymentArn": "arn:aws:ecs:us-west-2:123456789012:service-deployment/
my-cluster/my-service/0EYSiB0qap8xf0N76FsbE",
        "updatedAt": "2026-03-05T15:54:41.618059641Z"
    }
}
```

Amazon ECS コンテナインスタンスのヘルス変更イベント

以下の場合に、コンテナインスタンスのヘルス変更イベントが発生します。

Amazon ECS コンテナエージェントがコンテナランタイムのヘルスの変化を検出した場合。

コンテナエージェントは、コンテナインスタンスのコンテナランタイム (Docker デーモンなど) の状態を定期的に評価します。ランタイムの無反応や復旧が発生すると、エージェントはその変化を報告し、イベントが生成されます。

可用性: EC2 起動タイプ。

Amazon ECS コンテナエージェントが高速コンピューティングデバイスのヘルスの変化を検出した場合。

GPU などの高速コンピューティングデバイスがあるインスタンスでは、コンテナエージェントはデバイスのヘルスをモニタリングします。デバイスの障害や復旧が発生すると、イベントが生成されます。

可用性: Amazon ECS マネージドインスタンス起動タイプ。

Amazon ECS コンテナエージェントがデーモンのヘルスの変化を検出した場合。

コンテナエージェントは、コンテナインスタンスで必要なデーモンタスクをモニタリングします。デーモンタスクが失敗すると、イベントが生成されます。

可用性: Amazon ECS マネージドインスタンス起動タイプ。

Amazon ECS コンテナエージェントが、Amazon ECS コントロールプレーンへの接続を失います。

Amazon ECS は、コンテナエージェントと Amazon ECS コントロールプレーン間の接続を継続的にモニタリングします。コンテナインスタンスがしきい値を超えて切断されたままの場合、Amazon ECS はそのインスタンスを障害ありとしてマークします。

可用性: Amazon ECS マネージドインスタンス、AWS Fargate、および Amazon EC2 起動タイプ。

コンテナインスタンスの全体的なヘルスステータスが変化した場合。

Amazon ECS は、個々のヘルスチェック結果から総合ヘルスステータスを計算します。総合ステータスが遷移すると、イベントが生成されます。

可用性: Amazon ECS マネージドインスタンスと EC2 起動タイプ。

コンテナインスタンスのヘルス変化イベントの例

コンテナインスタンスのヘルス変化イベントは、次の形式で配信されます。パラメータの詳細については、「ユーザーガイド」の「AWS サービスイベントメタデータ」を参照してください。

コンテナランタイムヘルスイベントのサンプル

```
{
    "version": "0",
    "id": "EXAMPLE-48f8-c0ff-0dab-EXAMPLE2ac34",
    "detail-type": "ECS Container Instance Health Change",
    "source": "aws.ecs",
    "account": "123456789012",
    "time": "2026-04-11T00:49:09Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:123456789012:container-instance/my-cluster/
a1b2c3d4EXAMPLE"
    ],
    "detail": {
        "clusterArn": "arn:aws:ecs:us-west-2:123456789012:cluster/my-cluster",
        "containerInstanceArn": "arn:aws:ecs:us-west-2:123456789012:container-instance/
my-cluster/a1b2c3d4EXAMPLE",
        "capacityProviderName": null,
        "ec2InstanceId": "i-0abcdef1234567890",
        "overallStatus": "OK",
        "healthChecks": [
            {
                "type": "CONTAINER_RUNTIME",
                "status": "OK",
                "lastStatusChange": "2026-04-11T00:48:53Z",
                "lastUpdated": "2026-04-11T00:48:09Z",
                "statusReason": null
            }
        ]
    }
}
```

高速コンピューティングヘルスイベントのサンプル

```
{
    "version": "0",
    "id": "EXAMPLE-46c3-a48a-1bef-EXAMPLE72f3c",
```

```
    "detail-type": "ECS Container Instance Health Change",
    "source": "aws.ecs",
    "account": "123456789012",
    "time": "2026-04-11T00:49:09Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:123456789012:container-instance/my-cluster/
a1b2c3d4EXAMPLE"
    ],
    "detail": {
        "clusterArn": "arn:aws:ecs:us-west-2:123456789012:cluster/my-cluster",
        "containerInstanceArn": "arn:aws:ecs:us-west-2:123456789012:container-instance/
my-cluster/a1b2c3d4EXAMPLE",
        "capacityProviderName": "my-capacity-provider",
        "ec2InstanceId": "i-0abcdef1234567890",
        "overallStatus": "IMPAIRED",
        "healthChecks": [
            {
                "type": "ACCELERATED_COMPUTE",
                "status": "IMPAIRED",
                "lastStatusChange": "2026-04-11T00:48:53Z",
                "lastUpdated": "2026-04-11T00:48:09Z",
                "statusReason": "XID_46"
            }
        ]
    }
}
```

デーモンヘルスイベントのサンプル

```
{
    "version": "0",
    "id": "EXAMPLE-febc-b5e3-054b-EXAMPLE1a1f4f",
    "detail-type": "ECS Container Instance Health Change",
    "source": "aws.ecs",
    "account": "123456789012",
    "time": "2026-04-11T00:49:09Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:123456789012:container-instance/my-cluster/
a1b2c3d4EXAMPLE"
    ],
    "detail": {
```

```
        "clusterArn": "arn:aws:ecs:us-west-2:123456789012:cluster/my-cluster",
        "containerInstanceArn": "arn:aws:ecs:us-west-2:123456789012:container-instance/
my-cluster/a1b2c3d4EXAMPLE",
        "capacityProviderName": "my-capacity-provider",
        "ec2InstanceId": "i-0abcdef1234567890",
        "overallStatus": "IMPAIRED",
        "healthChecks": [
            {
                "type": "DAEMON",
                "status": "IMPAIRED",
                "lastStatusChange": "2026-04-11T00:48:53Z",
                "lastUpdated": "2026-04-11T00:48:09Z",
                "statusReason": "Daemon task: EXAMPLE-4af2-4b3c-b9c1-EXAMPLEb2bbf is
 unhealthy"
            }
        ]
    }
}
```

エージェント接続ヘルスイベントのサンプル

```
{
    "version": "0",
    "id": "EXAMPLE-7d2a-9f1c-4b6e-EXAMPLE9c8d1",
    "detail-type": "ECS Container Instance Health Change",
    "source": "aws.ecs",
    "account": "123456789012",
    "time": "2026-07-29T22:34:13Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:123456789012:container-instance/my-cluster/
a1b2c3d4EXAMPLE"
    ],
    "detail": {
        "clusterArn": "arn:aws:ecs:us-west-2:123456789012:cluster/my-cluster",
        "containerInstanceArn": "arn:aws:ecs:us-west-2:123456789012:container-instance/
my-cluster/a1b2c3d4EXAMPLE",
        "capacityProviderName": "my-capacity-provider",
        "ec2InstanceId": "i-0abcdef1234567890",
        "overallStatus": "IMPAIRED",
        "healthChecks": [
            {
                "type": "AGENT_CONNECTIVITY",
```

```
                "status": "IMPAIRED",
                "lastStatusChange": "2026-07-29T22:34:13Z",
                "lastUpdated": "2026-07-29T22:34:09Z",
                "statusReason": "Agent disconnected since 2026-07-29T22:34:13Z"
            }
        ]
    }
}
```

Amazon ECS イベントの処理

Amazon ECS は、少なくとも 1 回 の割合でイベントを送信します。つまり、あるイベントの複数のコピーを受け取る場合があるということです。さらに、イベントは発生順にイベントリスナーに送信されない場合があります。

イベントを適切に順序付けられるように、各イベントの `detail` セクションには `version` プロパティが含まれています。リソースの状態が変わるたびに、この `version` はインクリメントされます。重複するイベントには、`version` オブジェクト内で同じ `detail` バージョンがあります。EventBridge で Amazon ECS コンテナインスタンスおよびタスク状態をレプリケートする場合は、Amazon ECS API によって報告されるリソースのバージョンと、そのリソースについて EventBridge で報告される `version` を比較して、イベントストリームでのバージョンが最新であることを確認できます。バージョンプロパティ番号が高いイベントは、バージョン番号が低いイベントより後で発生したものとして処理されます。

例: AWS Lambda 関数でのイベントの処理

次の Python 3.9 で記述された Lambda 関数の例では、タスク状態変更イベントとコンテナインスタンス状態変更イベントの両方をキャプチャし、2 つの Amazon DynamoDB テーブルのいずれかに保存します。

- ECSCtrInstanceState – コンテナインスタンスの最新状態を保存します。テーブル ID は、コンテナインスタンスの `containerInstanceArn` 値です。
- ECSTaskState – タスクの最新状態を保存します。テーブル ID は、タスクの `taskArn` 値です。

```
import json
import boto3

def lambda_handler(event, context):
    id_name = ""
```

```
    new_record = {}

    # For debugging so you can see raw event format.
    print('Here is the event:')
    print((json.dumps(event)))

    if event["source"] != "aws.ecs":
       raise ValueError("Function only supports input from events with a source type
of: aws.ecs")

    # Switch on task/container events.
    table_name = ""
    if event["detail-type"] == "ECS Task State Change":
        table_name = "ECSTaskState"
        id_name = "taskArn"
        event_id = event["detail"]["taskArn"]
    elif event["detail-type"] == "ECS Container Instance State Change":
        table_name = "ECSCtrInstanceState"
        id_name =  "containerInstanceArn"
        event_id = event["detail"]["containerInstanceArn"]
    else:
        raise ValueError("detail-type for event is not a supported type. Exiting
without saving event.")

    new_record["cw_version"] = event["version"]
    new_record.update(event["detail"])

    # "status" is a reserved word in DDB, but it appears in containerPort
    # state change messages.
    if "status" in event:
        new_record["current_status"] = event["status"]
        new_record.pop("status")


    # Look first to see if you have received a newer version of an event ID.
    # If the version is OLDER than what you have on file, do not process it.
    # Otherwise, update the associated record with this latest information.
    print("Looking for recent event with same ID...")
    dynamodb = boto3.resource("dynamodb", region_name="us-east-1")
    table = dynamodb.Table(table_name)
    saved_event = table.get_item(
        Key={
            id_name : event_id
        }
```

```
    )
    if "Item" in saved_event:
        # Compare events and reconcile.
        print(("EXISTING EVENT DETECTED: Id " + event_id + " - reconciling"))
        if saved_event["Item"]["version"] < event["detail"]["version"]:
            print("Received event is a more recent version than the stored event -
 updating")
            table.put_item(
                Item=new_record
            )
        else:
            print("Received event is an older version than the stored event -
 ignoring")
    else:
        print(("Saving new event - ID " + event_id))

        table.put_item(
            Item=new_record
        )
```

以下の Fargate の例では、Python 3.9 で書かれた Lambda 関数で、タスク状態変更イベントを取得し、以下の Amazon DynamoDB テーブルに保存しています。

```
import json
import boto3

def lambda_handler(event, context):
    id_name = ""
    new_record = {}

    # For debugging so you can see raw event format.
    print('Here is the event:')
    print((json.dumps(event)))

    if event["source"] != "aws.ecs":
       raise ValueError("Function only supports input from events with a source type
 of: aws.ecs")

    # Switch on task/container events.
    table_name = ""
    if event["detail-type"] == "ECS Task State Change":
        table_name = "ECSTaskState"
        id_name = "taskArn"
```

```
        event_id = event["detail"]["taskArn"]
    else:
        raise ValueError("detail-type for event is not a supported type. Exiting
without saving event.")

    new_record["cw_version"] = event["version"]
    new_record.update(event["detail"])

    # "status" is a reserved word in DDB, but it appears in containerPort
    # state change messages.
    if "status" in event:
        new_record["current_status"] = event["status"]
        new_record.pop("status")


    # Look first to see if you have received a newer version of an event ID.
    # If the version is OLDER than what you have on file, do not process it.
    # Otherwise, update the associated record with this latest information.
    print("Looking for recent event with same ID...")
    dynamodb = boto3.resource("dynamodb", region_name="us-east-1")
    table = dynamodb.Table(table_name)
    saved_event = table.get_item(
        Key={
            id_name : event_id
        }
    )
    if "Item" in saved_event:
        # Compare events and reconcile.
        print(("EXISTING EVENT DETECTED: Id " + event_id + " - reconciling"))
        if saved_event["Item"]["version"] < event["detail"]["version"]:
            print("Received event is a more recent version than the stored event -
updating")
            table.put_item(
                Item=new_record
            )
        else:
            print("Received event is an older version than the stored event -
ignoring")
    else:
        print(("Saving new event - ID " + event_id))

        table.put_item(
            Item=new_record
```

)

オブザーバビリティが強化された Container Insights を使用し、Amazon ECS コンテナを監視する

Container Insights は、コンテナ化されたアプリケーションとマイクロサービスからメトリクスと口グを収集、集計、要約します。オブザーバビリティが強化された Container Insights は、すべての Container Insights メトリクスに加え、追加のタスクおよびコンテナメトリクスも提供します。

Container Insights は、クラスターで実行中のすべてのコンテナを検出し、パフォーマンススタックの各レイヤーでパフォーマンスデータを収集します。運用データは、パフォーマンスログイベントとして収集されます。これらは、高濃度データを大規模に取り込み、保存できる、構造化された JSON スキーマを使用するエントリです。CloudWatch はこのデータから、クラスター、サービスおよびサービスレベルで、高レベルの集約されたメトリクスを CloudWatch メトリクスとして作成します。このメトリクスには、CPU、メモリ、ディスク、ネットワークなどのリソース使用率が含まれます。メトリクスは、CloudWatch 自動ダッシュボードで使用できます。

このメトリクスは、指定した期間中にタスクが実行されているリソースのみを反映します。例えば、クラスターに 1 つのサービスがあるが、このサービスに `RUNNING` 状態のタスクがない場合、CloudWatch に送信されるメトリクスはありません。2 つのサービスがあり、1 つに実行中のタスクがあるが、別の 1 つに実行中のタスクがない場合、実行中のタスクがあるサービスのメトリクスのみが送信されます。

2024 年 12 月 2 日、AWS で Amazon ECS 用にオブザーバビリティが強化された Container Insights がリリースされました。このバージョンでは、Amazon EC2 および Fargate を使用して Amazon ECS クラスター用に強化されたオブザーバビリティがサポートされます。Amazon ECS でオブザーバビリティが強化された Container Insights を設定したら、Container Insights はクラスターレベルから環境内のコンテナレベルまで詳細なインフラストラクチャテレメトリを自動的に収集し、これらの重要なパフォーマンスデータを厳選されたダッシュボードに表示し、オブザーバビリティ設定に伴う負担を軽減します。オブザーバビリティが強化された Container Insights を設定する方法の詳細については、「オブザーバビリティが強化された Container Insights」を参照してください。

コンテナ環境で詳細な可視性を提供し、解決までの平均時間を短縮するため、Container Insights ではなく、オブザーバビリティが強化された Container Insights を使用することをお勧めします。詳細については、「Amazon CloudWatch ユーザーガイド」の「Amazon ECS 用にオブザーバビリティメトリクスが強化された Container Insights」を参照してください。

表示できるイベントは、Amazon ECS が Amazon EventBridge に送信するイベントです。詳細については、「Amazon ECS イベント」を参照してください。

> **Important**
>
> CloudWatch Container インサイトによって収集されたメトリクスは、カスタムメトリクスとして課金されます。CloudWatch の料金の詳細については、 CloudWatch の料金をご覧ください。

Amazon ECS マネージドインスタンスの追加メトリクス

次の表に、Container Insights を使用する際に Amazon ECS マネージドインスタンスで使用できる追加のメトリクスを示します。

メトリクス	説明	ディメンション	Unit
`InstanceOSFilesystemUtilization`	ディスクスペース合計に対する使用済みの割合 (OS ボリューム)。	`ClusterName` – Container Insights が有効になっている場合 `ClusterName` 、`CapacityProviderName` – Container Insights が有効になっている場合 `ClusterName` 、`CapacityProviderName` 、`ContainerInstanceId` 、`EC2InstanceId` – EnhancedContainerInsights が有効になっている場合	割合 (%)
`InstanceDataFilesystemUtilization`	ディスクスペース合計に対する使用済みの割合 (データボリューム)。	`ClusterName` – Container Insights が有効になっている場合	割合 (%)

メトリクス	説明	ディメンション	Unit
		`ClusterName` 、`CapacityProviderName` – Container Insights が有効になっている場合 `ClusterName` 、`CapacityProviderName` 、`ContainerInstanceId` 、`EC2InstanceId` – EnhancedContainerInsights が有効になっている場合	
`InstanceGPULimit`	インスタンスで使用可能な GPU の合計数。NVIDIA GPU 対応 Amazon EC2 インスタンスタイプを実行する Amazon ECS マネージドインスタンスでのみ利用可能です。	`ClusterName` - EnhancedContainerInsights が有効になっている場合 `ClusterName` 、`CapacityProviderName` - EnhancedContainerInsights が有効になっている場合 `ClusterName` 、`CapacityProviderName` 、`ContainerInstanceId` 、`EC2InstanceId` – EnhancedContainerInsights が有効になっている場合	カウント

メトリクス	説明	ディメンション	Unit
`InstanceGPUUsageTotal`	インスタンスで実行中のタスクに現在割り当てられている GPU の数。NVIDIA GPU 対応 Amazon EC2 インスタンスタイプを実行する Amazon ECS マネージドインスタンスでのみ利用可能です。	`ClusterName` - EnhancedContainerInsights が有効になっている場合 `ClusterName` 、`CapacityProviderName` - EnhancedContainerInsights が有効になっている場合 `ClusterName` 、`CapacityProviderName` 、`ContainerInstanceId` 、`EC2InstanceId` – EnhancedContainerInsights が有効になっている場合	カウント

GPU アプリケーションの再起動メトリクス

NVIDIA GPU 対応 Amazon EC2 インスタンスタイプを実行している Amazon ECS マネージドインスタンスの場合、`TaskGPURestartAppXidCount` と `ContainerGPURestartAppXidCount` メトリクスは、即時アクション解決バケットが `RESTART_APP` である NVIDIA Xid エラーをカウントします。これらの Xid は、NVIDIA がインスタンスではなく影響を受けるアプリケーションの再起動を推奨するアプリケーションレベルの障害を示します。これらのメトリクスを使用して、基盤となるハードウェア障害を暗示することなく、特定のタスクまたはコンテナに影響を与える一時的な GPU 障害を検出します。

次の Xid コードが含まれています。完全なリストと説明については、NVIDIA Xid Errors ドキュメントを参照してください。

xid	エラータイプ
8	GPU が処理を停止した
11	無効または破損したプッシュバッファストリーム

xid	エラータイプ
13	グラフィックスエンジンの例外
25	ECC ページの廃止が進行中
31	GPU メモリページの障害
32	無効または破損したプッシュバッファストリーム
39	バスエラー
40	ビデオプロセッサの例外
41	不測の障害
60	ビデオプロセッサの例外
68	ビデオプロセッサの例外
69	グラフィックスエンジンクラスエラー
70	CE ユーザーチャネルエラー
71	GPU セマフォタイムアウト
72	GPU セマフォアクセスエラー
75	Inforom ページのブラックリストイベント
76	エンジンエラーの表示
77	エンジンエラーの表示
80	破損データが GPU に送信された
82	NVJPG エラー
83	NVDEC エラー
84	SLI リンクの不一致

xid	エラータイプ
85	リソースの制約
86	オペレーティングシステムエラー
88	NVDEC エラー
89	NVENC エラー
94	ECC エラーが含まれた
96	NVDEC エラー
97	NVDEC エラー
98	NVDEC エラー
99	NVJPG エラー
100	NVJPG エラー
101	NVJPG エラー
102	NVJPG エラー
103	GSP RPC タイムアウト
104	GSP の停止
105	GSP エラー
126	C2C NVLink 再生エラー
127	C2C NVLink エラー
128	NVLink エラー
129	NVLink 復旧エラー
130	NVLink の致命的エラー

xid	エラータイプ
131	NVLink の非致命的エラー
132	NVLink エラー
133	NVLink エラー
134	NVLink エラー
135	NVLink エラー
139	GPU メモリページ廃止記録イベント

Amazon ECS マネージドインスタンスのモニタリング

モニタリングは、Amazon ECS マネージドインスタンスのワークロードの信頼性、可用性、パフォーマンスを維持する上で重要な要素です。AWS には、コンテナ化されたアプリケーションとインフラストラクチャのモニタリングに役立つツールとサービスがいくつか用意されています。

Container Insights のモニタリング

CloudWatch Container Insights では、コンテナ化されたアプリケーションとマイクロサービスの包括的なモニタリングが行えます。Container Insights は、Amazon ECS マネージドインスタンスで動作しているコンテナ化されたアプリケーションとマイクロサービスからメトリクスとログを自動的に収集、集計、要約します。

Container Insights は、クラスター、サービス、タスクのレベルでメトリクスを収集し、以下を可視化します。

- CPU とメモリの使用率
- ネットワークパフォーマンスメトリクス
- ストレージの使用率
- タスクとサービスのパフォーマンス

メトリクスは CloudWatch ダッシュボードで使用でき、パフォーマンスの問題に対するアラームと自動応答を作成するために使用できます。Container Insights では、問題をすばやく特定してトラブルシューティングするのに役立つモニタリング機能も強化されています。

> **Note**
>
> Container Insights を入手するには、追加料金がかかります。料金の詳細については、「Amazon CloudWatch Pricing」を参照してください。

GPU モニタリング

NVIDIA GPU 対応 Amazon EC2 インスタンスタイプを実行する Amazon ECS マネージドインスタンスの場合、オブザーバビリティが強化された Container Insights は、コンテナ、タスク、インスタンスレベルで NVIDIA Data Center GPU Manager (DCGM) から GPU メトリクスを収集します。GPU メトリクスは基本的な Container Insights では収集されません。GPU テレメトリにアクセスするためのオブザーバビリティを強化します。

エージェントの追加インストールは不要です。オブザーバビリティが強化された Container Insights がクラスターで有効になっている場合、GPU メトリクスは、サポートされているインスタンスタイプで自動的に収集されます。

GPU メトリクスとディメンションのリストについては、「オブザーバビリティメトリクスが強化された Amazon ECS Container Insights」を参照してください。

インスタンスのモニタリング

Amazon ECS マネージドインスタンスワークロードをサポートする基盤インフラストラクチャを監視するには、CloudWatch 経由で利用できる Amazon EC2 メトリクスを使用できます。

Amazon ECS マネージドインスタンスには、次の 2 つの Amazon EBS ボリュームが備わっています。

- OS ファイルシステムに使用されるルートボリューム
- アプリケーションが使用するデータボリューム

Container Insights を有効にすると、Amazon ECS がインスタンスレベルでの OS とデータファイルシステムの使用率メトリクスを自動的に発行します。

Amazon ECS マネージドインスタンスに使用できるメトリクスには以下が含まれます。

- Amazon EC2 メトリクス: CPU 使用率、ネットワークパフォーマンス、ディスク操作、ステータスチェック。詳細については、「CloudWatch を使用したインスタンスのモニタリング」を参照してください。
- Amazon ECS メトリクス (Container Insights 有効時): OS とデータのボリュームファイルシステム使用率。詳細については、「Amazon ECS Container Insights メトリクス」を参照してください。
- Amazon EBS メトリクス: IOPS、スループット、読み取りおよび書き込みレイテンシー。詳細については、「Amazon EBS CloudWatch メトリクス」を参照してください。

これらのメトリクスは、エージェントを手動でインストールしなくても自動的に利用できます。

Amazon ECS マネージドインスタンスの詳細なモニタリング

CloudWatch には、基本モニターリングと詳細モニターリングの 2 つのカテゴリのモニターリングが用意されています。デフォルトでは、マネージドインスタンスは基本モニタリングを使用するように設定されています。オプションで詳細モニタリングを有効にすることで、運用上の問題をより迅速に特定して対処できるようにすることができます。Amazon ECS マネージドインスタンスのキャパシティプロバイダーを作成または更新するときに、詳細モニタリングをオンまたはオフにできます。

マネージドインスタンスで詳細モニタリングを有効にしても、そのインスタンスにアタッチされている Amazon EBS ボリュームのモニタリングには影響しません。

次のテーブルはマネージドインスタンスの基本モニタリングと詳細モニタリングの違いを示しています。

モニタリングタイプ	説明	料金
基本モニタリング	ステータスチェックメトリクスは 1 分間隔で利用できます。その他のメトリクスはすべて 5 分間隔で利用できます。	料金は発生しません。
詳細モニターリング	ステータスチェックメトリクスを含むすべてのメトリクスは1 分間隔で利用できます。このレベルのデータを取得するには、マネージドインスタンスのデータ取得を明確に有効にする必要があります。詳細モニタリングを有効にしたマネージドインスタンスでは同様のマネージドインスタンスグループの集約データを取得することもできます。	料金は、Amazon ECS マネージドインスタンスが CloudWatch に送信するメトリクスごとに発生します。データストレージに対しては料金が発生しま

モニタリングタイプ	説明	料金
		せん。詳細については、CloudWatch の料金ページの、「有料利用枠」および「例 1 – EC2 の詳細モニタリング」を参照してください。

必要なアクセス許可

マネージドインスタンスの詳細モニタリングを有効にするにはユーザーに `MonitorInstances` API アクションを使用するための許可が必要です。マネージドインスタンスの詳細モニタリングをオフにするにはユーザーに `UnmonitorInstances` API アクションを使用するための許可が必要です。

コンテナのヘルスチェックを使用して Amazon ECS タスク状態を判定する

タスク定義を作成するときに、コンテナのヘルスチェックを設定できます。ヘルスチェックは、コンテナ上でローカルに実行され、アプリケーションのヘルスと可用性を検証するコマンドです。

Amazon ECS コンテナエージェントは、タスク定義で指定されたヘルスチェックについてのみ、モニタリングとレポートを行います。Amazon ECS は、コンテナイメージに組み込みた Docker ヘルスチェックについても、コンテナ定義で指定されていない場合はモニタリングしません。コンテナ定義で指定されているヘルスチェックのパラメータは、コンテナイメージ内に存在する Docker ヘルスチェックを上書きします。

タスク定義でヘルスチェックが定義されると、コンテナはコンテナ内でヘルスチェックプロセスを実行し、終了コードを評価してアプリケーションのヘルスを判定します。

ヘルスチェックは以下のパラメータで構成されています。

- コマンド – 正常かどうかを判定するためにコンテナが実行するコマンド。この文字列配列の先頭には、コマンド引数を直接実行するための `CMD`、またはコンテナのデフォルトシェルでコマンドを実行するための `CMD-SHELL` を付加できます。パイプ、リダイレクト、コマンドチェイニング、環境変数の拡張などのシェル機能が必要な場合は `CMD-SHELL` を使用します。例えば、`CMD-`

SHELL では、シェルが || 演算子を解釈する curl -f http://localhost/ || exit 1 のようなコマンドを使用できます。シェル解釈を必要としないシンプルなコマンドには CMD を使用します。

- 間隔 – 各ヘルスチェック間の時間間隔 (秒)。
- タイムアウト – 失敗したと見なされるまでの、ヘルスチェックが正常に終了するのを待つ時間 (秒)。
- 再試行回数 – コンテナが異常と見なされるまでに、失敗したヘルスチェックを再試行する回数。
- 開始期間 – 失敗したヘルスチェックの再試行が最大回数に達する前に、コンテナにブートストラップするまでの時間を与えるオプションの猶予期間です。

  ヘルスチェックが startPeriod 内で成功した場合、コンテナは正常であるとみなされ、その後の失敗は最大再試行回数にカウントされます。

タスク定義でヘルスチェックを指定する方法については、「ヘルスチェック」を参照してください。

以下はコンテナのヘルスステータスの取り得る値です。

- HEALTHY — コンテナのヘルスチェックが正常に完了しました。
- UNHEALTHY — コンテナのヘルスチェックが失敗しました。
- UNKNOWN — コンテナのヘルスチェックが評価中か、定義されていない、または Amazon ECS にコンテナのヘルスチェックのステータスがありません。

ヘルスチェックコマンドはコンテナ上で実行されます。そのため、コンテナイメージにそのコマンドを入れる必要があります。

ヘルスチェックは、localhost または 127.0.0.1 にあるコンテナのループバックインターフェイスを介してアプリケーションに接続します。0 である終了コードは成功を示し、ゼロ以外の終了コードは失敗を示します。

コンテナのヘルスチェックを使用する際は、以下の点を考慮してください。

- コンテナのヘルスチェックには、Amazon ECS コンテナエージェントのバージョン 1.17.0 以降が必要です。
- コンテナのヘルスチェックは、Linux プラットフォームバージョン 1.1.0 以降、または Windows プラットフォームバージョン 1.1.0 以降を使用している場合、Fargate タスクでサポートされます。

Amazon ECS がタスクの正常性を判定する方法

タスクの正常性を判定するのに考慮されるのは、必須で、タスク定義にヘルスチェックコマンドが含まれているコンテナだけです。

次のルールが順番に評価されます。

1. ある必須コンテナのステータスが UNHEALTHY の場合、そのタスクステータスが UNHEALTHY である。
2. ある必須コンテナのステータスが UNKNOWN の場合、そのタスクステータスが UNKNOWN である。
3. すべての必須コンテナのステータスが HEALTHY の場合、タスクステータスが HEALTHY である。

必須コンテナ 2 つでの次のタスクの正常性の例を挙げます。

コンテナ 1 の正常性	コンテナ 2 の正常性	タスクの正常性
UNHEALTHY	UNKNOWN	UNHEALTHY
UNHEALTHY	HEALTHY	UNHEALTHY
HEALTHY	UNKNOWN	UNKNOWN
HEALTHY	HEALTHY	HEALTHY

コンテナ 3 つでの次のタスクの正常性の例を挙げます。

コンテナ 1 の正常性	コンテナ 2 の正常性	コンテナ 3 の正常性	タスクの正常性
UNHEALTHY	UNKNOWN	UNKNOWN	UNHEALTHY
UNHEALTHY	UNKNOWN	HEALTHY	UNHEALTHY
UNHEALTHY	HEALTHY	HEALTHY	UNHEALTHY
HEALTHY	UNKNOWN	HEALTHY	UNKNOWN
HEALTHY	UNKNOWN	UNKNOWN	UNKNOWN

コンテナ 1 の正常性	コンテナ 2 の正常性	コンテナ 3 の正常性	タスクの正常性
HEALTHY	HEALTHY	HEALTHY	HEALTHY

エージェントの切断がヘルスチェックに及ぼす影響

Amazon ECS コンテナエージェントが Amazon ECS サービスから切断されても、コンテナが UNHEALTHY ステータスに移行することはありません。これは、エージェントの再起動中や一時的に使用不可になっている間もコンテナが実行され続けるように設計されているためです。ヘルスチェックのステータスは、Amazon ECS エージェントからの「最後に受信した」レスポンスであるため、コンテナが切断前に HEALTHY であると見なされていた場合、エージェントが再接続されて新たにヘルスチェックが行われるまではそのステータスが残ります。コンテナのヘルスチェックのステータスについては、いかなる仮定もありません。

Amazon ECS コンテナの正常性を表示する

コンテナの正常性は、コンソールで確認することも、API を使用して DescribeTasks レスポンスで確認することもできます。詳細については、「Amazon Elastic Container Service API リファレンス」の「DescribeTasks」を参照してください。

Amazon CloudWatch Logs などのコンテナにログ記録を使用する場合は、コンテナの正常性の出力をログに転送するようにヘルスチェックコマンドを設定できます。必ず 2&1 を使用して stdout と stderr の両方の情報を取得するようにしてください。次の例では、パイプ (>>) 演算子や論理 OR (||) 演算子などのシェル機能が必要なため CMD-SHELL を使用します。

```
"command": [
     "CMD-SHELL",
     "curl -f http://localhost/ >> /proc/1/fd/1 2>&1  || exit 1"
   ],
```

Amazon ECS コンテナインスタンスの正常性をモニタリングする

Amazon ECS は、コンテナインスタンスのヘルスモニタリングを提供します。Amazon ECS が、コンテナインスタンスがコンテナの実行を妨げる可能性のある問題を検出したか否かについて、すばやく判断できます。Amazon ECS は、エージェントバージョン 1.57.0 以降を使用して実行中のすべてのコンテナインスタンスに対して自動チェックを実行し、問題を特定します。コンテナインスタン

スのエージェントバージョンの検証に関する詳細については、Amazon ECS コンテナエージェントをアップデートする を参照してください。

AWS CLI バージョン 1.22.3 以降、または AWS CLI バージョン 2.3.6 以降を使用する必要があります。AWS CLI をアップデートする情報については、AWS Command Line Interface ユーザーガイドバージョン 2 の「AWS CLI の最新バージョンのインストールまたはアップデート」を参照してください。

コンテナインスタンスの正常性を表示するには、CONTAINER_INSTANCE_HEALTH オプションを使用して describe-container-instances を実行します。

overallStatus は、details 配列内の個々のヘルスチェックステータスによって決まります。最も重要度の高いステータスは、IMPAIRED、INSUFFICIENT_DATA、INITIALIZING および OK の順序で優先されます。

次に示すのは、overallStatus の有効な値です。

- OK – すべてのヘルスチェックに合格しています。
- IMPAIRED – 1 つ以上のヘルスチェックが失敗しました。
- INSUFFICIENT_DATA – 1 つ以上のヘルスチェックのデータが使用できない状態です。
- INITIALIZING – 1 つ以上のヘルスチェックが初期化されています。

ヘルスチェック type 値には以下が含まれます。

- CONTAINER_RUNTIME – コンテナランタイム (Docker デーモンなど) のヘルスをモニタリングします。EC2 起動タイプで使用できます。
- ACCELERATED_COMPUTE – GPUなどの高速コンピューティングデバイスのヘルスをモニタリングします。GPU に障害が発生した場合、statusReason には NVIDIA Xid エラーコードが XID_<number> 形式で含まれます。Amazon ECS マネージドインスタンス起動タイプで使用できます。
- DAEMON – コンテナインスタンスで必須のデーモンタスクのヘルスをモニタリングします。Amazon ECS マネージドインスタンス起動タイプで使用できます。
- AGENT_CONNECTIVITY – Amazon ECS コンテナエージェントと Amazon ECS コントロールプレーン間の接続をモニタリングします。Amazon ECS マネージドインスタンス、AWS Fargate、および Amazon EC2 起動タイプで使用できます。

イベントを通じてヘルスステータスの変化をモニタリングすることもできます。詳細については、「Amazon ECS コンテナインスタンスのヘルス変更イベント」を参照してください。

describe-container-instances の実行方法の例を次に示します。

```
aws ecs describe-container-instances \
     --cluster cluster_name \
     --container-instances 47279cd2cadb41cbaef2dcEXAMPLE \
     --include CONTAINER_INSTANCE_HEALTH
```

以下は、出力されたヘルスステータスオブジェクトの例です。

```
"healthStatus": {
 "overallStatus": "OK",
 "details": [{
  "type": "CONTAINER_RUNTIME",
  "status": "OK",
  "lastUpdated": "2021-11-10T03:30:26+00:00",
  "lastStatusChange": "2021-11-10T03:26:41+00:00"
 }]
}
```

コンテナインスタンスの異常

overallStatus が OK 以外のステータスの場合、以下を試してください。

- 待機してから describe-container-instances を実行します。
- EC2 コンソールまたは CLI を使用して、コンテナインスタンスの状態を表示します。
- CloudWatch メトリクスを確認します。詳細については、CloudWatch を使用して Amazon ECS をモニタリングするを参照してください。
- AWS Health Dashboard をチェックして、サービスに問題があるかどうかを確認してください。

アプリケーショントレースデータを使用して Amazon ECS 最適化の機会を特定する

Amazon ECS は OpenTelemetry 用 AWS Distro と統合して、アプリケーションからトレースデータを収集します。Amazon ECS は OpenTelemetry サイドカーコンテナ用の AWS Distroを使用して、トレースデータを収集し、AWS X-Ray にルーティングします。詳細については、「Amazon ECS の

OpenTelemetry Collector 用 AWS Distro の設定」を参照してください。そして、AWS X-Ray を使用してエラーや例外の特定、パフォーマンスのボトルネックや応答時間の分析を行うことができます。

OpenTelemetry Collector 用 AWS Distro がトレースデータを AWS X-Ray に送信するため、アプリケーションがトレースデータを作成できるように設定する必要があります。詳細については、「AWS X-Rayデベロッパーガイド」の「AWS X-Ray用にアプリケーションを計測する」を参照してください。

AWS X-Ray を使用した AWS Distro for OpenTelemetry の統合に必要な IAM 権限

Amazon ECS と AWS Distro for OpenTelemetry の統合には、タスク用のロールを作成し、タスク定義でそのロールを指定する必要があります。コンテナログが CloudWatch Logs にルーティングされるように AWS Distro for OpenTelemetry サイドカーを設定することをお勧めします。

> **Important**
>
> AWS Distro for OpenTelemetry 統合を使用してアプリケーションメトリクスの収集も行う場合は、タスク用の IAM ロールに、その統合に必要な権限も含まれている必要があります。詳細については、「アプリケーションメトリクスを使用して Amazon ECS アプリケーションのパフォーマンスを相関させる」を参照してください。

ロールを作成した後に、以下のアクセス許可でポリシーを作成し、ロールにアタッチします。

- `logs:PutLogEvents`
- `logs:CreateLogGroup`
- `logs:CreateLogStream`
- `logs:DescribeLogStreams`
- `logs:DescribeLogGroups`
- `logs:PutRetentionPolicy`
- `xray:PutTraceSegments`
- `xray:PutTelemetryRecords`
- `xray:GetSamplingRules`
- `xray:GetSamplingTargets`
- `xray:GetSamplingStatisticSummaries`

- ssm:GetParameters

タスク定義で AWS X-Ray 統合向け OpenTelemetry サイドカー用 AWS Distro の指定

Amazon ECS コンソールでは、[トレース収集の使用] オプションを使用することにより、OpenTelemetry サイドカーコンテナ用の AWS Distro の作成が簡素化されます。詳細については、「コンソールを使用した Amazon ECS タスク定義の作成」を参照してください。

Amazon ECS コンソールを使用していない場合、OpenTelemetry サイドカーコンテナ用 AWS Distro をタスク定義に追加できます。次のタスク定義のスニペットは、AWS X-Ray 統合向けの OpenTelemetry サイドカー用 AWS Distro を追加するためのコンテナの定義を示します。

```
{
 "family": "otel-using-xray",
 "taskRoleArn": "arn:aws:iam::111122223333:role/AmazonECS_OpenTelemetryXrayRole",
 "executionRoleArn": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole",
 "containerDefinitions": [{
   "name": "aws-otel-emitter",
   "image": "application-image",
   "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
     "awslogs-create-group": "true",
     "awslogs-group": "/ecs/aws-otel-emitter",
     "awslogs-region": "us-east-1",
     "awslogs-stream-prefix": "ecs"
    }
   },
   "dependsOn": [{
    "containerName": "aws-otel-collector",
    "condition": "START"
   }]
  },
  {
   "name": "aws-otel-collector",
   "image": "public.ecr.aws/aws-observability/aws-otel-collector:v0.30.0",
   "essential": true,
   "command": [
    "--config=/etc/ecs/otel-instance-metrics-config.yaml"
   ],
```

```
    "logConfiguration": {
     "logDriver": "awslogs",
     "options": {
      "awslogs-create-group": "True",
      "awslogs-group": "/ecs/ecs-aws-otel-sidecar-collector",
      "awslogs-region": "us-east-1",
      "awslogs-stream-prefix": "ecs"
     }
    }
   }
  ],
  "networkMode": "awsvpc",
  "requiresCompatibilities": [
   "FARGATE"
  ],
  "cpu": "1024",
  "memory": "3072"
}
```

アプリケーションメトリクスを使用して Amazon ECS アプリケーションのパフォーマンスを相関させる

Fargate の Amazon ECS は、Fargate で実行されているアプリケーションからメトリクスを収集してそれを Amazon CloudWatch または Amazon Managed Service for Prometheus のいずれかにエクスポートすることをサポートしています。

収集したメタデータを使用すると、アプリケーションパフォーマンスデータを基礎となるインフラストラクチャデータと相関させることができるため、問題解決までの平均時間を短縮できます。

Amazon ECS は、OpenTelemetry サイドカーコンテナ用 AWS Distro を使用して、アプリケーションメトリクスを収集して宛先にルーティングします。Amazon ECS コンソールエクスペリエンスでは、タスク定義の作成時にこの統合を追加するプロセスが簡素化されます。

トピック

- アプリケーションメトリクスを Amazon CloudWatch にエクスポートする
- アプリケーションメトリクスを Amazon Managed Service for Prometheus にエクスポートする

アプリケーションメトリクスを Amazon CloudWatch にエクスポートする

Fargate の Amazon ECS は、カスタムメトリクスとして、Amazon CloudWatch にカスタムアプリケーションメトリクスをエクスポートすることをサポートしています。これは、OpenTelemetry サイドカーコンテナ用 AWS Distro をタスク定義に追加することによって実行します。Amazon ECS コンソールでは、新しいタスク定義の作成時に [メトリクス収集の使用] オプションを追加することでこのプロセスが簡素化されます。詳細については、「コンソールを使用した Amazon ECS タスク定義の作成」を参照してください。

アプリケーションメトリクスは、ロググループ名 `/aws/ecs/application/metrics` を使用して CloudWatch Logs にエクスポートされ、メトリクスは `ECS/AWSOTel/Application` ネームスペースで表示できます。アプリケーションは、OpenTelemetry SDK を使用して計測する必要があります。詳細については、「AWS Distro for OpenTelemetry ドキュメント」の「AWS Distro for OpenTelemetry の概要」を参照してください。

考慮事項

OpenTelemetry 用 AWS Distro と Fargate 統合に Amazon ECS を使用して Amazon CloudWatch にアプリケーションメトリクスを送信する場合、次のことを考慮する必要があります。

- この統合では、カスタムアプリケーションメトリクスのみを CloudWatch に送信します。タスクレベルのメトリクスが必要な場合は、Amazon ECS クラスター設定で Container Insights を有効にします。詳細については、「オブザーバビリティが強化された Container Insights を使用し、Amazon ECS コンテナを監視する」を参照してください。
- AWS Distro for OpenTelemetry の統合は、Fargate でホストされている Amazon ECS ワークロード、および、Amazon EC2 インスタンスでホストされている Amazon ECS ワークロード用としてサポートされています。現在、外部インスタンスはサポートされていません。
- CloudWatch は、メトリクスごとに最大 30 個のディメンションをサポートします。デフォルトの Amazon ECS は `TaskARN`、`ClusterARN`、`LaunchType`、`TaskDefinitionFamily`、`TaskDefinitionRevision` ディメンションをメトリクスに含めることが既定となります。残りの 25 個のディメンションは、アプリケーションで定義できます。30 個以上のディメンションが設定されている場合、CloudWatch はそれらを表示できません。この場合、アプリケーションメトリクスは `ECS/AWSOTel/Application` CloudWatch のメトリクス名前空間に表示されますが、ディメンションは表示されません。アプリケーションを計測して、さらにディメンションを追加できます。詳細については、「AWS Distro for OpenTelemetry ドキュメント」の「AWS Distro for OpenTelemetry で CloudWatch のメトリクスを使用する」を参照してください。

OpenTelemetry 用 AWS Distro と Amazon CloudWatch の統合に必要な IAM 許可

Amazon ECS と OpenTelemetry 用 AWS Distro の統合には、タスク用 IAM ロールを作成し、タスク定義でロールを指定する必要があります。OpenTelemetry サイドカー用 AWS Distro サイドカーを、CloudWatch ログに対するルートコンテナログにも構成することをお勧めします。これには、タスク実行 IAM ロールを作成して、タスク定義でも指定する必要があります。Amazon ECS コンソールは、ユーザーに代わってタスク実行 IAM ロールを処理しますが、タスク IAM ロールは手動で作成してタスク定義に追加する必要があります。タスクの実行 IAM ロールの詳細については、「Amazon ECS タスク実行IAM ロール」を参照してください。

> **Important**
>
> OpenTelemetry 用 AWS Distro 統合を使用してアプリケーションのトレースデータも収集する場合、タスク IAM ロールもその統合に必要な許可が含まれていることを確認します。詳細については、「アプリケーショントレースデータを使用して Amazon ECS 最適化の機会を特定する」を参照してください。
> アプリケーションに権限が追加で必要な場合は、それらの権限をこのポリシーに追加する必要があります。各タスク定義では、タスク用の IAM ロールを 1 個のみ指定できます。例えば、Systems Manager に保存されているカスタム設定ファイルを使用する場合、この IAM ポリシーに `ssm:GetParameters` 権限を追加する必要があります。

Elastic Container Service のサービスロールを作成するには (IAM コンソール)

1. AWS マネジメントコンソール にサインインして、IAM コンソール https://console.aws.amazon.com/iam/ を開きます。
2. IAM コンソールのナビゲーションペインで、[ロール]、[ロールを作成] を選択します。
3. 信頼できるエンティティタイプ で、AWS のサービス を選択します。
4. [サービスまたはユースケース] で [エラスティックコンテナサービス] を選択し、次に [エラスティックコンテナのサービスタスク] を選択します。
5. [次へ] を選択します。
6. [アクセス許可を追加] セクションで [AWSDistroOpenTelemetryPolicyForXray] を検索し、そのポリシーを選択します。
7. (オプション) アクセス許可の境界を設定します。このアドバンスド機能は、サービスロールで使用できますが、サービスにリンクされたロールではありません。

a. [アクセス許可の境界の設定] セクションを開き、[アクセス許可の境界を使用してロールのアクセス許可の上限を設定する] を選択します。

   IAM には、アカウント内の AWS 管理ポリシーとカスタマー管理ポリシーのリストがあります。

b. アクセス許可の境界として使用するポリシーを選択します。

8. [次へ] を選択します。
9. このロールの目的を識別しやすいロール名またはロール名サフィックスを入力します。

> **Important**
>
> ロールに名前を付けるときは、次のことに注意してください。
>
> - ロール名は AWS アカウント内で一意である必要があります。ただし、大文字と小文字は区別されません。
>
>   例えば、**PRODROLE** と **prodrole** の両方の名前でロールを作成することはできません。ロール名がポリシーまたは ARN の一部として使用される場合、ロール名は大文字と小文字が区別されます。ただし、サインインプロセスなど、コンソールにロール名がユーザーに表示される場合、ロール名は大文字と小文字が区別されません。
> - 他のエンティティがロールを参照する可能性があるため、ロールを作成した後にロール名を編集することはできません。

10. (オプション) [説明] にロールの説明を入力します。
11. (オプション) ロールのユースケースとアクセス許可を編集するには、[ステップ 1: 信頼されたエンティティを選択] または [ステップ 2: アクセス権限を追加] のセクションで [編集] を選択します。
12. (オプション) ロールの識別、整理、検索を簡単にするには、キーと値のペアとしてタグを追加します。IAM でのタグの使用の詳細については、「IAM ユーザーガイド」の「AWS Identity and Access Management リソースのタグ」を参照してください。
13. ロールを確認したら、[ロールを作成] を選択します。

タスク定義の OpenTelemetry サイドカー用 AWS Distro を指定

Amazon ECS コンソールでは、[メトリクス収集の使用] オプションを使用することで、OpenTelemetry サイドカーコンテナ用の AWS Distro を作成するエクスペリエンスが簡素化され

ます。詳細については、「コンソールを使用した Amazon ECS タスク定義の作成」を参照してください。

Amazon ECS コンソールを使用していない場合、OpenTelemetry サイドカーコンテナ用 AWS Distro を手動でタスク定義に追加できます。次のタスク定義の例は、Amazon CloudWatch 統合に OpenTelemetry サイドカー用 AWS Distro を追加するためのコンテナの定義を示しています。

```
{
 "family": "otel-using-cloudwatch",
 "taskRoleArn": "arn:aws:iam::111122223333:role/AmazonECS_OpenTelemetryCloudWatchRole",
 "executionRoleArn": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole",
 "containerDefinitions": [
    {
  "name": "aws-otel-emitter",
  "image": "application-image",
  "logConfiguration": {
   "logDriver": "awslogs",
   "options": {
    "awslogs-create-group": "true",
    "awslogs-group": "/ecs/aws-otel-emitter",
    "awslogs-region": "us-east-1",
    "awslogs-stream-prefix": "ecs"
   }
  },
  "dependsOn": [{
   "containerName": "aws-otel-collector",
   "condition": "START"
  }]
 },
 {
  "name": "aws-otel-collector",
  "image": "public.ecr.aws/aws-observability/aws-otel-collector:v0.30.0",
  "essential": true,
  "command": [
   "--config=/etc/ecs/ecs-cloudwatch.yaml"
  ],
  "logConfiguration": {
   "logDriver": "awslogs",
   "options": {
    "awslogs-create-group": "True",
    "awslogs-group": "/ecs/ecs-aws-otel-sidecar-collector",
    "awslogs-region": "us-east-1",
    "awslogs-stream-prefix": "ecs"
```

```
    }
   }
  }
 ],
 "networkMode": "awsvpc",
 "requiresCompatibilities": [
  "FARGATE"
 ],
 "cpu": "1024",
 "memory": "3072"
}
```

アプリケーションメトリクスを Amazon Managed Service for Prometheus にエクスポートする

Amazon ECS は、タスクレベル CPU、メモリ、ネットワーク、ストレージメトリクス、カスタムアプリケーションメトリクスを Amazon Managed Service for Prometheus へエクスポートすることをサポートしています。これは、OpenTelemetry サイドカーコンテナ用 AWS Distro をタスク定義に追加することによって実行します。Amazon ECS コンソールでは、新しいタスク定義の作成時に [メトリクス収集の使用] オプションを追加することでこのプロセスが簡素化されます。詳細については、「コンソールを使用した Amazon ECS タスク定義の作成」を参照してください。

メトリクスは Amazon Managed Service for Prometheus にエクスポートされ、Amazon Managed Grafana ダッシュボードを使用して閲覧できます。アプリケーションは、Prometheus ライブラリまたは OpenTelemetry SDK のいずれかとインストルメント化する必要があります。OpenTelemetry SDK を使用したアプリケーションの計測の詳細については、AWS Distro for OpenTelemetry ドキュメントの「AWS Distro for OpenTelemetry の概要」を参照してください。

Prometheus ライブラリを使用する場合は、アプリケーションがメトリクスデータのスクレイプに使用する `/metrics` のエンドポイントを公開する必要があります。Prometheus ライブラリを使用してアプリケーションを計測する詳細については、「Prometheus ドキュメント」の「Prometheus クライアントライブラリ」を参照してください。

考慮事項

AWS Distro for OpenTelemetry での Fargate 統合で Amazon ECS を使用し、 Amazon Managed Service for Prometheus にアプリケーションメトリクスを送信する場合には、次のことを考慮する必要があります。

- AWS Distro for OpenTelemetry の統合は、Fargate でホストされている Amazon ECS ワークロード、および、Amazon EC2 インスタンスでホストされている Amazon ECS ワークロード用としてサポートされています。現在、外部インスタンスはサポートされていません。
- デフォルトの OpenElementry 用 AWS Distro には、Amazon Managed Service for Prometheus へエクスポートするとき、アプリケーションメトリクス用にタスクレベルのディメンションがすべて含まれています。アプリケーションをインストルメント化して、ディメンションを追加することもできます。詳細については、「AWS Distro for OpenTelemetry ドキュメント」の「Amazon Managed Service for Prometheus のリモート書き込みの使用開始」を参照してください。

OpenTelemetry 用 AWS Distro と Amazon Managed Service for Prometheus の統合に必要な IAM 許可

OpenTelemetry サイドカー用 AWS Distro を使用して Amazon ECS と Amazon Managed Service for Prometheus の統合には、タスク IAM ロールを作成してタスク定義でロールを指定する必要があります。このタスク用の IAM ロールは、タスク定義を登録する前に、手動で作成する必要があります。タスクロールの作成に関する詳細については、「Amazon ECS タスクの IAM ロール」を参照してください。

OpenTelemetry サイドカー用 AWS Distro サイドカーを、CloudWatch ログに対するルートコンテナログにも構成することをお勧めします。これには、タスク実行 IAM ロールを作成して、タスク定義でも指定する必要があります。Amazon ECS コンソールは、ユーザーに代わってタスク実行 IAM ロールを処理しますが、タスク IAM ロールは手動で作成する必要があります。タスク実行用の IAM ロールを作成する方法の詳細については、「Amazon ECS タスク実行IAM ロール」を参照してください。

> **Important**
>
> OpenTelemetry 用 AWS Distro 統合を使用してアプリケーションのトレースデータも収集する場合、タスク IAM ロールもその統合に必要な許可が含まれていることを確認します。詳細については、「アプリケーショントレースデータを使用して Amazon ECS 最適化の機会を特定する」を参照してください。

AWS Distro for OpenTelemetry と Amazon Managed Service for Prometheus の統合には、次のアクセス許可が必要です。

- logs:PutLogEvents

- logs:CreateLogGroup
- logs:CreateLogStream
- logs:DescribeLogStreams
- logs:DescribeLogGroups
- cloudwatch:PutMetricData

タスク定義の OpenTelemetry サイドカー用 AWS Distro を指定

Amazon ECS コンソールでは、[メトリクス収集の使用] オプションを使用することで、OpenTelemetry サイドカーコンテナ用の AWS Distro を作成するエクスペリエンスが簡素化されます。詳細については、「コンソールを使用した Amazon ECS タスク定義の作成」を参照してください。

Amazon ECS コンソールを使用していない場合、OpenTelemetry サイドカーコンテナ用 AWS Distro を手動でタスク定義に追加できます。次のタスク定義の例は、Amazon Managed Service for Prometheus 統合向けに OpenTelemetry サイドカー用 AWS Distro を追加するためのコンテナの定義を示しています。

```
{
 "family": "otel-using-cloudwatch",
 "taskRoleArn": "arn:aws:iam::111122223333:role/AmazonECS_OpenTelemetryCloudWatchRole",
 "executionRoleArn": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole",
 "containerDefinitions": [{
   "name": "aws-otel-emitter",
   "image": "application-image",
   "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
     "awslogs-create-group": "true",
     "awslogs-group": "/ecs/aws-otel-emitter",
     "awslogs-region": "aws-region",
     "awslogs-stream-prefix": "ecs"
    }
   },
   "dependsOn": [{
    "containerName": "aws-otel-collector",
    "condition": "START"
   }]
  },
```

```
  {
   "name": "aws-otel-collector",
   "image": "public.ecr.aws/aws-observability/aws-otel-collector:v0.30.0",
   "essential": true,
   "command": [
    "--config=/etc/ecs/ecs-amp.yaml"
   ],
   "environment": [{
    "name": "AWS_PROMETHEUS_ENDPOINT",
    "value": "https://aps-workspaces.aws-region.amazonaws.com/workspaces/
ws-a1b2c3d4-5678-90ab-cdef-EXAMPLE11111/api/v1/remote_write"
   }],
   "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
     "awslogs-create-group": "True",
     "awslogs-group": "/ecs/ecs-aws-otel-sidecar-collector",
     "awslogs-region": "aws-region",
     "awslogs-stream-prefix": "ecs"
    }
   }
  }
 ],
 "networkMode": "awsvpc",
 "requiresCompatibilities": [
  "FARGATE"
 ],
 "cpu": "1024",
 "memory": "3072"
}
```

AWS CloudTrail を使用して Amazon ECS API コールをログに記録する

Amazon Elastic Container Service は、ユーザー、ロール、または AWS のサービスによって実行されたアクションの記録を提供するサービスである AWS CloudTrail と統合されています。CloudTrail は、Amazon ECS へのすべての API コールをイベントとしてキャプチャします。キャプチャされる呼び出しには、Amazon ECS コンソールからの呼び出しと、Amazon ECS API オペレーションへのコード呼び出しが含まれます。CloudTrail で収集された情報を使用して、Amazon ECS に対するリクエスト、リクエスト元の IP アドレス、リクエストの作成日時、その他の詳細を確認できます。

各イベントまたはログエントリには、リクエストの生成者に関する情報が含まれます。アイデンティティ情報は、以下を判別するのに役立ちます。

- ルートユーザーまたはユーザー認証情報のどちらを使用してリクエストが送信されたか。
- リクエストが IAM Identity Center ユーザーに代わって行われたかどうか。
- リクエストがロールまたはフェデレーションユーザーのテンポラリなセキュリティ認証情報を使用して行われたかどうか。
- リクエストが、別の AWS のサービス によって送信されたかどうか。

アカウントを作成すると、AWS アカウント で CloudTrail がアクティブになり、自動的に CloudTrail の[イベント履歴] にアクセスできるようになります。CloudTrail の [イベント履歴] では、AWS リージョン で過去 90 日間に記録された管理イベントの表示、検索、およびダウンロードが可能で、変更不可能な記録を確認できます。詳細については、「AWS CloudTrail ユーザーガイド」の「CloudTrail イベント履歴の使用」を参照してください。[イベント履歴] の閲覧には CloudTrail の料金はかかりません。

AWS アカウント で過去 90 日間のイベントを継続的に記録するには、証跡または CloudTrail Lake イベントデータストアを作成します。

CloudTrail 証跡

証跡により、CloudTrail はログファイルを Amazon S3 バケットに配信できます。AWS マネジメントコンソール を使用して作成した証跡はマルチリージョンです。AWS CLI を使用する際は、単一リージョンまたは複数リージョンの証跡を作成できます。アカウント内のすべて AWS リージョン でアクティビティを把握するため、マルチリージョン証跡を作成することをお勧めします。単一リージョンの証跡を作成する場合、証跡の AWS リージョン に記録されたイベントのみを表示できます。証跡の詳細については、「AWS CloudTrail ユーザーガイド」の「AWS アカウントの証跡の作成」および「組織の証跡の作成」を参照してください。

証跡を作成すると、進行中の管理イベントのコピーを 1 つ無料で CloudTrail から Amazon S3 バケットに配信できますが、Amazon S3 ストレージには料金がかかります。CloudTrail の料金の詳細については、「AWS CloudTrail の料金」を参照してください。Amazon S3 の料金に関する詳細については、「Amazon S3 の料金」を参照してください。

CloudTrail Lake イベントデータストア

[CloudTrail Lake] を使用すると、イベントに対して SQL ベースのクエリを実行できます。CloudTrail Lake は、行ベースの JSON 形式の既存のイベントを Apache ORC 形式に変換し

ます。ORC は、データを高速に取得するために最適化された単票ストレージ形式です。イベントは、イベントデータストアに集約されます。イベントデータストアは、高度なイベントセレクタを適用することによって選択する条件に基づいた、イベントのイミュータブルなコレクションです。どのイベントが存続し、クエリに使用できるかは、イベントデータストアに適用するセレクタが制御します。CloudTrail Lake の詳細については、「AWS CloudTrail ユーザーガイド」の「AWS CloudTrail Lake の使用」を参照してください。

CloudTrail Lake のイベントデータストアとクエリにはコストがかかります。イベントデータストアを作成する際に、イベントデータストアに使用する料金オプションを選択します。料金オプションによって、イベントの取り込みと保存にかかる料金、および、そのイベントデータストアのデフォルトと最長の保持期間が決まります。CloudTrail の料金の詳細については、「AWS CloudTrail の料金」を参照してください。

CloudTrail の Amazon ECS 管理イベント

管理イベントでは、AWS アカウント のリソースに対して実行される管理オペレーションについての情報が得られます。これらのイベントは、コントロールプレーンオペレーションとも呼ばれます。CloudTrail は、デフォルトで管理イベントをログ記録します。

Amazon Elastic Container Service は、すべての Amazon ECS コントロールプレーンオペレーションを管理イベントとしてログに記録します。例えば、`CreateService`、`RunTask`、および `DeleteCluster` セクションを呼び出すと、CloudTrail ログファイルにエントリが生成されます。Amazon ECS が CloudTrail に記録する Amazon Elastic Container Service コントロールプレーンオペレーションのリストについては、「Amazon Elastic Container Service API リファレンス」を参照してください。

CloudTrail の Amazon ECS データイベント

データイベントでは、リソース上またはリソース内で実行されたリソースオペレーションについての情報が得られます。デフォルトでは、CloudTrail はデータイベントをログ記録しません。CloudTrail [イベント履歴] にはデータイベントは記録されません。

追加の変更がイベントデータに適用されます。CloudTrail の料金の詳細については、「AWS CloudTrail の料金」を参照してください。

CloudTrail コンソール、AWS CLI、または CloudTrail API オペレーションを使用して、Amazon ECS リソースタイプのデータイベントをログ記録できます。データイベントをログに記録する方法の詳細については、「AWS CloudTrail ユーザーガイド」の「AWS マネジメントコンソール を使用した

データイベントのログ記録」および「AWS Command Line Interface を使用したデータイベントのログ記録」を参照してください。

次の表に、データイベントをログに記録できる Amazon ECS リソースタイプを示します。データイベントタイプ (コンソール) 列には、CloudTrail コンソールの[データイベントタイプ]リストから選択する値が表示されます。resources.type 値列には、AWS CLI または CloudTrail API を使用して高度なイベントセレクタを設定するときに指定する `resources.type` 値が表示されます。CloudTrail に記録されたデータ API 列には、リソースタイプの CloudTrail にログ記録された API コールが表示されます。

データイベントタイプ (コンソール)	resources.type 値	CloudTrail にログ記録されたデータ API
AwsApiCall	`AWS::ECS::ContainerInstance`	• `ecs:Poll` (EC2 および Amazon ECS マネージドインスタンスの場合) • `ecs:StartTelemetrySession` (EC2 および Amazon ECS マネージドインスタンスの場合) • `ecs:PutSystemLogEvents` (Amazon ECS マネージドインスタンスの場合)

`eventName`、`readOnly`、および `resources.ARN` フィールドでフィルタリングして、自分にとって重要なイベントのみをログに記録するように高度なイベントセレクタを設定できます。オブジェクトの詳細については、「AWS CloudTrail API リファレンス」の「AdvancedFieldSelector」を参照してください。

Amazon ECS イベントの例

各イベントは任意の送信元からの単一のリクエストを表し、リクエストされた API オペレーション、オペレーションの日時、リクエストパラメータなどに関する情報を含みます。CloudTrail ログファイルは、パブリック API コールの順序付けられたスタックトレースではないため、イベントは特定の順序で表示されません。

以下の例は、CreateService アクションを示す CloudTrail イベントエントリです。

> **Note**
>
> この例は、読みやすくするために書式設定しています。CloudTrail ログファイルでは、すべてのエントリとイベントが単一の行に連結されています。さらに、この例は 1 つの Amazon ECS エントリに限定しています。実際の CloudTrail ログファイルには、複数の AWS のサービスからのエントリとイベントが表示されます。


```
{
    "eventVersion": "1.04",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE:account_name",
        "arn": "arn:aws:sts::123456789012:user/Mary_Major",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-06-20T18:32:25Z"
            },
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AIDACKCEVSQ6C2EXAMPLE",
                "arn": "arn:aws:iam::123456789012:role/Admin",
                "accountId": "123456789012",
                "userName": "Mary_Major"
            }
        }
    },
    "eventTime": "2018-06-20T19:04:36Z",
    "eventSource": "ecs.amazonaws.com",
    "eventName": "CreateCluster",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "console.amazonaws.com",
    "requestParameters": {
        "clusterName": "default"
    },
    "responseElements": {
```

```
        "cluster": {
            "clusterArn": "arn:aws:ecs:us-east-1:123456789012:cluster/default",
            "pendingTasksCount": 0,
            "registeredContainerInstancesCount": 0,
            "status": "ACTIVE",
            "runningTasksCount": 0,
            "statistics": [],
            "clusterName": "default",
            "activeServicesCount": 0
        }
    },
    "requestID": "cb8c167e-EXAMPLE",
    "eventID": "e3c6f4ce-EXAMPLE",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

CloudTrail レコードの内容については、「AWS CloudTrail ユーザーガイド」の「CloudTrail record contents」を参照してください。

Amazon ECS サービスとタスクに関する CloudWatch Logs Live Tail の表示

CloudWatch Logs Live Tail を使用すると、新しいログイベントが取り込まれたときにストリーミングリストを表示して、インシデントのトラブルシューティングをすばやく行うことができます。Amazon ECS コンソールでタスクイベントとサービスイベントを表示できます。このビューは、タスクとサービスの状態を一元的に表示します。

Amazon ECS サービスの各タスクには、コンテナごとに専用のログストリームがあります。サービスがスケールアップした場合、各タスクインスタンスには独自のログストリームのセットがあります。ログストリームの命名規則は、次のパターンに従います。

```
log-group-prefix/container-name/task-id
```

単一のタスクが、その存続期間中に同じログストリームに書き込みます。ログストリームには、そのタスクのコンテナからのメッセージと、アプリケーションコードからの出力も含まれます。カスタムログを含め、すべてのメッセージにはタイムスタンプが付けられます。

> **Note**
>
> Live Tail セッションでは、セッションの使用時間ごとに 1 分間隔でコストが発生します。料金の詳細については、「Amazon CloudWatch の料金」を参照してください。

必要なアクセス許可

コンソール IAM ユーザーが CloudWatch Logs Live Tail セッションを開始および停止するには、以下のアクセス許可が必要です:

- `logs:DescribeLogGroups`
- `logs:StartLiveTail`
- `logs:StopLiveTail`

手順

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
2. 表示するリソースを決定する

リソース	Steps
タスク	a. [クラスター] ページで、クラスターを選択します。 クラスター詳細のページが表示されます。 b. [タスク] タブを選択します。 [タスクの詳細] ページが表示されます。 c. [ログ] タブを選択します。
サービス	a. [クラスター] ページで、クラスターを選択します。

リソース	Steps
	クラスター詳細のページが表示されます。 b. [サービス] を選択します。 [サービスの詳細] ページが表示されます。 c. [ログ] タブを選択します。

3. [CloudWatch Logs Live Tail] を選択し、[開始] を選択します。
4. (オプション) ストリームをフィルタリングするには、[フィルター] の[ロググループを選択] で、ロググループを選択します。

AWS CLI のリファレンス

AWS CLI を使用して CloudWatch Logs の Live Tail セッションを開始することもできます。以下のリソースは、Live Tail での AWS CLI の使用に関する詳細情報を提供します。

- start-live-tail コマンドリファレンス – 完全なコマンド構文、パラメータ、および `aws logs start-live-tail` コマンドの例。
- CloudWatch Logs Live Tail ユーザーガイド – 印刷専用モードとインタラクティブモードの両方での AWS CLI の使用法を含む包括的なガイド。
- StartLiveTail SDK の例 – StartLiveTail API をさまざまな AWSSDK で使用するためのプログラム例。

アクションログを使用して Amazon ECS オペレーションをモニタリングする

Amazon ECS アクションログの概要

Amazon ECS アクションログを使用すると、コンテナライフサイクルオペレーション中に Amazon ECS がユーザーに代わって実行するすべてのアクションをモニタリングできます。アクションログは、クラスター内のすべてのリソースオペレーション、状態遷移、およびサービス開始 API コールのタイムスタンプ付きレコードをキャプチャします。

アクションログは、コンテナイメージのダウンロード、ロードバランサーの登録、セキュリティグループの設定など、以前は表示されなかったオペレーションを記録します。アクションログがない場合、リソースの開始状態と終了状態のみを確認できます。アクションログを使用すると、Amazon ECS がそれらの状態間で実行するすべての中間手順を確認できます。

アクションログを有効にすると、AWS マネジメントコンソールの Amazon Q はこれらのログにアクセスして、デプロイの失敗とデーモンの問題をコンソールから直接デバッグできます。

アクションログの利点

アクションログには、Amazon ECS ワークロードを管理するための以下の利点があります。

- サービスアクションの可視性 – インフラストラクチャのプロビジョニング、ネットワーク設定、リソース登録など、状態遷移間で Amazon ECS が実行する以前は表示されなかったアクションを確認できます。
- トラブルシューティングの向上 – AWS サポートに連絡せずに、デプロイの失敗とインフラストラクチャの問題を診断できます。アクションログは、増加し続ける一連の失敗理由をサポートします。
- 拡張された失敗メタデータ – 標準の 1 時間の保持期間を超えて、失敗したタスクに関するメタデータを保持します。詳細な障害情報は、自分のペースで確認できます。
- セルフサービスデバッグ – 障害の原因となった一連のアクションを確認して、問題を個別に調査・解決します。

ログで記録される内容

アクションログは、次の Amazon ECS オペレーションのアクションをキャプチャします。

- サービスデプロイ – 状態遷移、ロールバック、ライフサイクルフックの実行。
- マネージドデーモンライフサイクル – マネージドデーモンの作成、更新、削除、インスタンスドレインオペレーション。

アクションログの仕組み

アクションログはクラスターレベルでオプトインします。アクションログを有効にするには、Amazon ECS コンソールを使用するか、Amazon CloudWatch API (`PutDeliverySource`、`PutDeliveryDestination`、および `CreateDelivery`) を使用します。

アクションログを有効にすると、Amazon ECS は CloudWatch Ingestion Hub を介して構造化 JSON ログを出力します。Amazon ECS は、CloudWatch Logs、Amazon S3、または Amazon Data Firehose という選択した送信先にログを配信します。

アクションログは Fire-and-Forget 方式の発行を使用します。ログ発行によって Amazon ECS リソースオペレーションがブロックまたは低下することはありません。一時的な配信障害が発生した場合、Amazon ECS は中断することなくワークロードの処理を続行します。

ログスキーマの概要

各アクションログエントリには、次のフィールドが含まれます。

- `timestamp` – イベントが発生した時刻の Unix タイムスタンプ (ミリ秒単位)。
- `logLevel` – 重要度レベル: `INFO`、`WARN`、または `ERROR`。
- `account` – AWS アカウント ID。
- `region` – イベントが発生した AWS リージョン。
- `resourceArn` – オプトインスコープを表すクラスター ARN。
- `actionSourceId` – サービス ARN やデーモン ARN などのサブリソース識別子。
- `eventName` – アクション識別子。例: `DAEMON_DEPLOYMENT_IN_PROGRESS`。
- `detail` – アクションに関する追加のコンテキストを含むイベント固有の JSON ペイロード。

ログの送信先と整理

アクションログは、次の送信先に配信できます。

- CloudWatch Logs – CloudWatch Logs は、`/aws/vendedlogs/ecs/action-logs/{resourceId}` という名前のロググループにログを保存します。Amazon ECS は、`{actionSourceId}` をログストリーム名として使用して、サブリソースごとに個別のログストリームを作成します。
- Amazon S3 – Amazon S3 は、アカウント、リージョン、および 5 分間の時間枠でログを分割します。Amazon S3 は、効率的なストレージのために gzip を使用してファイルを圧縮します。
- Amazon Data Firehose – Amazon Data Firehose は、ログを改行区切りの JSON レコードとしてほぼリアルタイムで配信します。

料金

アクションログは、CloudWatch が提供する標準のログ料金を使用する有料機能です。ログの取り込みとストレージには、クラスターが生成するログの量に基づいて料金が発生します。保持期間は送信先の設定で構成できます。Amazon ECS コンソールでアクションログを有効にすると、CloudWatch Logs はデフォルトの保持期間である 7 日間を使用します。

トピック

- Amazon ECS アクションログの導入
- Amazon ECS アクションログを使用したトラブルシューティング

Amazon ECS アクションログの導入

Amazon ECS アクションログは、Amazon CloudWatch Logs が提供するログ配信メカニズムを使用して、サービスアクションの詳細なレコードを任意の送信先に発行します。アクションログを有効にすると、インフラストラクチャのプロビジョニング、リソース登録、障害の詳細など、Amazon ECS がデプロイおよびデーモンオペレーション中に実行するすべての手順を確認できます。この可視性により、AWS サポートに連絡せずに問題をトラブルシューティングし、根本原因の特定にかかる時間を短縮できます。Amazon ECS コンソールまたは CloudWatch Logs APIs を使用して、クラスターのアクションログを有効にできます。

前提条件

アクションログを有効にする前に、IAM ID に必要なアクセス許可があることを確認してください。

必要な IAM 許可

アクションログを有効化および管理するには、IAM ID に次のアクセス許可が必要です。

```
{
    "Effect": "Allow",
    "Action": [
        "logs:PutDeliverySource",
        "logs:PutDeliveryDestination",
        "logs:CreateDelivery",
        "logs:GetDelivery",
        "ecs:AllowVendedLogDeliveryForResource"
    ],
    "Resource": "*"
```

```
}
```

送信先リソースポリシー

送信先の CloudWatch Logs ロググループには、`delivery.logs.amazonaws.com` サービスがログを書き込むことを許可するリソースポリシーが必要です。ロググループリソースポリシーに次のステートメントを追加します。

```
{
    "Effect": "Allow",
    "Principal": {
        "Service": "delivery.logs.amazonaws.com"
    },
    "Action": [
        "logs:CreateLogStream",
        "logs:PutLogEvents"
    ],
    "Resource": "arn:aws:logs:region:account-id:log-group:log-group-name:log-stream:*"
}
```

Amazon ECS コンソールを使用してアクションログを有効にする

アクションログは、Amazon ECS コンソールのクラスターの詳細ページから直接有効にできます。

コンソールを使用してアクションログを有効にするには

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
2. ナビゲーションペインで、[クラスター] を選択し、次にクラスターを選択します。
3. [設定] タブを選択します。
4. [アクションログ] セクションで、[追加] を選択します。
5. [Destination] で、以下のいずれかのオプションを選択します。

   - [CloudWatch Logs] (デフォルト)
   - Amazon S3
   - Amazon Data Firehose

   CloudWatch Logs の場合、Amazon ECS は 7 日間の保持期間を持つ `/aws/vendedlogs/ecs/action-logs/cluster-name` という名前のデフォルトのロググループを作成します。
6. [保存] を選択します。

CloudWatch Logs API を使用してアクションログを有効にする

アクションログは、CloudWatch Logs が提供するログ配信 API を使用してプログラムで有効にできます。このアプローチには、配信ソースの登録、送信先の設定、配信の作成の 3 つの手順が必要です。

AWS CLI を使用してアクションログを有効にするには

1. 次のコマンドを実行して、配信ソースを登録します。

```
aws logs put-delivery-source \
    --name my-ecs-action-logs \
    --resource-arn arn:aws:ecs:region:account-id:cluster/cluster-name \
    --log-type EcsActionLogs
```

2. 次のコマンドを実行して、配信先を設定します。

```
aws logs put-delivery-destination \
    --name my-ecs-logs-destination \
    --output-format json \
    --delivery-destination-configuration '{"destinationResourceArn":
 "arn:aws:logs:region:account-id:log-group:/aws/vendedlogs/ecs/action-logs/cluster-
name"}'
```

3. 次のコマンドを実行して、送信元を送信先にリンクする配信を作成します。

```
aws logs create-delivery \
    --delivery-source-name my-ecs-action-logs \
    --delivery-destination-arn arn:aws:logs:region:account-id:delivery-
destination:my-ecs-logs-destination
```

ログ配信を確認する

アクションログを有効にしたら、ログが送信先に配信されていることを確認します。

ログ配信を確認するには

1. クラスターでアクションをトリガーします。たとえば、デーモンサービスを作成または更新します。
2. CloudWatch コンソールの https://console.aws.amazon.com/cloudwatch/ を開いてください。

3. ナビゲーションペインで、[Log groups] (ロググループ) を選択します。
4. アクションログロググループを選択し、新しいログストリームが表示されることを確認します。

アクションログの削除

アクションログは、AWS CLI または Amazon ECS コンソールを使用して削除できます。

アクションログを削除するには、IAM ID に次の追加のアクセス許可が必要です。

```
{
    "Effect": "Allow",
    "Action": [
        "logs:DeleteDelivery",
        "logs:DeleteDeliverySource",
        "logs:DeleteDeliveryDestination"
    ],
    "Resource": "*"
}
```

AWS CLI を使用してアクションログを削除するには

1. 次のコマンドを実行して、配信を削除します。

```
aws logs delete-delivery \
    --delivery-id delivery-id
```

2. 次のコマンドを実行して、配信ソースを削除します。

```
aws logs delete-delivery-source \
    --name my-ecs-action-logs
```

3. 次のコマンドを実行して、配信先を削除します。

```
aws logs delete-delivery-destination \
    --name my-ecs-logs-destination
```

コンソールを使用してアクションログを削除するには、クラスターの詳細ページに移動し、[設定] タブを選択し、[アクションログ] セクションで [削除] を選択します。

次のステップ

アクションログを有効にしたら、次のリソースを使用して Amazon ECS クラスターをモニタリングおよびトラブルシューティングします。

- Amazon ECS アクションログを使用したトラブルシューティング

Amazon ECS アクションログを使用したトラブルシューティング

アクションログは、Amazon ECS のサービス開始オペレーションの詳細なレコードを提供することで、問題の診断に役立ちます。各ログエントリには、タイムスタンプ、ログレベル、イベント名、および発生した内容と理由に関するコンテキストを含む詳細ペイロードが含まれます。このトピックでは、一般的なトラブルシューティングシナリオについて説明し、すぐに使用できる CloudWatch Logs Insights クエリを提供します。

ログレベルを理解する

アクションログは、3 つのログレベルを使用して各イベントの重要度を示します。

`INFO`

デプロイの開始、タスクの起動、リソースのプロビジョニングなどの通常のオペレーション。これらのイベントは、Amazon ECS が期待どおりにアクションを実行していることを示します。

`WARN`

再試行や容量の制約など、対処が必要となる可能性のある、致命的ではない問題。Amazon ECS は順調に進行していますが、オペレーションに予想以上の時間がかかる場合があります。

`ERROR`

デプロイの失敗、プロビジョニングエラー、タスク開始の障害など、アクションが必要な障害。これらのイベントを確認して根本原因を特定し、是正措置を講じます。

> **Tip**
>
> まず、`ERROR` または `WARN` のログレベルでフィルタリングして、問題をすばやく特定します。その後、検索を `INFO` イベントに拡張して、追加のコンテキストを取得できます。

失敗したサービスデプロイのデバッグ

症状

デプロイが進行中のままスタックするか、自動的にロールバックされます。

検索対象

eventName に DEPLOYMENT が含まれている ERROR ログをフィルタリングします 。detail ペイロードでデプロイ ARN、ステータスの理由、失敗したタスク数を確認します。

次の例は、ロールバック候補が利用できなかった場合のデプロイの失敗を示しています。

```
{
  "resourceArn": "arn:aws:ecs:us-east-1:111122223333:cluster/my-cluster",
  "actionSourceId": "service/my-cluster/my-service",
  "logLevel": "ERROR",
  "eventTimestamp": 1784573730986,
  "detail": {
    "statusReason": "No rollback candidate was found to run the rollback.",
    "serviceDeploymentArn": "arn:aws:ecs:us-east-1:111122223333:service-deployment/my-
cluster/my-service/abc123def456",
    "status": "FAILED",
    "eventName": "SERVICE_DEPLOYMENT_ROLLBACK_FAILED"
  }
}
```

解決策

detail.statusReason フィールドを確認して、タスクが失敗した理由を特定します。次の CloudWatch Logs Insights クエリを使用して、すべてのデプロイエラーを検索します。

```
fields @timestamp, detail.statusReason
| filter logLevel = "ERROR" and detail.deploymentArn like /my-cluster\/my-service/
| sort @timestamp desc
| limit 20
```

マネージドデーモンの問題の診断

症状

デーモンがインスタンスで起動しないか、デーモンのデプロイがスタックしています。

検索対象

デーモン ARN でログをフィルタリングして、デーモンに関連するイベントを検索します。logLevel でフィルタリングして、結果を警告やエラーに絞り込むこともできます。

次の例は、ターゲットインスタンスのメモリ不足が原因で起動に失敗したデーモンタスクを示しています。

```
{
  "timestamp": 1719500050000,
  "logLevel": "WARN",
  "account": "123456789012",
  "region": "us-east-1",
  "resourceArn": "arn:aws:ecs:us-east-1:123456789012:cluster/my-cluster",
  "actionSourceId": "arn:aws:ecs:us-east-1:123456789012:daemon/my-cluster/my-logging-
daemon",
  "eventName": "DAEMON_TASK_START_IMPAIRED",
  "detail": {
    "statusReason": "RESOURCE:MEMORY - Unable to place daemon task on container
 instance: insufficient memory",
    "containerInstanceArn": "arn:aws:ecs:us-east-1:123456789012:container-instance/my-
cluster/a1b2c3d4"
  }
}
```

解決策

ターゲットインスタンスにデーモンタスクを実行するのに十分なリソースがあることを確認します。インスタンスがドレイン中の場合、新しいインスタンスが利用可能になるまでデーモンが起動しないことがあります。次のクエリを使用して、特定のデーモンのイベントをフィルタリングします。

```
fields @timestamp, logLevel, detail.statusReason
| filter actionSourceId like /my-logging-daemon/
| filter logLevel in ["ERROR", "WARN"]
| sort @timestamp desc
| limit 20
```

デーモンタスクの停止理由の確認

症状

デーモンタスクが予期せず停止する。

検索対象

デーモンタスクの失敗の詳細を含む WARN ログと ERROR ログをフィルタリングします。detail ペイロードの次のフィールドは、デーモンタスクが停止した理由に関する情報を提供します。

`detail.stopCode`

タスクが停止した理由を分類する機械可読コード。

`detail.statusReason`

タスクが停止した理由を人間が読み取れる形式で説明します。このフィールドは、すべてのログエントリに存在するわけではありません。

`detail.containerStoppedReason`

タスクを停止させた特定のコンテナに関する追加のコンテキスト。

タスク停止コードの完全なリストとその説明については、「Amazon ECS の停止したタスクのエラーメッセージ」を参照してください。

> **Note**
>
> UserInitiated などの予想されるデーモンタスクの停止は、アクションログから除外されます。予期しないデーモンタスクの停止またはエラー関連のデーモンタスクの停止のみが表示されます。

便利な CloudWatch Logs Insights クエリの例

次の CloudWatch Logs Insights クエリを使用して、Amazon ECS サービスの一般的な問題を調査できます。

過去 1 時間のすべてのエラー

```
fields @timestamp, logLevel, eventName, detail.statusReason
| filter logLevel = "ERROR"
| sort @timestamp desc
| limit 50
```

> **Note**
>
> `detail.statusReason` フィールドは、すべてのログエントリに存在するわけではありません。結果でフィールドが空の場合は、コンテキストとして `eventName` およびその他のフィールドを `detail` ペイロードで使用します。

特定のサービスのイベント

```
fields @timestamp, logLevel, detail.statusReason
| filter @message like /my-service/
| sort @timestamp desc
| limit 50
```

特定のデプロイのデプロイタイムライン

```
fields @timestamp, logLevel, detail.statusReason
| filter detail.deploymentArn like /my-cluster\/my-service\/abc123/
| sort @timestamp asc
```

理由別にグループ化されたデーモンタスクの失敗

```
filter logLevel = "ERROR" and detail.stopCode != ""
| stats count(*) as failureCount by detail.stopCode, detail.statusReason
| sort failureCount desc
| limit 20
```

Amazon ECS メタデータを使用したワークロードのモニタリング

タスクとコンテナのメタデータを使用して、ワークロードをトラブルシューティングしたり、ランタイム環境に基づいて設定を変更したりすることができます。

メタデータには以下のカテゴリが含まれます。

- タスクが実行されている場所に関する情報を提供するタスクレベルの属性。
- Docker ID、名前、およびイメージの詳細を提供するコンテナレベルの属性。

  これにより、コンテナを可視化できます。

- IP アドレス、サブネットおよびネットワークモードなどのネットワーク設定。

  これはネットワークの設定やトラブルシューティングに役立ちます。
- タスクのステータスと正常性

  これにより、タスクが実行されているかどうかを確認できます。

次のいずれかの方法で、メタデータを表示できます。

- コンテナメタデータファイル

  Amazon ECS コンテナエージェントのバージョン 1.15.0 以降では、コンテナまたはホストコンテナインスタンス内でさまざまなコンテナメタデータを使用できます。この機能を有効にすると、タスク、コンテナ、およびコンテナ内部のコンテナインスタンスまたはホストコンテナインスタンスに関する情報をクエリできます。メタデータファイルはホストインスタンスで作成され、Docker ボリュームとしてコンテナにマウントされるため、タスクが AWS Fargate でホストされているときは使用できません。
- タスクメタデータエンドポイント

  Amazon ECS コンテナエージェントは、各コンテナに環境変数を注入します。これは、タスクメタデータエンドポイントと呼ばれ、コンテナにさまざまなタスクメタデータと Docker 統計を提供します。
- コンテナの詳細分析

  Amazon ECS コンテナエージェントは、エージェントが実行しているコンテナインスタンスおよびインスタンスで実行されている関連タスクに関する詳細を収集する API オペレーションを提供します。

Amazon ECS 環境変数

環境変数は、タスクの実行に使用される容量など、タスク動作を制御します。Amazon ECS は、次の環境変数をタスクに設定します。

- `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` - SDK が認証情報をリクエストするときに使用する完全な HTTP URL エンドポイントが含まれます。これにはスキームとホストの両方が含まれます。詳細については、「AWS SDK リファレンスガイド」の「コンテナ認証情報プロバイダー」を参照してください。

- ECS_CONTAINER_METADATA_URI_V4 - タスクメタデータバージョン 4 のアドレス。詳細については、「Amazon ECS タスクメタデータエンドポイントバージョン 4」を参照してください。
- ECS_CONTAINER_METADATA_URI - タスクメタデータバージョン 3 のアドレス。詳細については、「Amazon ECS タスクメタデータエンドポイントバージョン 3」を参照してください。
- ECS_AGENT_URI - Fargate でサポートされているさまざまなエンドポイントの基本アドレス。詳細については、以下を参照してください。
  - Amazon ECS タスクスケールイン保護のエンドポイント
  - Amazon ECS フォールトインジェクションエンドポイント
- AWS_EXECUTION_ENV – タスクが実行されるコンピューティングオプションに関する情報。
  - Fargate の場合、Amazon ECS はこれを AWS_ECS_FARGATE に設定します。
  - EC2 の場合、Amazon ECS はこれを AWS_ECS_EC2 に設定します。
- AWS_DEFAULT_REGION – AWS リージョン のデフォルト AWS アカウント。これは、タスクが実行されるデフォルトのリージョンです。
- AWS_REGION - タスクが実行されるリージョン。定義されている場合、この値は AWS_DEFAULT_REGION を上書きします。

タスクメタデータで環境変数を表示できます。詳細については、次のトピックのいずれかを参照してください。

- Fargate - Fargate のタスクで使用できる Amazon ECS タスクメタデータ
- EC2 - EC2 のタスクで使用できる Amazon ECS タスクメタデータ

Amazon ECS コンテナメタデータファイル

Amazon ECS コンテナエージェントのバージョン 1.15.0 以降では、コンテナまたはホストコンテナインスタンス内でさまざまなコンテナメタデータを使用できます。この機能を有効にすると、タスク、コンテナ、およびコンテナ内部のコンテナインスタンスまたはホストコンテナインスタンスに関する情報をクエリできます。メタデータファイルはホストインスタンスで作成され、Docker ボリュームとしてコンテナにマウントされるため、タスクが AWS Fargate でホストされているときは使用できません。

コンテナメタデータファイルは、コンテナがクリーンアップされるときにホストインスタンスでクリーンアップされます。ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION コンテナエージェント変数により、クリーンアップを実行するタイミングを調整できます。詳細については、「Amazon ECS タスクとイメージの自動クリーンアップ」を参照してください。

トピック

- コンテナメタデータファイルの場所
- Amazon ECS コンテナメタデータをオンにする
- Amazon ECS コンテナメタデータファイル形式

コンテナメタデータファイルの場所

デフォルトでは、コンテナメタデータファイルは、次のホストパスとコンテナパスに書き込まれます。

- Linux インスタンスの場合:
  - ホストパス: `/var/lib/ecs/data/metadata/cluster_name/task_id/container_name/ecs-container-metadata.json`

    > **Note**
    >
    > Linux ホストパスでは、エージェントの起動時にデフォルトのデータディレクトリマウントパス (`/var/lib/ecs/data`) が使用されます。Amazon ECS に最適化された AMI を使用しない場合 (または、`ecs-init` パッケージを使用してコンテナエージェントを開始および維持しない場合) は、`ECS_HOST_DATA_DIR` エージェント設定変数を、コンテナエージェントの状態ファイルがあるホストパスに設定します。詳細については、「Amazon ECS コンテナエージェントの設定」を参照してください。

  - コンテナパス: `/opt/ecs/metadata/random_ID/ecs-container-metadata.json`
- Windows インスタンスの場合:
  - ホストパス: `C:\ProgramData\Amazon\ECS\data\metadata\task_id\container_name\ecs-container-metadata.json`
  - コンテナパス: `C:\ProgramData\Amazon\ECS\metadata\random_ID\ecs-container-metadata.json`

ただし、簡単にアクセスできるようにするため、コンテナメタデータファイルの場所は、コンテナ内の `ECS_CONTAINER_METADATA_FILE` 環境変数に設定されます。コンテナ内から、以下のコマンドを使用してファイルの内容を読み取ることができます。

- Linux インスタンスの場合:

```
cat $ECS_CONTAINER_METADATA_FILE
```

- Windows インスタンス (PowerShell) の場合:

```
Get-Content -path $env:ECS_CONTAINER_METADATA_FILE
```

Amazon ECS コンテナメタデータをオンにする

コンテナメタデータは、ECS_ENABLE_CONTAINER_METADATA コンテナエージェント変数を true に設定して、コンテナインスタンスレベルでオンにできます。/etc/ecs/ecs.config 設定ファイルでこの変数を設定し、エージェントを再起動できます。実行時 (エージェントコンテナが起動するとき) に、Docker 環境変数として設定することもできます。詳細については、「Amazon ECS コンテナエージェントの設定」を参照してください。

エージェントの起動時に ECS_ENABLE_CONTAINER_METADATA が true に設定されている場合、それ以降に作成されたのコンテナに対してメタデータファイルが作成されます。Amazon ECS コンテナエージェントは、ECS_ENABLE_CONTAINER_METADATA コンテナエージェント変数が true に設定される前に作成されたコンテナに対してメタデータファイルを作成することはできません。すべてのコンテナがメタデータファイルを受け取るようにするには、コンテナインスタンスの起動時に、このエージェント変数を設定します。以下は、この変数を設定し、コンテナインスタンスをクラスターに登録するユーザーデータスクリプトの例です。

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=your_cluster_name
ECS_ENABLE_CONTAINER_METADATA=true
EOF
```

Amazon ECS コンテナメタデータファイル形式

次の情報は、コンテナのメタデータ JSON ファイルに保存されます。

`Cluster`

コンテナのタスクが実行されているクラスターの名前。

`ContainerInstanceARN`

ホストコンテナインスタンスの完全な Amazon リソースネーム (ARN)。

`TaskARN`

コンテナが属しているタスクの完全な Amazon リソースネーム (ARN)。

`TaskDefinitionFamily`

コンテナで使用しているタスク定義ファミリーの名前。

`TaskDefinitionRevision`

コンテナが使用しているタスク定義リビジョン。

`ContainerID`

コンテナの Docker コンテナ ID (Amazon ECS コンテナの ID ではありません)。

`ContainerName`

コンテナ用の Amazon ECS タスク定義からのコンテナ名。

`DockerContainerName`

Docker デーモンがコンテナに使用するコンテナ名 (例えば、docker ps コマンドの出力に表示される名前)。

`ImageID`

コンテナの起動に使用される Docker イメージの SHA ダイジェスト。

`ImageName`

コンテナの起動に使用される Docker イメージのイメージ名とタグ。

`PortMappings`

コンテナに関連付けられているすべてのポートマッピング。

`ContainerPort`

公開されるコンテナのポート。

`HostPort`

公開されるホストコンテナインスタンスのポート。

`BindIp`

Docker によってコンテナに割り当てられたバインド IP アドレス。この IP アドレスは、`bridge` ネットワークモードでのみ適用され、コンテナインスタンスからのみアクセス可能です。

Protocol

ポートマッピングに使用されるネットワークプロトコル。

Networks

コンテナのネットワークモードと IP アドレス。

NetworkMode

コンテナが属するタスクのネットワークモード。

IPv4Addresses

コンテナに関連付けられた IP アドレス。

> **Important**
>
> タスクで awsvpc ネットワークモードを使用している場合、コンテナの IP アドレスは返されません。この場合は、次のコマンドを使用して /etc/hosts ファイルを読み取ることにより IP アドレスを取得できます。
>
> ```
> tail -1 /etc/hosts | awk '{print $1}'
> ```

MetadataFileStatus

メタデータファイルのステータス。ステータスが READY である場合、メタデータファイルは最新で完了済みです。ファイルがまだ準備できていない場合 (例えば、タスクが開始された時点)、切り捨てられたバージョンのファイル形式を使用できます。コンテナが開始されたが、メタデータがまだ書き込まれていないという競合状態が発生する可能性を回避するため、メタデータファイルを解析し、メタデータに基づいてこのパラメータが READY に設定されるのを待つことができます。これは通常、コンテナの開始から 1 秒未満で使用できます。

AvailabilityZone

ホストコンテナインスタンスが存在するアベイラビリティーゾーン。

HostPrivateIPv4Address

コンテナが属するタスクのプライベート IP アドレス。

HostPublicIPv4Address

コンテナが属するタスクのパブリック IP アドレス。

Example Amazon ECS コンテナメタデータファイル (**READY**)

次の例では、READY ステータスのコンテナメタデータファイルを示します。

```
{
    "Cluster":"arn:aws:ecs:us-east-1:123456789012:cluster/MyCluster",
    "TaskARN":"arn:aws:ecs:us-east-1:123456789012:task/MyCluster/
b593651c4d6b44a6b2b583f45c957e15",
    "Family":"curltest-container",
    "Revision":"2",
    "DesiredStatus":"RUNNING",
    "KnownStatus":"RUNNING",
    "Limits":
        {
            "CPU":0.25,
            "Memory":512
        },
    "PullStartedAt":"2025-01-17T20:56:17.394610044Z",
    "PullStoppedAt":"2025-01-17T20:56:25.282708213Z",
    "AvailabilityZone":"us-east-1b",
    "LaunchType":"FARGATE",
    "Containers":[
        {
            "DockerId":"b593651c4d6b44a6b2b583f45c957e15-3356213583",
            "Name":"curltest","DockerName":"curltest",
            "Image":"public.ecr.aws/amazonlinux/amazonlinux:latest",

 "ImageID":"sha256:7f371357694782356b65c7fd60dd1ca124c47bd5ed1b1ffe7c0e17f562898367",
            "Labels":
                {
                    "com.amazonaws.ecs.cluster":"arn:aws:ecs:us-
east-1:123456789012:cluster/MyCluster",
                    "com.amazonaws.ecs.container-name":"curltest",
                    "com.amazonaws.ecs.task-arn":"arn:aws:ecs:us-
east-1:123456789012:task/MyCluster/b593651c4d6b44a6b2b583f45c957e15",
                    "com.amazonaws.ecs.task-definition-family":"curltest-
container","com.amazonaws.ecs.task-definition-version":"2"
                },
            "DesiredStatus":"RUNNING",
            "KnownStatus":"RUNNING",
            "Limits":
                {
                    "CPU":2
                },
```

```
            "CreatedAt":"2025-01-17T20:56:26.180347056Z",
            "StartedAt":"2025-01-17T20:56:26.180347056Z",
            "Type":"NORMAL",
            "LogDriver":"awslogs",
            "LogOptions":
                {
                    "awslogs-create-group":"true",
                    "awslogs-group":"/ecs/curltest-container",
                    "awslogs-region":"us-east-1",
                    "awslogs-stream":"ecs/curltest/b593651c4d6b44a6b2b583f45c957e15"
                       },
            "ContainerARN":"arn:aws:ecs:us-east-1:123456789012:container/MyCluster/
b593651c4d6b44a6b2b583f45c957e15/934575e8-5bdb-478f-b763-2341a85b690e",
            "Networks":[
                {
                    "NetworkMode":"awsvpc",
                    "IPv4Addresses":["10.0.1.58"]
                }
            ],
            "Snapshotter":"overlayfs"
        }
    ],
    "ClockDrift":
        {
            "ClockErrorBound":0.487801,"ReferenceTimestamp":"2025-01-17T20:56:02Z",
            "ClockSynchronizationStatus":"SYNCHRONIZED"
        },
    "FaultInjectionEnabled":false
}
```

Example不完全な Amazon ECS コンテナメタデータファイル (まだ READY でない)

次の例では、まだ READY ステータスに達していないコンテナメタデータファイルを示します。ファイルの情報は、タスクの定義から既知であるいくつかのパラメータに限定されます。コンテナメタデータファイルは、コンテナの開始後 1 秒以内に準備できます。

```
{
    "Cluster": "default",
    "ContainerInstanceARN": "arn:aws:ecs:us-west-2:012345678910:container-instance/
default/1f73d099-b914-411c-a9ff-81633b7741dd",
    "TaskARN": "arn:aws:ecs:us-west-2:012345678910:task/default/
d90675f8-1a98-444b-805b-3d9cabb6fcd4",
    "ContainerName": "metadata"
```

```
}
```

Amazon ECS マネージドインスタンスのタスクで使用できる Amazon ECS タスクメタデータ

Amazon ECS マネージドインスタンスの Amazon ECS は、コンテナとその一部であるタスクに関するさまざまなメタデータ、ネットワークメトリクス、および Docker 統計情報 を取得する方法を提供します。これは、タスクメタデータエンドポイント と呼ばれます。タスクメタデータエンドポイントバージョン 4 は、Amazon ECS マネージドインスタンスの Amazon ECS タスクで使用できます。

awsvpc ネットワークモードで起動されたタスクに属するすべてのコンテナには、事前定義されたリンクローカルアドレス範囲内のローカル IPv4 アドレスが割り当てられます。コンテナがメタデータエンドポイントにクエリを実行する場合、コンテナエージェントは、一意の IP アドレスに基づいてコンテナが属するタスクを判断し、そのタスクのメタデータと統計情報を返します。

トピック

- Amazon ECS マネージドインスタンスのタスク用の Amazon ECS タスクメタデータエンドポイントバージョン 4

Amazon ECS マネージドインスタンスのタスク用の Amazon ECS タスクメタデータエンドポイントバージョン 4

> **Important**
>
> Amazon EC2 インスタンスでホストされている Amazon ECS タスクを使用している場合は、「Amazon ECS タスクのメタデータエンドポイント」を参照してください。

Amazon ECS マネージドインスタンス以降、ECS_CONTAINER_METADATA_URI_V4 という環境変数がタスク内の各コンテナに挿入されます。タスクメタデータエンドポイントバージョン 4 に対してクエリを実行すると、さまざまなタスクメタデータおよび Docker 統計 をタスクで利用できます。

Amazon ECS マネージドインスタンスで実行されるすべての Amazon ECS タスクについて、タスクメタデータエンドポイントはデフォルトでオンになっています。

> **Note**
>
> 新しいタスクメタデータエンドポイントバージョンを今後作成する必要がないように、追加のメタデータをバージョン 4 の出力に追加できます。既存のメタデータが削除されたり、メタデータのフィールド名が変更されたりすることはありません。

Amazon ECS マネージドインスタンスのタスクメタデータエンドポイントバージョン 4

次のタスクメタデータエンドポイントをコンテナで使用できます。

`${ECS_CONTAINER_METADATA_URI_V4}`

このパスはコンテナのメタデータを返します。

`${ECS_CONTAINER_METADATA_URI_V4}/task`

このパスはタスクのメタデータを返します。これには、タスクに関連付けられたすべてのコンテナのコンテナ ID および名前のリストが含まれています。このエンドポイントのレスポンスの詳細については、「Amazon ECS マネージドインスタンスのタスク用の Amazon ECS タスクメタデータ v4 JSON レスポンス」を参照してください。

`${ECS_CONTAINER_METADATA_URI_V4}/stats`

このパスは Docker コンテナの Docker 統計を返します。返される各統計の詳細については、Docker API ドキュメントの「ContainerStats」を参照してください。

> **Note**
>
> Amazon ECS マネージドインスタンスの Amazon ECS タスクは、コンテナの統計を返す前にコンテナを約 1 秒間実行する必要があります。

`${ECS_CONTAINER_METADATA_URI_V4}/task/stats`

このパスはタスクに関連付けられたすべてのコンテナの Docker 統計を返します。返される各統計の詳細については、Docker API ドキュメントの「ContainerStats」を参照してください。

> **Note**
>
> Amazon ECS マネージドインスタンスの Amazon ECS タスクは、コンテナの統計を返す前にコンテナを約 1 秒間実行する必要があります。

`${ECS_CONTAINER_METADATA_URI_V4}/tasks`

このパスは、コンテナインスタンスで実行されているすべてのタスクのメタデータを返します。レスポンスは、タスクメタデータオブジェクトの JSON 配列で、各要素は `/task` エンドポイントからのレスポンスと同じ構造を持ちます。各タスクのレスポンスの詳細については、「Amazon ECS マネージドインスタンスのタスク用の Amazon ECS タスクメタデータ v4 JSON レスポンス」を参照してください。

> **Note**
>
> このエンドポイントは、Amazon ECS マネージドインスタンスでマネージドデーモンサービスとして実行されているタスクでのみ利用可能です。

`${ECS_CONTAINER_METADATA_URI_V4}/tasks/stats`

このパスは、コンテナインスタンスで実行されているすべてのタスク全体のすべてのコンテナの Docker 統計を返します。レスポンスは、各要素に特定のタスクの Docker 統計へのコンテナ ID のマップが含まれる JSON 配列です。返される各統計の詳細については、Docker API ドキュメントの「ContainerStats」を参照してください。

> **Note**
>
> このエンドポイントは、Amazon ECS マネージドインスタンスでマネージドデーモンサービスとして実行されているタスクでのみ利用可能です。

Amazon ECS マネージドインスタンスのタスク用の Amazon ECS タスクメタデータ v4 JSON レスポンス

次のメタデータは、タスクメタデータエンドポイント (`${ECS_CONTAINER_METADATA_URI_V4}/task`) JSON レスポンスで返されます。

`Cluster`

タスクが属する Amazon ECS クラスターの Amazon リソースネーム (ARN)または短縮名。

`ServiceName`

タスクが属しているサービスの名前。ServiceName は、タスクがサービスに関連付けられている場合に表示されます。

`VPCID`

コンテナインスタンスの VPC ID。

`TaskARN`

コンテナが属しているタスクの Amazon リソースネーム (ARN)。

`Family`

タスクの Amazon ECS タスク定義のファミリー。

`Revision`

タスクの Amazon ECS タスク定義のリビジョン。

`DesiredStatus`

Amazon ECSからのタスクの望ましいステータス。

`KnownStatus`

Amazon ECS からのタスクの既知のステータス。

`Limits`

CPU (vCPU で表される) やメモリなど、タスクレベルで指定されたリソースの制限。リソースの制限が定義されていない場合はこのパラメータは省略されます。

`PullStartedAt`

最初のコンテナイメージのプル開始時のタイムスタンプ。

`PullStoppedAt`

最後のコンテナイメージのプル終了時のタイムスタンプ。

`AvailabilityZone`

タスクがあるアベイラビリティーゾーン。

`LaunchType`

タスクが使用している起動タイプ。Amazon ECS マネージドインスタンスで実行されているタスクの場合、起動タイプは `MANAGED_INSTANCES` です。

`Containers`

タスクに関連付けられている各コンテナのコンテナメタデータのリスト。

`DockerId`

コンテナの Docker ID。

Amazon ECS マネージドインスタンスを使用する場合、ID は 32 桁の 16 進数に 10 桁の数字が続くものとなります。

`Name`

タスク定義で指定されたコンテナの名前。

`DockerName`

Docker に提供されたコンテナの名前。Amazon ECSコンテナエージェントはコンテナに一意の名前を生成し、同じタスク定義の複数のコピーが単一のインスタンスで実行される場合に名前の競合を回避します。

`Image`

コンテナのイメージ。

`ImageID`

イメージマニフェストの SHA-256 ダイジェスト。これは、`repository-url/image@sha256:digest` 形式を使用してイメージをプルするために使用できるダイジェストです。

`Ports`

コンテナに対して公開されている任意のポート。公開ポートがない場合はこのパラメータは省略されます。

`Labels`

コンテナに適用された任意のラベル。ラベルが適用されていない場合はこのパラメータは省略されます。

`DesiredStatus`

Amazon ECSからのコンテナの望ましいステータス。

`KnownStatus`

Amazon ECS からのコンテナの既知のステータス。

`ExitCode`

コンテナの終了コード。このパラメータは、コンテナが終了していない場合は省略されます。

`Limits`

CPU (CPU 単位で表される) やメモリなど、コンテナレベルで指定されたリソースの制限。リソースの制限が定義されていない場合はこのパラメータは省略されます。

`CreatedAt`

コンテナ作成時のタイムスタンプ。このパラメータは、コンテナがまだ作成されていない場合は省略されます。

`StartedAt`

コンテナ起動時のタイムスタンプ。このパラメータは、コンテナがまだ起動していない場合は省略されます。

`FinishedAt`

コンテナ停止時のタイムスタンプ。このパラメータは、コンテナがまだ停止していない場合は省略されます。

`Type`

コンテナのタイプ。タスク定義で指定されているコンテナのタイプは `NORMAL` です。他のコンテナタイプは無視してかまいません。これらはAmazon ECS コンテナエージェントによってプロビジョニングされる内部タスクリソースで使用されます。

`LogDriver`

コンテナに使用されるログドライバー。

`LogOptions`

コンテナに定義されたログドライバオプション。

`ContainerARN`

コンテナの Amazon リソースネーム (ARN)。

`Networks`

ネットワークモードや IP アドレスなど、コンテナのネットワーク情報。ネットワーク情報が定義されていない場合はこのパラメータは省略されます。

`Snapshotter`

このコンテナイメージをダウンロードするために containerd によって使用された snapshotter。有効な値はデフォルトの `overlayfs` です。`soci` は SOCI インデックスによる遅延読み込み時に使用されます。

`RestartCount`

コンテナが再起動された回数。

> **Note**
>
> `RestartCount` メタデータが含まれるのは、コンテナの再起動ポリシーが有効になっている場合のみです。詳細については、「コンテナ再起動ポリシーを使用して Amazon ECS タスク内の個々のコンテナを再起動する」を参照してください。

`ClockDrift`

基準時刻とシステム時刻の違いに関する情報。この機能では、Amazon Time Sync Service を使用してクロックの精度を測定し、コンテナのクロックエラー範囲を特定します。詳細については、「Linux インスタンス用 Amazon EC2 ユーザーガイド」の「Linux インスタンスの時刻の設定」を参照してください。

`ReferenceTime`

クロック精度の基礎。Amazon ECS は、`2021-09-07T16:57:44Z` などの NTP を通じて協定世界時 (UTC) の世界標準を使用しています。

`ClockErrorBound`

UTC へのオフセットとして定義されるクロックエラーの対策。この誤差は、基準時刻とシステム時刻の差 (ミリ秒単位) です。

`ClockSynchronizationStatus`

最新のシステム時刻と基準時刻間の同期試行における成功状況を示します。

有効な値は `SYNCHRONIZED` および `NOT_SYNCHRONIZED` です。

`ExecutionStoppedAt`

タスクの `DesiredStatus` が `STOPPED` に移行した時のタイムスタンプ。これは、必須コンテナが `STOPPED` に移行したときに発生します。

`FaultInjectionEnabled`

タスクでフォールトインジェクションが有効になっているかどうかを示します。詳細については、「Amazon ECS および Fargate ワークロードでフォールトインジェクションを使用する」を参照してください。

Amazon ECS マネージドインスタンスのタスク用の Amazon ECS タスクメタデータ v4 の例

以下の例は、Amazon ECS マネージドインスタンスで実行される Amazon ECS タスクのタスクメタデータエンドポイントからの出力例を示しています。

コンテナから、タスクメタデータのエンドポイントに続けて curl を使用すると、`curl ${ECS_CONTAINER_METADATA_URI_V4}/task` などのエンドポイントをクエリできます。

コンテナメタデータレスポンスの例

`${ECS_CONTAINER_METADATA_URI_V4}` エンドポイントにクエリを実行すると、コンテナ自体に関するメタデータのみが返されます。以下に出力例を示します。

```
{
  "DockerId": "400c0466d1e54c0691aae0f86e0f9dc6-2531612879",
  "Name": "nginx",
  "DockerName": "nginx",
  "Image": "public.ecr.aws/nginx/nginx:latest"
  "ImageID": "sha256:d5f28ef21aabddd098f3dbc21fe5b7a7d7a184720bc07da0b6c9b9820e97f25e",
  "Labels": {
    "com.amazonaws.ecs.cluster": "arn:aws:ecs:us-west-2:111122223333:cluster/managed-
instances-cluster",
    "com.amazonaws.ecs.container-name": "curl",
    "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/managed-
instances-cluster/400c0466d1e54c0691aae0f86e0f9dc6",
    "com.amazonaws.ecs.task-definition-family": "ecs-managed-instances-task-def",
    "com.amazonaws.ecs.task-definition-version": "7"
  },
  "DesiredStatus": "RUNNING",
  "KnownStatus": "RUNNING",
  "Limits": { "CPU": 2 },
  "CreatedAt": "2025-09-24T19:39:04.88336233Z",
  "StartedAt": "2025-09-24T19:39:04.88336233Z",
  "Type": "NORMAL",
  "LogDriver": "awslogs",
  "LogOptions": {
    "awslogs-create-group": "true",
```

```
    "awslogs-group": "/ecs/managed-instances-task-def",
    "awslogs-region": "us-west-2",
    "awslogs-stream": "ecs/nginx/400c0466d1e54c0691aae0f86e0f9dc6"
  },
  "ContainerARN": "arn:aws:ecs:us-west-2:111122223333:container/managed-instances-
cluster/400c0466d1e54c0691aae0f86e0f9dc6/3703f0e4-a351-4f1e-a0e7-6981ea7adc8b",
  "Networks": [
    {
      "NetworkMode": "awsvpc",
      "IPv4Addresses": ["172.31.62.223"],
      "AttachmentIndex": 0,
      "MACAddress": "0e:41:0b:1e:f2:fb",
      "IPv4SubnetCIDRBlock": "172.31.48.0/20",
      "PrivateDNSName": "ip-172-31-62-223.us-west-2.compute.internal",
      "SubnetGatewayIpv4Address": "172.31.48.1/20"
    }
  ],
  "Snapshotter": "overlayfs"
}
```

Amazon ECS マネージドインスタンスのタスク用の Amazon ECS タスクメタデータ v4 の例

`${ECS_CONTAINER_METADATA_URI_V4}/task` エンドポイントにクエリを実行すると、コンテナが属しているタスクに関するメタデータが返されます。以下に出力例を示します。

```
{
  "Cluster": "arn:aws:ecs:us-west-2:111122223333:cluster/managed-instances-cluster",
  "TaskARN": "arn:aws:ecs:us-west-2:111122223333:task/managed-instances-
cluster/400c0466d1e54c0691aae0f86e0f9dc6",
  "Family": "managed-instances-task-def",
  "Revision": "7",
  "DesiredStatus": "RUNNING",
  "KnownStatus": "RUNNING",
  "Limits": { "CPU": 1, "Memory": 3072 },
  "PullStartedAt": "2025-09-24T19:38:58.682942001Z",
  "PullStoppedAt": "2025-09-24T19:39:03.091597524Z",
  "AvailabilityZone": "us-west-2d",
  "LaunchType": "MANAGED_INSTANCES",
  "Containers": [
    {
      "DockerId": "400c0466d1e54c0691aae0f86e0f9dc6-2531612879",
      "Name": "curl",
      "DockerName": "curl",
```

```
      "Image": "nginx",
      "ImageID":
 "sha256:d5f28ef21aabddd098f3dbc21fe5b7a7d7a184720bc07da0b6c9b9820e97f25e",
      "Labels": {
        "com.amazonaws.ecs.cluster": "arn:aws:ecs:us-west-2:111122223333:cluster/
managed-instances-cluster",
        "com.amazonaws.ecs.container-name": "nginx",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/managed-
instances-cluster/400c0466d1e54c0691aae0f86e0f9dc6",
        "com.amazonaws.ecs.task-definition-family": "managed-instances-task-def",
        "com.amazonaws.ecs.task-definition-version": "7"
      },
      "DesiredStatus": "RUNNING",
      "KnownStatus": "RUNNING",
      "Limits": { "CPU": 2 },
      "CreatedAt": "2025-09-24T19:39:04.88336233Z",
      "StartedAt": "2025-09-24T19:39:04.88336233Z",
      "Type": "NORMAL",
      "LogDriver": "awslogs",
      "LogOptions": {
        "awslogs-create-group": "true",
        "awslogs-group": "/ecs/managed-instances-task-def",
        "awslogs-region": "us-west-2",
        "awslogs-stream": "ecs/nginx/400c0466d1e54c0691aae0f86e0f9dc6"
      },
      "ContainerARN": "arn:aws:ecs:us-west-2:111122223333:container/managed-instances-
cluster/400c0466d1e54c0691aae0f86e0f9dc6/3703f0e4-a351-4f1e-a0e7-6981ea7adc8b",
      "Networks": [
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": ["172.31.62.223"],
          "AttachmentIndex": 0,
          "MACAddress": "0e:41:0b:1e:f2:fb",
          "IPv4SubnetCIDRBlock": "172.31.48.0/20",
          "PrivateDNSName": "ip-172-31-62-223.us-west-2.compute.internal",
          "SubnetGatewayIpv4Address": "172.31.48.1/20"
        }
      ],
      "Snapshotter": "overlayfs"
    }
  ],
  "ServiceName": "exec-service-2",
  "ClockDrift": {
    "ClockErrorBound": 0.14120749999999999,
```

```
    "ReferenceTimestamp": "2025-09-24T19:48:37Z",
    "ClockSynchronizationStatus": "SYNCHRONIZED"
  },
  "FaultInjectionEnabled": false
}
```

タスク統計情報のレスポンス例

${ECS_CONTAINER_METADATA_URI_V4}/task/stats エンドポイントにクエリを実行すると、コンテナが属しているタスクに関するネットワークメトリクスが返されます。以下に出力例を示します。

```
{
  "400c0466d1e54c0691aae0f86e0f9dc6-2531612879": {
    "read": "2025-09-24T19:51:54.899736614Z",
    "preread": "2025-09-24T19:51:44.901199024Z",
    "pids_stats": {},
    "blkio_stats": {
      "io_service_bytes_recursive": [
        { "major": 259, "minor": 1, "op": "read", "value": 0 },
        { "major": 259, "minor": 1, "op": "write", "value": 131072 },
        { "major": 259, "minor": 0, "op": "read", "value": 48173056 },
        { "major": 259, "minor": 0, "op": "write", "value": 0 },
        { "major": 252, "minor": 0, "op": "read", "value": 48173056 },
        { "major": 252, "minor": 0, "op": "write", "value": 0 }
      ],
      "io_serviced_recursive": null,
      "io_queue_recursive": null,
      "io_service_time_recursive": null,
      "io_wait_time_recursive": null,
      "io_merged_recursive": null,
      "io_time_recursive": null,
      "sectors_recursive": null
    },
    "num_procs": 0,
    "storage_stats": {},
    "cpu_stats": {
      "cpu_usage": {
        "total_usage": 670462000,
        "usage_in_kernelmode": 276072000,
        "usage_in_usermode": 394389000
      },
      "system_cpu_usage": 787660000000,
```

```
      "online_cpus": 1,
      "throttling_data": {
        "periods": 0,
        "throttled_periods": 0,
        "throttled_time": 0
      }
    },
    "precpu_stats": {
      "cpu_usage": {
        "total_usage": 663809000,
        "usage_in_kernelmode": 273333000,
        "usage_in_usermode": 390476000
      },
      "system_cpu_usage": 777710000000,
      "online_cpus": 1,
      "throttling_data": {
        "periods": 0,
        "throttled_periods": 0,
        "throttled_time": 0
      }
    },
    "memory_stats": {
      "usage": 83562496,
      "stats": {
        "active_anon": 24576,
        "active_file": 258048,
        "anon": 32264192,
        "anon_thp": 0,
        "file": 48533504,
        "file_dirty": 0,
        "file_mapped": 36286464,
        "file_writeback": 0,
        "inactive_anon": 32243712,
        "inactive_file": 48271360,
        "kernel_stack": 442368,
        "pgactivate": 6,
        "pgdeactivate": 0,
        "pgfault": 18936,
        "pglazyfree": 0,
        "pglazyfreed": 0,
        "pgmajfault": 311,
        "pgrefill": 0,
        "pgscan": 0,
        "pgsteal": 0,
```

```
        "shmem": 4096,
        "slab": 1598960,
        "slab_reclaimable": 1080040,
        "slab_unreclaimable": 518920,
        "sock": 0,
        "thp_collapse_alloc": 0,
        "thp_fault_alloc": 0,
        "unevictable": 0,
        "workingset_activate": 0,
        "workingset_nodereclaim": 0,
        "workingset_refault": 0
      },
      "limit": 18446744073709551615
    },
    "name": "nginx",
    "id": "400c0466d1e54c0691aae0f86e0f9dc6-2531612879"
  }
}
```


EC2 のタスクで使用できる Amazon ECS タスクメタデータ

Amazon ECS コンテナエージェントでは、さまざまなタスクメタデータおよび [Docker]統計を取得するメソッドを提供します。これは、タスクメタデータエンドポイントと呼ばれます。以下のバージョンが利用できます。

- タスクメタデータエンドポイントバージョン 4 コンテナにさまざまなメタデータと Docker 統計を提供します。ネットワークレートデータも提供できます。少なくともバージョン`1.39.0`のAmazon ECS コンテナエージェントを実行しているAmazon EC2 Linuxインスタンスで起動されたAmazon ECSタスクで利用できます。`awsvpc` ネットワークモードを使用するAmazon EC2 Windows インスタンスの場合、Amazon ECS コンテナエージェントは少なくともバージョン `1.54.0` でなければなりません。詳細については、「Amazon ECS タスクメタデータエンドポイントバージョン 4」を参照してください。
- タスクメタデータエンドポイントバージョン 3 コンテナにさまざまなメタデータと Docker 統計を提供します。少なくともバージョン`1.21.0`のAmazon ECS コンテナエージェントを実行しているAmazon EC2 Linuxインスタンスで起動されたAmazon ECSタスクで利用できます。`awsvpc` ネットワークモードを使用するAmazon EC2 Windows インスタンスの場合、Amazon ECS コンテナエージェントは少なくともバージョン `1.54.0` でなければなりません。詳細については、「Amazon ECS タスクメタデータエンドポイントバージョン 3」を参照してください。

- タスクメタデータエンドポイントバージョン2–少なくともバージョン `1.17.0`のAmazonECS コンテナエージェントを実行しているAmazon EC2 Linuxインスタンスで起動されたAmazon ECS タスクで利用できます。`awsvpc` ネットワークモードを使用するAmazon EC2 Windows インスタンスの場合、Amazon ECS コンテナエージェントは少なくともバージョン `1.54.0` でなければなりません。詳細については、「Amazon ECS タスクメタデータエンドポイントバージョン 2」を参照してください。

Amazon ECS タスクが Amazon EC2 でホストされている場合、またはタスクが `host` ネットワークモードを使用して Amazon ECS マネージドインスタンスでホストされている場合、インスタンスメタデータサービス (IMDS) エンドポイントを使用してタスクホストのメタデータにアクセスすることもできます。次のコマンドは、タスクをホストするインスタンス内から実行すると、ホストインスタンスの ID を一覧表示します。

```
curl http://169.254.169.254/latest/meta-data/instance-id
```

Amazon ECS タスクが Amazon EC2 上で、IPv6-only 設定でホストされている場合、IPv6 IMDS エンドポイントを使用してタスクホストのメタデータにアクセスできます。次のコマンドは、タスクをホストするインスタンス内から実行すると、IPv6 上のホストインスタンスの ID を一覧表示します。

```
curl http://[fd00:ec2::254]/latest/meta-data/instance-id
```

IPv6 IMDS エンドポイントにアクセスするには、コンテナインスタンスで IPv6 IMDS エンドポイントを有効にし、選択した SDK の IMDS 認証情報プロバイダーを使用してメタデータサービスエンドポイントモードを `IPv6` に設定します。コンテナインスタンスの IPv6 IMDS エンドポイントを有効にする方法については、「Amazon EC2 ユーザーガイド」の「インスタンスメタデータサービスのオプションを設定する」を参照してください。SDK の IMDS 認証情報プロバイダーの詳細については、AWS SDK とツールのリファレンスガイドの「IMDS 認証情報プロバイダー」を参照してください。

> **Note**
>
> `awsvpcTrunking` アカウント設定が有効になっている場合、IPv6 IMDS エンドポイントにはアクセスできません。`awsvpcTrunking` が有効になっている状態でコンテナインスタンスの IAM ロール認証情報にアクセスするには、代わりにタスクの IAM ロールを使用します。タスクの IAM ロールの詳細については、「Amazon ECS タスクの IAM ロール」を参照してください。

エンドポイントから取得できる情報は、*instance-id* などのようなカテゴリに分類されます。エンドポイントを使用して取得できるホストインスタンスのメタデータのさまざまなカテゴリの詳細については、「インスタンスメタデータのカテゴリ」を参照してください。

Amazon ECS タスクメタデータエンドポイントバージョン 4

Amazon ECS コンテナエージェントは、各コンテナに環境変数を注入します。これは、タスクメタデータエンドポイントと呼ばれ、コンテナにさまざまなタスクメタデータと Docker 統計を提供します。

タスクメタデータとネットワークレートの統計は CloudWatch に送信され、 AWS マネジメントコンソール で表示できます。詳細については、「オブザーバビリティが強化された Container Insights を使用し、Amazon ECS コンテナを監視する」を参照してください。

> **Note**
>
> Amazon ECS は、以前のバージョンのタスクメタデータエンドポイントを提供しています。新しいタスクメタデータエンドポイントバージョンを今後作成する必要がないように、追加のメタデータをバージョン 4 の出力に追加できます。既存のメタデータが削除されたり、メタデータのフィールド名が変更されたりすることはありません。

環境変数は、デフォルトでは、少なくともバージョン `1.39.0` の Amazon ECS コンテナエージェントを実行している Amazon ECS Linux インスタンスで起動された Amazon EC2 タスクのコンテナに挿入されます。`awsvpc` ネットワークモードを使用するAmazon EC2 Windows インスタンスの場合、Amazon ECS コンテナエージェントは少なくともバージョン `1.54.0` でなければなりません。詳細については、「Amazon ECS Linux コンテナインスタンスの管理」を参照してください。

> **Note**
>
> エージェントを最新バージョンに更新することで、古いバージョンのAmazon ECS コンテナエージェントを使用する Amazon EC2 インスタンスでこの機能のサポートを追加できます。詳細については、「Amazon ECS コンテナエージェントをアップデートする」を参照してください。

タスクメタデータの出力例については、「Amazon ECS タスクメタデータ v4 の例」を参照してください。

タスクメタデータエンドポイントバージョン 4 のパス

次のタスクメタデータエンドポイントパスをコンテナで使用できます。

`${ECS_CONTAINER_METADATA_URI_V4}`

このパスはコンテナのメタデータを返します。

`${ECS_CONTAINER_METADATA_URI_V4}/task`

このパスはタスクのメタデータを返します。これには、タスクに関連付けられたすべてのコンテナのコンテナ ID および名前のリストが含まれています。このエンドポイントのレスポンスの詳細については、「Amazon ECS タスクメタデータ V4 JSON レスポンス」を参照してください。

`${ECS_CONTAINER_METADATA_URI_V4}/taskWithTags`

このパスは、`/task` API を使用して取得できるタスクとコンテナインスタンスタグに加えて、`ListTagsForResource` エンドポイントに含まれるタスクのメタデータを返します。タグメタデータの取得時に受信したエラーは、レスポンスの `Errors` フィールドに含まれます。

> **Note**
>
> この`Errors`フィールドは、少なくとも`1.50.0`バージョンのコンテナエージェントを実行しているAmazon EC2 Linux インスタンスでホストされているタスクの応答にのみ表示されます。`awsvpc` ネットワークモードを使用するAmazon EC2 Windows インスタンスの場合、Amazon ECS コンテナエージェントは少なくともバージョン `1.54.0` でなければなりません。
> このエンドポイントには、`ecs.ListTagsForResource` アクセス許可が必要です。

> **Important**
>
> `${ECS_CONTAINER_METADATA_URI_V4}/taskWithTags` エンドポイントを使用する場合、各呼び出しが `ecs:ListTagsForResource` に対して最大 2 つの API リクエストを行うこと (1 つはコンテナインスタンスタグ用、もう 1 つはタスクタグ用)、そして、タスク内のサイドカーコンテナがユーザーに代わってこれらの呼び出しを実行する可能性があることに留意してください。エンドポイントを頻繁に呼び出すと、API スロットリングが発生する可能性があります。
> 特にトラフィックの多いアプリケーションでは、呼び出す頻度を減らすために
> キャッシュまたはバッチ処理戦略を実施することを検討してください。また、AWS

CloudTrail を使用して API のスロットリング問題をデバッグすることも考慮します。`ListTagsForResource` API のスロットリング制限の詳細については、「Amazon Elastic Container Service API リファレンス」の「Request throttling for the Amazon ECS API」を参照してください。AWS CloudTrail を使用した Amazon ECS API コールのデバッグについては、「AWS CloudTrail を使用して Amazon ECS API コールを口グに記録する」を参照してください。

`${ECS_CONTAINER_METADATA_URI_V4}/stats`

このパスは特定のコンテナの Docker 統計を返します。返される各統計の詳細については、Docker API ドキュメントの「ContainerStats」を参照してください。

少なくともバージョン `1.43.0` のコンテナエージェントを実行しているAmazon EC2 Linuxインスタンスでホストされている `awsvpc` または `bridge` ネットワークモードを使用するAmazon ECS タスクの場合、応答に追加のネットワークレート統計が含まれます。他のすべてのタスクでは、レスポンスには累積ネットワーク統計のみが含まれます。

`${ECS_CONTAINER_METADATA_URI_V4}/task/stats`

このパスはタスクに関連付けられたすべてのコンテナの Docker 統計を返します。これをサイドカーコンテナで使用し、ネットワークメトリクスを抽出できます。返される各統計の詳細については、Docker API ドキュメントの「ContainerStats」を参照してください。

少なくともバージョン `1.43.0` のコンテナエージェントを実行しているAmazon EC2 Linuxインスタンスでホストされている `awsvpc` または `bridge` ネットワークモードを使用するAmazon ECS タスクの場合、応答に追加のネットワークレート統計が含まれます。他のすべてのタスクでは、レスポンスには累積ネットワーク統計のみが含まれます。

Amazon ECS タスクメタデータ V4 JSON レスポンス

次の情報が、タスクメタデータエンドポイント (`${ECS_CONTAINER_METADATA_URI_V4}/task`) JSON レスポンスから返されます。これには、タスク内の各コンテナのメタデータに加えて、タスクに関連付けられたメタデータも含まれます。

`Cluster`

タスクが属する Amazon ECS クラスターの Amazon リソースネーム (ARN)または短縮名。

`ServiceName`

タスクが属しているサービスの名前。ServiceName は、タスクがサービスに関連付けられている場合、Amazon EC2 と Amazon ECS Anywhere コンテナインスタンスに表示されます。

> **Note**
>
> この `ServiceName` メタデータは、Amazon ECS コンテナエージェントのバージョン `1.63.1`、もしくはそれ以降を使用している場合にのみ含まれます。

`VPCID`

Amazon EC2 コンテナインスタンスの VPC ID。このフィールドは、Amazon EC2 インスタンスでのみ表示されます。

> **Note**
>
> この `VPCID` メタデータは、Amazon ECS コンテナエージェントのバージョン `1.63.1`、もしくはそれ以降を使用している場合にのみ含まれます。

`TaskARN`

コンテナが属しているタスクの Amazon リソースネーム (ARN)。

`Family`

タスクの Amazon ECS タスク定義のファミリー。

`Revision`

タスクの Amazon ECS タスク定義のリビジョン。

`DesiredStatus`

Amazon ECSからのタスクの望ましいステータス。

`KnownStatus`

Amazon ECS からのタスクの既知のステータス。

`Limits`

CPU (vCPU で表される) やメモリなど、タスク レベルで指定されたリソースの制限。リソースの制限が定義されていない場合はこのパラメータは省略されます。

`PullStartedAt`

最初のコンテナイメージのプル開始時のタイムスタンプ。

`PullStoppedAt`

最後のコンテナイメージのプル終了時のタイムスタンプ。

`AvailabilityZone`

タスクがあるアベイラビリティーゾーン。

> **Note**
>
> アベイラビリティーゾーンのメタデータは、プラットフォームバージョン 1.4 以降 (Linux) または 1.0.0 (Windows) を使用する Fargate タスクでのみ使用できます。

`LaunchType`

タスクが使用している起動タイプ。クラスター容量プロバイダーを使用する場合、タスクが Fargate インフラストラクチャと EC2 インフラストラクチャのいずれを使用しているかを示します。

> **Note**
>
> この`LaunchType`メタデータは、Amazon ECS Linux コンテナエージェントのバージョン`1.45.0` 以降 (Linux) または 1.0.0 以降 (Windows) を使用している場合にのみ含まれます。

`Containers`

タスクに関連付けられている各コンテナのコンテナメタデータのリスト。

`DockerId`

コンテナの Docker ID。

Fargate を使用する場合、IDは32桁の16進数に10桁の数字が続きます。

`Name`

タスク定義で指定されたコンテナの名前。

`DockerName`

Docker に提供されたコンテナの名前。Amazon ECSコンテナエージェントはコンテナに一意の名前を生成し、同じタスク定義の複数のコピーが単一のインスタンスで実行される場合に名前の競合を回避します。

`Image`

コンテナのイメージ。

`ImageID`

イメージマニフェストの SHA-256 ダイジェスト。これは、`repository-url/image@sha256:digest` 形式を使用してイメージをプルするために使用できるダイジェストです。

`Ports`

コンテナに対して公開されている任意のポート。公開ポートがない場合はこのパラメータは省略されます。

`Labels`

コンテナに適用された任意のラベル。ラベルが適用されていない場合はこのパラメータは省略されます。

`DesiredStatus`

Amazon ECSからのコンテナの望ましいステータス。

`KnownStatus`

Amazon ECS からのコンテナの既知のステータス。

`ExitCode`

コンテナの終了コード。このパラメータは、コンテナが終了していない場合は省略されます。

`Limits`

CPU (CPU 単位で表される) やメモリなど、コンテナレベルで指定されたリソースの制限。リソースの制限が定義されていない場合はこのパラメータは省略されます。

`CreatedAt`

コンテナ作成時のタイムスタンプ。このパラメータは、コンテナがまだ作成されていない場合は省略されます。

`StartedAt`

コンテナ起動時のタイムスタンプ。このパラメータは、コンテナがまだ起動していない場合は省略されます。

`FinishedAt`

コンテナ停止時のタイムスタンプ。このパラメータは、コンテナがまだ停止していない場合は省略されます。

`Type`

コンテナのタイプ。タスク定義で指定されているコンテナのタイプは `NORMAL` です。他のコンテナタイプは無視してかまいません。これらはAmazon ECS コンテナエージェントによってプロビジョニングされる内部タスクリソースで使用されます。

`LogDriver`

コンテナに使用されるログドライバー。

> **Note**
>
> この`LogDriver`メタデータは、Amazon ECS Linux コンテナエージェントのバージョン`1.45.0`、もしくはそれ以降を使用している場合にのみ含まれます。

`LogOptions`

コンテナに定義されたログドライバオプション。

> **Note**
>
> この`LogOptions`メタデータは、Amazon ECS Linux コンテナエージェントのバージョン`1.45.0`、もしくはそれ以降を使用している場合にのみ含まれます。

`ContainerARN`

コンテナの Amazon リソースネーム (ARN)。

> **Note**
>
> このContainerARNメタデータは、Amazon ECS Linux コンテナエージェントのバージョン1.45.0、もしくはそれ以降を使用している場合にのみ含まれます。

`Networks`

ネットワークモードや IP アドレスなど、コンテナのネットワーク情報。ネットワーク情報が定義されていない場合はこのパラメータは省略されます。

`RestartCount`

コンテナが再起動された回数。

> **Note**
>
> `RestartCount` メタデータが含まれるのは、コンテナの再起動ポリシーが有効になっている場合のみです。詳細については、「コンテナ再起動ポリシーを使用して Amazon ECS タスク内の個々のコンテナを再起動する」を参照してください。

`ExecutionStoppedAt`

タスクの `DesiredStatus` が `STOPPED` に移行した時のタイムスタンプ。これは、必須コンテナが `STOPPED` に移行したときに発生します。

Amazon ECS タスクメタデータ v4 の例

以下の例は、タスクメタデータエンドポイントそれぞれからの出力例を示しています。

コンテナメタデータレスポンスの例

`${ECS_CONTAINER_METADATA_URI_V4}` エンドポイントにクエリを実行すると、コンテナ自体に関するメタデータのみが返されます。以下は、サービス (MyService) の一部として実行されるタスクからの出力例です。

```
{
    "DockerId": "ea32192c8553fbff06c9340478a2ff089b2bb5646fb718b4ee206641c9086d66",
    "Name": "curl",
    "DockerName": "ecs-curltest-24-curl-cca48e8dcadd97805600",
```

```
    "Image": "111122223333.dkr.ecr.us-west-2.amazonaws.com/curltest:latest",
    "ImageID":
 "sha256:d691691e9652791a60114e67b365688d20d19940dde7c4736ea30e660d8d3553",
    "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/
default/8f03e41243824aea923aca126495f665",
        "com.amazonaws.ecs.task-definition-family": "curltest",
        "com.amazonaws.ecs.task-definition-version": "24"
    },
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "Limits": {
        "CPU": 10,
        "Memory": 128
    },
    "CreatedAt": "2020-10-02T00:15:07.620912337Z",
    "StartedAt": "2020-10-02T00:15:08.062559351Z",
    "Type": "NORMAL",
    "LogDriver": "awslogs",
    "LogOptions": {
        "awslogs-create-group": "true",
        "awslogs-group": "/ecs/metadata",
        "awslogs-region": "us-west-2",
        "awslogs-stream": "ecs/curl/8f03e41243824aea923aca126495f665"
    },
    "ContainerARN": "arn:aws:ecs:us-west-2:111122223333:container/0206b271-
b33f-47ab-86c6-a0ba208a70a9",
    "Networks": [
        {
            "NetworkMode": "awsvpc",
            "IPv4Addresses": [
                "10.0.2.100"
            ],
            "AttachmentIndex": 0,
            "MACAddress": "0e:9e:32:c7:48:85",
            "IPv4SubnetCIDRBlock": "10.0.2.0/24",
            "PrivateDNSName": "ip-10-0-2-100.us-west-2.compute.internal",
            "SubnetGatewayIpv4Address": "10.0.2.1/24"
        }
    ]
}
```

タスクメタデータレスポンスの例

${ECS_CONTAINER_METADATA_URI_V4}/task エンドポイントをクエリすると、タスク内の各コンテナのメタデータに加えて、コンテナが一部であるタスクに関するメタデータが返されます。以下に出力例を示します。

```
{
    "Cluster": "default",
    "TaskARN": "arn:aws:ecs:us-west-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
    "Family": "curltest",
    "ServiceName": "MyService",
    "Revision": "26",
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "PullStartedAt": "2020-10-02T00:43:06.202617438Z",
    "PullStoppedAt": "2020-10-02T00:43:06.31288465Z",
    "AvailabilityZone": "us-west-2d",
    "VPCID": "vpc-1234567890abcdef0",
    "LaunchType": "EC2",
    "Containers": [
        {
            "DockerId":
 "598cba581fe3f939459eaba1e071d5c93bb2c49b7d1ba7db6bb19deeb70d8e38",
            "Name": "~internal~ecs~pause",
            "DockerName": "ecs-curltest-26-internalecspause-e292d586b6f9dade4a00",
            "Image": "amazon/amazon-ecs-pause:0.1.0",
            "ImageID": "",
            "Labels": {
                "com.amazonaws.ecs.cluster": "default",
                "com.amazonaws.ecs.container-name": "~internal~ecs~pause",
                "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
                "com.amazonaws.ecs.task-definition-family": "curltest",
                "com.amazonaws.ecs.task-definition-version": "26"
            },
            "DesiredStatus": "RESOURCES_PROVISIONED",
            "KnownStatus": "RESOURCES_PROVISIONED",
            "Limits": {
                "CPU": 0,
                "Memory": 0
            },
            "CreatedAt": "2020-10-02T00:43:05.602352471Z",
```

```
            "StartedAt": "2020-10-02T00:43:06.076707576Z",
            "Type": "CNI_PAUSE",
            "Networks": [
                {
                    "NetworkMode": "awsvpc",
                    "IPv4Addresses": [
                        "10.0.2.61"
                    ],
                    "AttachmentIndex": 0,
                    "MACAddress": "0e:10:e2:01:bd:91",
                    "IPv4SubnetCIDRBlock": "10.0.2.0/24",
                    "PrivateDNSName": "ip-10-0-2-61.us-west-2.compute.internal",
                    "SubnetGatewayIpv4Address": "10.0.2.1/24"
                }
            ]
        },
        {
            "DockerId":
 "ee08638adaaf009d78c248913f629e38299471d45fe7dc944d1039077e3424ca",
            "Name": "curl",
            "DockerName": "ecs-curltest-26-curl-a0e7dba5aca6d8cb2e00",
            "Image": "111122223333.dkr.ecr.us-west-2.amazonaws.com/curltest:latest",
            "ImageID":
 "sha256:d691691e9652791a60114e67b365688d20d19940dde7c4736ea30e660d8d3553",
            "Labels": {
                "com.amazonaws.ecs.cluster": "default",
                "com.amazonaws.ecs.container-name": "curl",
                "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
                "com.amazonaws.ecs.task-definition-family": "curltest",
                "com.amazonaws.ecs.task-definition-version": "26"
            },
            "DesiredStatus": "RUNNING",
            "KnownStatus": "RUNNING",
            "Limits": {
                "CPU": 10,
                "Memory": 128
            },
            "CreatedAt": "2020-10-02T00:43:06.326590752Z",
            "StartedAt": "2020-10-02T00:43:06.767535449Z",
            "Type": "NORMAL",
            "LogDriver": "awslogs",
            "LogOptions": {
                "awslogs-create-group": "true",
```

```
                "awslogs-group": "/ecs/metadata",
                "awslogs-region": "us-west-2",
                "awslogs-stream": "ecs/curl/158d1c8083dd49d6b527399fd6414f5c"
            },
            "ContainerARN": "arn:aws:ecs:us-west-2:111122223333:container/
abb51bdd-11b4-467f-8f6c-adcfe1fe059d",
            "Networks": [
                {
                    "NetworkMode": "awsvpc",
                    "IPv4Addresses": [
                        "10.0.2.61"
                    ],
                    "AttachmentIndex": 0,
                    "MACAddress": "0e:10:e2:01:bd:91",
                    "IPv4SubnetCIDRBlock": "10.0.2.0/24",
                    "PrivateDNSName": "ip-10-0-2-61.us-west-2.compute.internal",
                    "SubnetGatewayIpv4Address": "10.0.2.1/24"
                }
            ]
        }
    ]
}
```

タグメタデータの応答を持つタスクの例

${ECS_CONTAINER_METADATA_URI_V4}/taskWithTags エンドポイントをクエリすると、タスクおよびコンテナインスタンスタグを含むタスクに関するメタデータが返されます。以下に出力例を示します。

```
{
    "Cluster": "default",
    "TaskARN": "arn:aws:ecs:us-west-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
    "Family": "curltest",
    "ServiceName": "MyService",
    "Revision": "26",
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "PullStartedAt": "2020-10-02T00:43:06.202617438Z",
    "PullStoppedAt": "2020-10-02T00:43:06.31288465Z",
    "AvailabilityZone": "us-west-2d",
    "VPCID": "vpc-1234567890abcdef0",
    "TaskTags": {
```

```
        "tag-use": "task-metadata-endpoint-test"
    },
    "ContainerInstanceTags":{
        "tag_key":"tag_value"
    },
    "LaunchType": "EC2",
    "Containers": [
        {
            "DockerId":
 "598cba581fe3f939459eaba1e071d5c93bb2c49b7d1ba7db6bb19deeb70d8e38",
            "Name": "~internal~ecs~pause",
            "DockerName": "ecs-curltest-26-internalecspause-e292d586b6f9dade4a00",
            "Image": "amazon/amazon-ecs-pause:0.1.0",
            "ImageID": "",
            "Labels": {
                "com.amazonaws.ecs.cluster": "default",
                "com.amazonaws.ecs.container-name": "~internal~ecs~pause",
                "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
                "com.amazonaws.ecs.task-definition-family": "curltest",
                "com.amazonaws.ecs.task-definition-version": "26"
            },
            "DesiredStatus": "RESOURCES_PROVISIONED",
            "KnownStatus": "RESOURCES_PROVISIONED",
            "Limits": {
                "CPU": 0,
                "Memory": 0
            },
            "CreatedAt": "2020-10-02T00:43:05.602352471Z",
            "StartedAt": "2020-10-02T00:43:06.076707576Z",
            "Type": "CNI_PAUSE",
            "Networks": [
                {
                    "NetworkMode": "awsvpc",
                    "IPv4Addresses": [
                        "10.0.2.61"
                    ],
                    "AttachmentIndex": 0,
                    "MACAddress": "0e:10:e2:01:bd:91",
                    "IPv4SubnetCIDRBlock": "10.0.2.0/24",
                    "PrivateDNSName": "ip-10-0-2-61.us-west-2.compute.internal",
                    "SubnetGatewayIpv4Address": "10.0.2.1/24"
                }
            ]
```

```
        },
        {
            "DockerId":
 "ee08638adaaf009d78c248913f629e38299471d45fe7dc944d1039077e3424ca",
            "Name": "curl",
            "DockerName": "ecs-curltest-26-curl-a0e7dba5aca6d8cb2e00",
            "Image": "111122223333.dkr.ecr.us-west-2.amazonaws.com/curltest:latest",
            "ImageID":
 "sha256:d691691e9652791a60114e67b365688d20d19940dde7c4736ea30e660d8d3553",
            "Labels": {
                "com.amazonaws.ecs.cluster": "default",
                "com.amazonaws.ecs.container-name": "curl",
                "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
                "com.amazonaws.ecs.task-definition-family": "curltest",
                "com.amazonaws.ecs.task-definition-version": "26"
            },
            "DesiredStatus": "RUNNING",
            "KnownStatus": "RUNNING",
            "Limits": {
                "CPU": 10,
                "Memory": 128
            },
            "CreatedAt": "2020-10-02T00:43:06.326590752Z",
            "StartedAt": "2020-10-02T00:43:06.767535449Z",
            "Type": "NORMAL",
            "LogDriver": "awslogs",
            "LogOptions": {
                "awslogs-create-group": "true",
                "awslogs-group": "/ecs/metadata",
                "awslogs-region": "us-west-2",
                "awslogs-stream": "ecs/curl/158d1c8083dd49d6b527399fd6414f5c"
            },
            "ContainerARN": "arn:aws:ecs:us-west-2:111122223333:container/
abb51bdd-11b4-467f-8f6c-adcfe1fe059d",
            "Networks": [
                {
                    "NetworkMode": "awsvpc",
                    "IPv4Addresses": [
                        "10.0.2.61"
                    ],
                    "AttachmentIndex": 0,
                    "MACAddress": "0e:10:e2:01:bd:91",
                    "IPv4SubnetCIDRBlock": "10.0.2.0/24",
```

```
                    "PrivateDNSName": "ip-10-0-2-61.us-west-2.compute.internal",
                    "SubnetGatewayIpv4Address": "10.0.2.1/24"
                }
            ]
        }
    ]
}
```

エラーメタデータレスポンスのタグを含むタスクの例

${ECS_CONTAINER_METADATA_URI_V4}/taskWithTags エンドポイントをクエリすると、タスクおよびコンテナインスタンスタグを含むタスクに関するメタデータが返されます。タグ付けデータの取得中にエラーが発生した場合は、そのエラーがレスポンスで返されます。コンテナインスタンスに関連付けられた IAM ロール に ecs:ListTagsForResource の権限が付与されていない場合の出力例を次に示します。

```
{
    "Cluster": "default",
    "TaskARN": "arn:aws:ecs:us-west-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
    "Family": "curltest",
    "ServiceName": "MyService",
    "Revision": "26",
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "PullStartedAt": "2020-10-02T00:43:06.202617438Z",
    "PullStoppedAt": "2020-10-02T00:43:06.31288465Z",
    "AvailabilityZone": "us-west-2d",
    "VPCID": "vpc-1234567890abcdef0",
    "Errors": [
        {
            "ErrorField": "ContainerInstanceTags",
            "ErrorCode": "AccessDeniedException",
            "ErrorMessage": "User: arn:aws:sts::111122223333:assumed-
role/ecsInstanceRole/i-0744a608689EXAMPLE is not authorized to perform:
 ecs:ListTagsForResource on resource: arn:aws:ecs:us-west-2:111122223333:container-
instance/default/2dd1b186f39845a584488d2ef155c131",
            "StatusCode": 400,
            "RequestId": "cd597ef0-272b-4643-9bd2-1de469870fa6",
            "ResourceARN": "arn:aws:ecs:us-west-2:111122223333:container-instance/
default/2dd1b186f39845a584488d2ef155c131"
        },
        {
```

```
            "ErrorField": "TaskTags",
            "ErrorCode": "AccessDeniedException",
            "ErrorMessage": "User: arn:aws:sts::111122223333:assumed-
role/ecsInstanceRole/i-0744a608689EXAMPLE is not authorized to perform:
 ecs:ListTagsForResource on resource: arn:aws:ecs:us-west-2:111122223333:task/
default/9ef30e4b7aa44d0db562749cff4983f3",
            "StatusCode": 400,
            "RequestId": "862c5986-6cd2-4aa6-87cc-70be395531e1",
            "ResourceARN": "arn:aws:ecs:us-west-2:111122223333:task/
default/9ef30e4b7aa44d0db562749cff4983f3"
        }
    ],
    "LaunchType": "EC2",
    "Containers": [
        {
            "DockerId":
 "598cba581fe3f939459eaba1e071d5c93bb2c49b7d1ba7db6bb19deeb70d8e38",
            "Name": "~internal~ecs~pause",
            "DockerName": "ecs-curltest-26-internalecspause-e292d586b6f9dade4a00",
            "Image": "amazon/amazon-ecs-pause:0.1.0",
            "ImageID": "",
            "Labels": {
                "com.amazonaws.ecs.cluster": "default",
                "com.amazonaws.ecs.container-name": "~internal~ecs~pause",
                "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
                "com.amazonaws.ecs.task-definition-family": "curltest",
                "com.amazonaws.ecs.task-definition-version": "26"
            },
            "DesiredStatus": "RESOURCES_PROVISIONED",
            "KnownStatus": "RESOURCES_PROVISIONED",
            "Limits": {
                "CPU": 0,
                "Memory": 0
            },
            "CreatedAt": "2020-10-02T00:43:05.602352471Z",
            "StartedAt": "2020-10-02T00:43:06.076707576Z",
            "Type": "CNI_PAUSE",
            "Networks": [
                {
                    "NetworkMode": "awsvpc",
                    "IPv4Addresses": [
                        "10.0.2.61"
                    ],
```

```
                    "AttachmentIndex": 0,
                    "MACAddress": "0e:10:e2:01:bd:91",
                    "IPv4SubnetCIDRBlock": "10.0.2.0/24",
                    "PrivateDNSName": "ip-10-0-2-61.us-west-2.compute.internal",
                    "SubnetGatewayIpv4Address": "10.0.2.1/24"
                }
            ]
        },
        {
            "DockerId":
 "ee08638adaaf009d78c248913f629e38299471d45fe7dc944d1039077e3424ca",
            "Name": "curl",
            "DockerName": "ecs-curltest-26-curl-a0e7dba5aca6d8cb2e00",
            "Image": "111122223333.dkr.ecr.us-west-2.amazonaws.com/curltest:latest",
            "ImageID":
 "sha256:d691691e9652791a60114e67b365688d20d19940dde7c4736ea30e660d8d3553",
            "Labels": {
                "com.amazonaws.ecs.cluster": "default",
                "com.amazonaws.ecs.container-name": "curl",
                "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
                "com.amazonaws.ecs.task-definition-family": "curltest",
                "com.amazonaws.ecs.task-definition-version": "26"
            },
            "DesiredStatus": "RUNNING",
            "KnownStatus": "RUNNING",
            "Limits": {
                "CPU": 10,
                "Memory": 128
            },
            "CreatedAt": "2020-10-02T00:43:06.326590752Z",
            "StartedAt": "2020-10-02T00:43:06.767535449Z",
            "Type": "NORMAL",
            "LogDriver": "awslogs",
            "LogOptions": {
                "awslogs-create-group": "true",
                "awslogs-group": "/ecs/metadata",
                "awslogs-region": "us-west-2",
                "awslogs-stream": "ecs/curl/158d1c8083dd49d6b527399fd6414f5c"
            },
            "ContainerARN": "arn:aws:ecs:us-west-2:111122223333:container/
abb51bdd-11b4-467f-8f6c-adcfe1fe059d",
            "Networks": [
                {
```

```
                    "NetworkMode": "awsvpc",
                    "IPv4Addresses": [
                        "10.0.2.61"
                    ],
                    "AttachmentIndex": 0,
                    "MACAddress": "0e:10:e2:01:bd:91",
                    "IPv4SubnetCIDRBlock": "10.0.2.0/24",
                    "PrivateDNSName": "ip-10-0-2-61.us-west-2.compute.internal",
                    "SubnetGatewayIpv4Address": "10.0.2.1/24"
                }
            ]
        }
    ]
}
```

コンテナ統計レスポンスの例

${ECS_CONTAINER_METADATA_URI_V4}/stats エンドポイントにクエリを実行すると、コンテナのネットワークメトリクスが返されます。少なくともバージョン 1.43.0 のコンテナエージェントを実行しているAmazon EC2 インスタンスでホストされている awsvpc または bridge ネットワークモードを使用するAmazon ECS タスクの場合、応答に追加のネットワークレート統計が含まれます。他のすべてのタスクでは、レスポンスには累積ネットワーク統計のみが含まれます。

次に、bridge ネットワークモードを使用する Amazon ECS の Amazon EC2 タスクからの出力例を示します。

```
{
    "read": "2020-10-02T00:51:13.410254284Z",
    "preread": "2020-10-02T00:51:12.406202398Z",
    "pids_stats": {
        "current": 3
    },
    "blkio_stats": {
        "io_service_bytes_recursive": [

        ],
        "io_serviced_recursive": [

        ],
        "io_queue_recursive": [

        ],
        "io_service_time_recursive": [
```

```
        ],
        "io_wait_time_recursive": [

        ],
        "io_merged_recursive": [

        ],
        "io_time_recursive": [

        ],
        "sectors_recursive": [

        ]
    },
    "num_procs": 0,
    "storage_stats": {

    },
    "cpu_stats": {
        "cpu_usage": {
            "total_usage": 360968065,
            "percpu_usage": [
                182359190,
                178608875
            ],
            "usage_in_kernelmode": 40000000,
            "usage_in_usermode": 290000000
        },
        "system_cpu_usage": 13939680000000,
        "online_cpus": 2,
        "throttling_data": {
            "periods": 0,
            "throttled_periods": 0,
            "throttled_time": 0
        }
    },
    "precpu_stats": {
        "cpu_usage": {
            "total_usage": 360968065,
            "percpu_usage": [
                182359190,
                178608875
            ],
```

```
            "usage_in_kernelmode": 40000000,
            "usage_in_usermode": 290000000
        },
        "system_cpu_usage": 13937670000000,
        "online_cpus": 2,
        "throttling_data": {
            "periods": 0,
            "throttled_periods": 0,
            "throttled_time": 0
        }
    },
    "memory_stats": {
        "usage": 1806336,
        "max_usage": 6299648,
        "stats": {
            "active_anon": 606208,
            "active_file": 0,
            "cache": 0,
            "dirty": 0,
            "hierarchical_memory_limit": 134217728,
            "hierarchical_memsw_limit": 268435456,
            "inactive_anon": 0,
            "inactive_file": 0,
            "mapped_file": 0,
            "pgfault": 4185,
            "pgmajfault": 0,
            "pgpgin": 2926,
            "pgpgout": 2778,
            "rss": 606208,
            "rss_huge": 0,
            "total_active_anon": 606208,
            "total_active_file": 0,
            "total_cache": 0,
            "total_dirty": 0,
            "total_inactive_anon": 0,
            "total_inactive_file": 0,
            "total_mapped_file": 0,
            "total_pgfault": 4185,
            "total_pgmajfault": 0,
            "total_pgpgin": 2926,
            "total_pgpgout": 2778,
            "total_rss": 606208,
            "total_rss_huge": 0,
            "total_unevictable": 0,
```

```
            "total_writeback": 0,
            "unevictable": 0,
            "writeback": 0
        },
        "limit": 134217728
    },
    "name": "/ecs-curltest-26-curl-c2e5f6e0cf91b0bead01",
    "id": "5fc21e5b015f899d22618f8aede80b6d70d71b2a75465ea49d9462c8f3d2d3af",
    "networks": {
        "eth0": {
            "rx_bytes": 84,
            "rx_packets": 2,
            "rx_errors": 0,
            "rx_dropped": 0,
            "tx_bytes": 84,
            "tx_packets": 2,
            "tx_errors": 0,
            "tx_dropped": 0
        }
    },
    "network_rate_stats": {
        "rx_bytes_per_sec": 0,
        "tx_bytes_per_sec": 0
    }
}
```

タスク統計情報のレスポンス例

${ECS_CONTAINER_METADATA_URI_V4}/task/stats エンドポイントにクエリを実行すると、コンテナが属しているタスクに関するネットワークメトリクスが返されます。以下に出力例を示します。

```
{
    "01999f2e5c6cf4df3873f28950e6278813408f281c54778efec860d0caad4854": {
        "read": "2020-10-02T00:51:32.51467703Z",
        "preread": "2020-10-02T00:51:31.50860463Z",
        "pids_stats": {
            "current": 1
        },
        "blkio_stats": {
            "io_service_bytes_recursive": [

            ],
```

```
            "io_serviced_recursive": [

            ],
            "io_queue_recursive": [

            ],
            "io_service_time_recursive": [

            ],
            "io_wait_time_recursive": [

            ],
            "io_merged_recursive": [

            ],
            "io_time_recursive": [

            ],
            "sectors_recursive": [

            ]
        },
        "num_procs": 0,
        "storage_stats": {

        },
        "cpu_stats": {
            "cpu_usage": {
                "total_usage": 177232665,
                "percpu_usage": [
                    13376224,
                    163856441
                ],
                "usage_in_kernelmode": 0,
                "usage_in_usermode": 160000000
            },
            "system_cpu_usage": 13977820000000,
            "online_cpus": 2,
            "throttling_data": {
                "periods": 0,
                "throttled_periods": 0,
                "throttled_time": 0
            }
        },
```

```
        "precpu_stats": {
            "cpu_usage": {
                "total_usage": 177232665,
                "percpu_usage": [
                    13376224,
                    163856441
                ],
                "usage_in_kernelmode": 0,
                "usage_in_usermode": 160000000
            },
            "system_cpu_usage": 13975800000000,
            "online_cpus": 2,
            "throttling_data": {
                "periods": 0,
                "throttled_periods": 0,
                "throttled_time": 0
            }
        },
        "memory_stats": {
            "usage": 532480,
            "max_usage": 6279168,
            "stats": {
                "active_anon": 40960,
                "active_file": 0,
                "cache": 0,
                "dirty": 0,
                "hierarchical_memory_limit": 9223372036854771712,
                "hierarchical_memsw_limit": 9223372036854771712,
                "inactive_anon": 0,
                "inactive_file": 0,
                "mapped_file": 0,
                "pgfault": 2033,
                "pgmajfault": 0,
                "pgpgin": 1734,
                "pgpgout": 1724,
                "rss": 40960,
                "rss_huge": 0,
                "total_active_anon": 40960,
                "total_active_file": 0,
                "total_cache": 0,
                "total_dirty": 0,
                "total_inactive_anon": 0,
                "total_inactive_file": 0,
                "total_mapped_file": 0,
```

```
                "total_pgfault": 2033,
                "total_pgmajfault": 0,
                "total_pgpgin": 1734,
                "total_pgpgout": 1724,
                "total_rss": 40960,
                "total_rss_huge": 0,
                "total_unevictable": 0,
                "total_writeback": 0,
                "unevictable": 0,
                "writeback": 0
            },
            "limit": 4073377792
        },
        "name": "/ecs-curltest-26-internalecspause-a6bcc3dbadfacfe85300",
        "id": "01999f2e5c6cf4df3873f28950e6278813408f281c54778efec860d0caad4854",
        "networks": {
            "eth0": {
                "rx_bytes": 84,
                "rx_packets": 2,
                "rx_errors": 0,
                "rx_dropped": 0,
                "tx_bytes": 84,
                "tx_packets": 2,
                "tx_errors": 0,
                "tx_dropped": 0
            }
        },
        "network_rate_stats": {
            "rx_bytes_per_sec": 0,
            "tx_bytes_per_sec": 0
        }
    },
    "5fc21e5b015f899d22618f8aede80b6d70d71b2a75465ea49d9462c8f3d2d3af": {
        "read": "2020-10-02T00:51:32.512771349Z",
        "preread": "2020-10-02T00:51:31.510597736Z",
        "pids_stats": {
            "current": 3
        },
        "blkio_stats": {
            "io_service_bytes_recursive": [

            ],
            "io_serviced_recursive": [
```

```
            ],
            "io_queue_recursive": [

            ],
            "io_service_time_recursive": [

            ],
            "io_wait_time_recursive": [

            ],
            "io_merged_recursive": [

            ],
            "io_time_recursive": [

            ],
            "sectors_recursive": [

            ]
        },
        "num_procs": 0,
        "storage_stats": {

        },
        "cpu_stats": {
            "cpu_usage": {
                "total_usage": 379075681,
                "percpu_usage": [
                    191355275,
                    187720406
                ],
                "usage_in_kernelmode": 60000000,
                "usage_in_usermode": 310000000
            },
            "system_cpu_usage": 13977800000000,
            "online_cpus": 2,
            "throttling_data": {
                "periods": 0,
                "throttled_periods": 0,
                "throttled_time": 0
            }
        },
        "precpu_stats": {
            "cpu_usage": {
```

```
                "total_usage": 378825197,
                "percpu_usage": [
                    191104791,
                    187720406
                ],
                "usage_in_kernelmode": 60000000,
                "usage_in_usermode": 310000000
            },
            "system_cpu_usage": 13975800000000,
            "online_cpus": 2,
            "throttling_data": {
                "periods": 0,
                "throttled_periods": 0,
                "throttled_time": 0
            }
        },
        "memory_stats": {
            "usage": 1814528,
            "max_usage": 6299648,
            "stats": {
                "active_anon": 606208,
                "active_file": 0,
                "cache": 0,
                "dirty": 0,
                "hierarchical_memory_limit": 134217728,
                "hierarchical_memsw_limit": 268435456,
                "inactive_anon": 0,
                "inactive_file": 0,
                "mapped_file": 0,
                "pgfault": 5377,
                "pgmajfault": 0,
                "pgpgin": 3613,
                "pgpgout": 3465,
                "rss": 606208,
                "rss_huge": 0,
                "total_active_anon": 606208,
                "total_active_file": 0,
                "total_cache": 0,
                "total_dirty": 0,
                "total_inactive_anon": 0,
                "total_inactive_file": 0,
                "total_mapped_file": 0,
                "total_pgfault": 5377,
                "total_pgmajfault": 0,
```

```
                "total_pgpgin": 3613,
                "total_pgpgout": 3465,
                "total_rss": 606208,
                "total_rss_huge": 0,
                "total_unevictable": 0,
                "total_writeback": 0,
                "unevictable": 0,
                "writeback": 0
            },
            "limit": 134217728
        },
        "name": "/ecs-curltest-26-curl-c2e5f6e0cf91b0bead01",
        "id": "5fc21e5b015f899d22618f8aede80b6d70d71b2a75465ea49d9462c8f3d2d3af",
        "networks": {
            "eth0": {
                "rx_bytes": 84,
                "rx_packets": 2,
                "rx_errors": 0,
                "rx_dropped": 0,
                "tx_bytes": 84,
                "tx_packets": 2,
                "tx_errors": 0,
                "tx_dropped": 0
            }
        },
        "network_rate_stats": {
            "rx_bytes_per_sec": 0,
            "tx_bytes_per_sec": 0
        }
    }
}
```


Amazon ECS タスクメタデータエンドポイントバージョン 3

> **Important**
>
> タスクメタデータのバージョン 3 エンドポイントは、現在アクティブにメンテナンスされていません。タスクメタデータバージョン 4 エンドポイントを更新して、最新のメタデータエンドポイント情報を取得することをお勧めします。詳細については、「the section called “タスクメタデータエンドポイントバージョン 4”」を参照してください。

AWS Fargate でホストされている Amazon ECS タスクを使用している場合は、「Fargate 上のタスク用の Amazon ECS タスクのメタデータエンドポイントバージョン 3」を参照してください。

Amazon ECS コンテナエージェントの 1.21.0 バージョン以降では、エージェントはタスクの各コンテナに、環境変数 ECS_CONTAINER_METADATA_URI を挿入します。タスクメタデータバージョン 3 エンドポイントに対してクエリを実行すると、さまざまなタスクメタデータおよび Docker 統計 をタスクで利用できます。bridge ネットワークモードを使用するタスクの場合、/stats エンドポイントにクエリを実行する際にネットワークメトリクスを使用できます。

タスクメタデータエンドポイントバージョン 3 の機能は、プラットフォームバージョン v1.3.0 以降で Fargate を使用するタスクと、EC2 を使用し、Amazon ECS コンテナエージェントのバージョン 1.21.0 以降を実行する Amazon EC2 Linux インフラストラクチャで起動されるタスク、または、Amazon ECS コンテナエージェントのバージョン 1.54.0 以降を実行する Amazon EC2 Windows インフラストラクチャで起動され、awsvpc ネットワークモードを使用するタスクは、デフォルトで有効になります。詳細については、「Amazon ECS Linux コンテナインスタンスの管理」を参照してください。

エージェントを最新バージョンに更新することで、古いコンテナインスタンスにこの機能のサポートを追加できます。詳細については、「Amazon ECS コンテナエージェントをアップデートする」を参照してください。

Important

Fargate およびプラットフォームバージョン v1.3.0 以前を使用するタスクでは、タスクメタデータバージョン 2 エンドポイントはサポートされています。詳細については、「Amazon ECS タスクメタデータエンドポイントバージョン 2」を参照してください。

タスクメタデータエンドポイントバージョン 3 パス

次のタスクメタデータエンドポイントをコンテナで使用できます。

```
${ECS_CONTAINER_METADATA_URI}
```

このパスはコンテナのメタデータ JSON を返します。

`${ECS_CONTAINER_METADATA_URI}/task`

このパスはタスクのメタデータ JSON を返します。これには、タスクに関連付けられたすべてのコンテナのコンテナ ID および名前のリストが含まれています。このエンドポイントのレスポンスの詳細については、「Amazon ECS タスクメタデータ v3 JSON レスポンス」を参照してください。

`${ECS_CONTAINER_METADATA_URI}/taskWithTags`

このパスは、`/task` API を使用して取得できるタスクとコンテナインスタンスタグに加えて、`ListTagsForResource` エンドポイントに含まれるタスクのメタデータを返します。

`${ECS_CONTAINER_METADATA_URI}/stats`

このパスは特定の Docker コンテナの Docker 統計 JSON を返します。返される各統計の詳細については、Docker API ドキュメントの「ContainerStats」を参照してください。

`${ECS_CONTAINER_METADATA_URI}/task/stats`

このパスはタスクに関連付けられたすべてのコンテナの Docker 統計 JSON を返します。返される各統計の詳細については、Docker API ドキュメントの「ContainerStats」を参照してください。

Amazon ECS タスクメタデータ v3 JSON レスポンス

次の情報が、タスクメタデータエンドポイント (`${ECS_CONTAINER_METADATA_URI}/task`) JSON レスポンスから返されます。

`Cluster`

タスクが属する Amazon ECS クラスターの Amazon リソースネーム (ARN)または短縮名。

`TaskARN`

コンテナが属しているタスクの Amazon リソースネーム (ARN)。

`Family`

タスクの Amazon ECS タスク定義のファミリー。

`Revision`

タスクの Amazon ECS タスク定義のリビジョン。

`DesiredStatus`

Amazon ECSからのタスクの望ましいステータス。

`KnownStatus`

Amazon ECS からのタスクの既知のステータス。

`Limits`

CPU (vCPU で表される) やメモリなど、タスク レベルで指定されたリソースの制限。リソースの制限が定義されていない場合はこのパラメータは省略されます。

`PullStartedAt`

最初のコンテナイメージのプル開始時のタイムスタンプ。

`PullStoppedAt`

最後のコンテナイメージのプル終了時のタイムスタンプ。

`AvailabilityZone`

タスクがあるアベイラビリティーゾーン。

> **Note**
> アベイラビリティーゾーンのメタデータは、プラットフォームバージョン 1.4 以降 (Linux) または 1.0.0 以降 (Windows) を使用する Fargate タスクでのみ使用できます。

`Containers`

タスクに関連付けられている各コンテナのコンテナメタデータのリスト。

`DockerId`

コンテナの Docker ID。

`Name`

タスク定義で指定されたコンテナの名前。

`DockerName`

Docker に提供されたコンテナの名前。Amazon ECSコンテナエージェントはコンテナに一意の名前を生成し、同じタスク定義の複数のコピーが単一のインスタンスで実行される場合に名前の競合を回避します。

`Image`

コンテナのイメージ。

`ImageID`

イメージマニフェストの SHA-256 ダイジェスト。これは、`repository-url/image@sha256:digest` 形式を使用してイメージをプルするために使用できるダイジェストです。

`Ports`

コンテナに対して公開されている任意のポート。公開ポートがない場合はこのパラメータは省略されます。

`Labels`

コンテナに適用された任意のラベル。ラベルが適用されていない場合はこのパラメータは省略されます。

`DesiredStatus`

Amazon ECSからのコンテナの望ましいステータス。

`KnownStatus`

Amazon ECS からのコンテナの既知のステータス。

`ExitCode`

コンテナの終了コード。このパラメータは、コンテナが終了していない場合は省略されます。

`Limits`

CPU (CPU 単位で表される) やメモリなど、コンテナレベルで指定されたリソースの制限。リソースの制限が定義されていない場合はこのパラメータは省略されます。

`CreatedAt`

コンテナ作成時のタイムスタンプ。このパラメータは、コンテナがまだ作成されていない場合は省略されます。

`StartedAt`

コンテナ起動時のタイムスタンプ。このパラメータは、コンテナがまだ起動していない場合は省略されます。

`FinishedAt`

コンテナ停止時のタイムスタンプ。このパラメータは、コンテナがまだ停止していない場合は省略されます。

Type

コンテナのタイプ。タスク定義で指定されているコンテナのタイプは NORMAL です。他のコンテナタイプは無視してかまいません。これらはAmazon ECS コンテナエージェントによってプロビジョニングされる内部タスクリソースで使用されます。

Networks

ネットワークモードや IP アドレスなど、コンテナのネットワーク情報。ネットワーク情報が定義されていない場合はこのパラメータは省略されます。

ClockDrift

基準時刻とシステム時刻の違いに関する情報。これは、Linux オペレーティングシステムに適用されます。この機能では、Amazon Time Sync Service を使用してクロックの精度を測定し、コンテナのクロックエラー範囲を特定します。詳細については、「Linux インスタンス用 Amazon EC2 ユーザーガイド」の「Linux インスタンスの時刻の設定」を参照してください。

ReferenceTime

クロック精度の基礎。Amazon ECS は、2021-09-07T16:57:44Z などの NTP を通じて協定世界時 (UTC) の世界標準を使用しています。

ClockErrorBound

UTC へのオフセットとして定義されるクロックエラーの対策。この誤差は、基準時刻とシステム時刻の差 (ミリ秒単位) です。

ClockSynchronizationStatus

最新のシステム時刻と基準時刻間の同期試行における成功状況を示します。

有効な値は SYNCHRONIZED および NOT_SYNCHRONIZED です。

ExecutionStoppedAt

タスクの DesiredStatus が STOPPED に移行した時のタイムスタンプ。これは、必須コンテナが STOPPED に移行したときに発生します。

Amazon ECS タスクメタデータ v3 の例

以下の例は、タスクメタデータエンドポイントからの出力を示しています。

コンテナメタデータレスポンスの例

${ECS_CONTAINER_METADATA_URI} エンドポイントにクエリを実行すると、コンテナ自体に関するメタデータのみが返されます。以下に出力例を示します。

```
{
    "DockerId": "43481a6ce4842eec8fe72fc28500c6b52edcc0917f105b83379f88cac1ff3946",
    "Name": "nginx-curl",
    "DockerName": "ecs-nginx-5-nginx-curl-ccccb9f49db0dfe0d901",
    "Image": "nrdlngr/nginx-curl",
    "ImageID":
 "sha256:2e00ae64383cfc865ba0a2ba37f61b50a120d2d9378559dcd458dc0de47bc165",
    "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "nginx-curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
        "com.amazonaws.ecs.task-definition-family": "nginx",
        "com.amazonaws.ecs.task-definition-version": "5"
    },
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "Limits": {
        "CPU": 512,
        "Memory": 512
    },
    "CreatedAt": "2018-02-01T20:55:10.554941919Z",
    "StartedAt": "2018-02-01T20:55:11.064236631Z",
    "Type": "NORMAL",
    "Networks": [
        {
            "NetworkMode": "awsvpc",
            "IPv4Addresses": [
                "10.0.2.106"
            ]
        }
    ]
}
```

タスクメタデータレスポンスの例

${ECS_CONTAINER_METADATA_URI}/task エンドポイントにクエリを実行すると、コンテナが属しているタスクに関するメタデータが返されます。以下に出力例を示します。

1 つのコンテナタスクの JSON レスポンスを次に示します。

```
{
  "Cluster": "default",
  "TaskARN": "arn:aws:ecs:us-east-2:012345678910:task/9781c248-0edd-4cdb-9a93-
f63cb662a5d3",
  "Family": "nginx",
  "Revision": "5",
  "DesiredStatus": "RUNNING",
  "KnownStatus": "RUNNING",
  "Containers": [
    {
      "DockerId": "731a0d6a3b4210e2448339bc7015aaa79bfe4fa256384f4102db86ef94cbbc4c",
      "Name": "~internal~ecs~pause",
      "DockerName": "ecs-nginx-5-internalecspause-acc699c0cbf2d6d11700",
      "Image": "amazon/amazon-ecs-pause:0.1.0",
      "ImageID": "",
      "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "~internal~ecs~pause",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
        "com.amazonaws.ecs.task-definition-family": "nginx",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
      "DesiredStatus": "RESOURCES_PROVISIONED",
      "KnownStatus": "RESOURCES_PROVISIONED",
      "Limits": {
        "CPU": 0,
        "Memory": 0
      },
      "CreatedAt": "2018-02-01T20:55:08.366329616Z",
      "StartedAt": "2018-02-01T20:55:09.058354915Z",
      "Type": "CNI_PAUSE",
      "Networks": [
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": [
            "10.0.2.106"
          ]
        }
      ]
    },
    {
```

```
      "DockerId": "43481a6ce4842eec8fe72fc28500c6b52edcc0917f105b83379f88cac1ff3946",
      "Name": "nginx-curl",
      "DockerName": "ecs-nginx-5-nginx-curl-ccccb9f49db0dfe0d901",
      "Image": "nrdlngr/nginx-curl",
      "ImageID":
 "sha256:2e00ae64383cfc865ba0a2ba37f61b50a120d2d9378559dcd458dc0de47bc165",
      "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "nginx-curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
        "com.amazonaws.ecs.task-definition-family": "nginx",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
      "DesiredStatus": "RUNNING",
      "KnownStatus": "RUNNING",
      "Limits": {
        "CPU": 512,
        "Memory": 512
      },
      "CreatedAt": "2018-02-01T20:55:10.554941919Z",
      "StartedAt": "2018-02-01T20:55:11.064236631Z",
      "Type": "NORMAL",
      "Networks": [
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": [
            "10.0.2.106"
          ]
        }
      ]
    }
  ],
  "PullStartedAt": "2018-02-01T20:55:09.372495529Z",
  "PullStoppedAt": "2018-02-01T20:55:10.552018345Z",
  "AvailabilityZone": "us-east-2b"
}
```

Amazon ECS タスクメタデータエンドポイントバージョン 2

> **Important**
>
> タスクメタデータのバージョン 2 エンドポイントは、現在アクティブにメンテナンスされていません。タスクメタデータバージョン 4 エンドポイントを更新して、最新のメタデータエンドポイント情報を取得することをお勧めします。詳細については、「the section called “タスクメタデータエンドポイントバージョン 4”」を参照してください。

Amazon ECS コンテナエージェントのバージョン 1.17.0 から、さまざまなタスクメタデータおよび [Docker 統計]を、Amazon ECS コンテナエージェントによって指定される HTTP エンドポイントで awsvpc ネットワークモードを使用するタスクで利用できます。

awsvpc ネットワークモードで起動されたタスクに属するすべてのコンテナには、事前定義されたリンクローカルアドレス範囲内のローカル IPv4 アドレスが割り当てられます。コンテナがメタデータエンドポイントにクエリを実行する場合、Amazon ECS コンテナエージェントは、一意の IP アドレスに基づいてコンテナが属するタスクを判断し、そのタスクのメタデータと統計情報を返します。

タスクメタデータの有効化

> **Important**
>
> タスクメタデータのバージョン 2 エンドポイントは、現在アクティブにメンテナンスされていません。タスクメタデータバージョン 4 エンドポイントを更新して、最新のメタデータエンドポイント情報を取得することをお勧めします。詳細については、「the section called “タスクメタデータエンドポイントバージョン 4”」を参照してください。

Amazon ECS コンテナエージェントのバージョン 1.17.0 から、さまざまなタスクメタデータおよび [Docker 統計]を、Amazon ECS コンテナエージェントによって指定される HTTP エンドポイントで awsvpc ネットワークモードを使用するタスクで利用できます。

awsvpc ネットワークモードで起動されたタスクに属するすべてのコンテナには、事前定義されたリンクローカルアドレス範囲内のローカル IPv4 アドレスが割り当てられます。コンテナがメタデータエンドポイントにクエリを実行する場合、Amazon ECS コンテナエージェントは、一意の IP アドレスに基づいてコンテナが属するタスクを判断し、そのタスクのメタデータと統計情報を返します。

タスクメタデータの有効化

タスクメタデータバージョン 2 機能は、デフォルトで次のタスクに対して有効に設定されています。

- プラットフォームバージョンが v1.1.0 以降の Fargate を使用するタスク。詳細については、「Amazon ECS 向け Fargate プラットフォームバージョン」を参照してください。
- awsvpc ネットワークモードも使用し、Amazon ECS コンテナエージェントのバージョン 1.17.0 以上を実行している Amazon EC2 Linux インフラストラクチャ、または Amazon ECS コンテナエージェントのバージョン 1.54.0 以上を実行している Amazon EC2 Windows インフラストラクチャで起動される EC2 を使用するタスク。詳細については、「Amazon ECS Linux コンテナインスタンスの管理」を参照してください。

エージェントを最新バージョンに更新することで、古いコンテナインスタンスにこの機能のサポートを追加できます。詳細については、「Amazon ECS コンテナエージェントをアップデートする」を参照してください。

タスクメタデータエンドポイントのパス

次の API エンドポイントをコンテナで使用できます。

`169.254.170.2/v2/metadata`

このエンドポイントはタスクのメタデータ JSON を返します。これには、タスクに関連付けられたすべてのコンテナのコンテナ ID および名前のリストが含まれています。このエンドポイントのレスポンスの詳細については、「タスクメタデータ JSON レスポンス」を参照してください。

`169.254.170.2/v2/metadata/<container-id>`

このエンドポイントは指定された Docker コンテナ ID のメタデータ JSON を返します。

`169.254.170.2/v2/metadata/taskWithTags`

このパスは、/task API を使用して取得できるタスクとコンテナインスタンスタグに加えて、ListTagsForResource エンドポイントに含まれるタスクのメタデータを返します。

`169.254.170.2/v2/stats`

このエンドポイントはタスクに関連付けられたすべてのコンテナの Docker 統計 JSON を返します。返される各統計の詳細については、Docker API ドキュメントの「ContainerStats」を参照してください。

`169.254.170.2/v2/stats/<container-id>`

このエンドポイントは指定された Docker コンテナ ID の Docker 統計 JSON を返します。返される各統計の詳細については、Docker API ドキュメントの「ContainerStats」を参照してください。

タスクメタデータ JSON レスポンス

次の情報が、タスクメタデータエンドポイント (`169.254.170.2/v2/metadata`) JSON レスポンスから返されます。

`Cluster`

タスクが属する Amazon ECS クラスターの Amazon リソースネーム (ARN)または短縮名。

`TaskARN`

コンテナが属しているタスクの Amazon リソースネーム (ARN)。

`Family`

タスクの Amazon ECS タスク定義のファミリー。

`Revision`

タスクの Amazon ECS タスク定義のリビジョン。

`DesiredStatus`

Amazon ECSからのタスクの望ましいステータス。

`KnownStatus`

Amazon ECS からのタスクの既知のステータス。

`Limits`

CPU (vCPU で表される) やメモリなど、タスク レベルで指定されたリソースの制限。リソースの制限が定義されていない場合はこのパラメータは省略されます。

`PullStartedAt`

最初のコンテナイメージのプル開始時のタイムスタンプ。

`PullStoppedAt`

最後のコンテナイメージのプル終了時のタイムスタンプ。

`AvailabilityZone`

タスクがあるアベイラビリティーゾーン。

> **Note**
>
> アベイラビリティーゾーンのメタデータは、プラットフォームバージョン 1.4 以降 (Linux) または 1.0.0 以降 (Windows) を使用する Fargate タスクでのみ使用できます。

`Containers`

タスクに関連付けられている各コンテナのコンテナメタデータのリスト。

`DockerId`

コンテナの Docker ID。

`Name`

タスク定義で指定されたコンテナの名前。

`DockerName`

Docker に提供されたコンテナの名前。Amazon ECSコンテナエージェントはコンテナに一意の名前を生成し、同じタスク定義の複数のコピーが単一のインスタンスで実行される場合に名前の競合を回避します。

`Image`

コンテナのイメージ。

`ImageID`

イメージマニフェストの SHA-256 ダイジェスト。これは、`repository-url/image@sha256:digest` 形式を使用してイメージをプルするために使用できるダイジェストです。

`Ports`

コンテナに対して公開されている任意のポート。公開ポートがない場合はこのパラメータは省略されます。

`Labels`

コンテナに適用された任意のラベル。ラベルが適用されていない場合はこのパラメータは省略されます。

`DesiredStatus`

Amazon ECSからのコンテナの望ましいステータス。

`KnownStatus`

Amazon ECS からのコンテナの既知のステータス。

`ExitCode`

コンテナの終了コード。このパラメータは、コンテナが終了していない場合は省略されます。

`Limits`

CPU (CPU 単位で表される) やメモリなど、コンテナレベルで指定されたリソースの制限。リソースの制限が定義されていない場合はこのパラメータは省略されます。

`CreatedAt`

コンテナ作成時のタイムスタンプ。このパラメータは、コンテナがまだ作成されていない場合は省略されます。

`StartedAt`

コンテナ起動時のタイムスタンプ。このパラメータは、コンテナがまだ起動していない場合は省略されます。

`FinishedAt`

コンテナ停止時のタイムスタンプ。このパラメータは、コンテナがまだ停止していない場合は省略されます。

`Type`

コンテナのタイプ。タスク定義で指定されているコンテナのタイプは `NORMAL` です。他のコンテナタイプは無視してかまいません。これらはAmazon ECS コンテナエージェントによってプロビジョニングされる内部タスクリソースで使用されます。

`Networks`

ネットワークモードや IP アドレスなど、コンテナのネットワーク情報。ネットワーク情報が定義されていない場合はこのパラメータは省略されます。

`ClockDrift`

基準時刻とシステム時刻の違いに関する情報。これは、Linux オペレーティングシステムに適用されます。この機能では、Amazon Time Sync Service を使用してクロックの精度を測定し、コ

ンテナのクロックエラー範囲を特定します。詳細については、「Linux インスタンス用 Amazon EC2 ユーザーガイド」の「Linux インスタンスの時刻の設定」を参照してください。

`ReferenceTime`

クロック精度の基礎。Amazon ECS は、`2021-09-07T16:57:44Z` などの NTP を通じて協定世界時 (UTC) の世界標準を使用しています。

`ClockErrorBound`

UTC へのオフセットとして定義されるクロックエラーの対策。この誤差は、基準時刻とシステム時刻の差 (ミリ秒単位) です。

`ClockSynchronizationStatus`

最新のシステム時刻と基準時刻間の同期試行における成功状況を示します。

有効な値は `SYNCHRONIZED` および `NOT_SYNCHRONIZED` です。

`ExecutionStoppedAt`

タスクの `DesiredStatus` が `STOPPED` に移行した時のタイムスタンプ。これは、必須コンテナが `STOPPED` に移行したときに発生します。

タスクメタデータレスポンスの例

1 つのコンテナタスクの JSON レスポンスを次に示します。

```
{
  "Cluster": "default",
  "TaskARN": "arn:aws:ecs:us-east-2:012345678910:task/9781c248-0edd-4cdb-9a93-
f63cb662a5d3",
  "Family": "nginx",
  "Revision": "5",
  "DesiredStatus": "RUNNING",
  "KnownStatus": "RUNNING",
  "Containers": [
    {
      "DockerId": "731a0d6a3b4210e2448339bc7015aaa79bfe4fa256384f4102db86ef94cbbc4c",
      "Name": "~internal~ecs~pause",
      "DockerName": "ecs-nginx-5-internalecspause-acc699c0cbf2d6d11700",
      "Image": "amazon/amazon-ecs-pause:0.1.0",
      "ImageID": "",
      "Labels": {
        "com.amazonaws.ecs.cluster": "default",
```

```
        "com.amazonaws.ecs.container-name": "~internal~ecs~pause",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
        "com.amazonaws.ecs.task-definition-family": "nginx",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
      "DesiredStatus": "RESOURCES_PROVISIONED",
      "KnownStatus": "RESOURCES_PROVISIONED",
      "Limits": {
        "CPU": 0,
        "Memory": 0
      },
      "CreatedAt": "2018-02-01T20:55:08.366329616Z",
      "StartedAt": "2018-02-01T20:55:09.058354915Z",
      "Type": "CNI_PAUSE",
      "Networks": [
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": [
            "10.0.2.106"
          ]
        }
      ]
    },
    {
      "DockerId": "43481a6ce4842eec8fe72fc28500c6b52edcc0917f105b83379f88cac1ff3946",
      "Name": "nginx-curl",
      "DockerName": "ecs-nginx-5-nginx-curl-ccccb9f49db0dfe0d901",
      "Image": "nrdlngr/nginx-curl",
      "ImageID":
 "sha256:2e00ae64383cfc865ba0a2ba37f61b50a120d2d9378559dcd458dc0de47bc165",
      "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "nginx-curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
        "com.amazonaws.ecs.task-definition-family": "nginx",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
      "DesiredStatus": "RUNNING",
      "KnownStatus": "RUNNING",
      "Limits": {
        "CPU": 512,
        "Memory": 512
```

```
      },
      "CreatedAt": "2018-02-01T20:55:10.554941919Z",
      "StartedAt": "2018-02-01T20:55:11.064236631Z",
      "Type": "NORMAL",
      "Networks": [
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": [
            "10.0.2.106"
          ]
        }
      ]
    }
  ],
  "PullStartedAt": "2018-02-01T20:55:09.372495529Z",
  "PullStoppedAt": "2018-02-01T20:55:10.552018345Z",
  "AvailabilityZone": "us-east-2b"
}
```

Fargate のタスクで使用できる Amazon ECS タスクメタデータ

FargateのAmazon ECS は、コンテナとその一部であるタスクに関するさまざまなメタデータ、ネットワークメトリクス、および Docker 統計情報を取得する方法を提供します。これは、タスクメタデータエンドポイント と呼ばれます。Fargate タスクのAmazon ECSでは、次のタスクメタデータエンドポイントバージョンを使用できます:

- タスクメタデータエンドポイントバージョン 4 – プラットフォームバージョン 1.4.0 以降を使用するタスクで利用可能です。
- タスクメタデータエンドポイントバージョン 3 – プラットフォームバージョン 1.1.0 以降を使用するタスクで利用可能です。

awsvpc ネットワークモードで起動されたタスクに属するすべてのコンテナには、事前定義されたリンクローカルアドレス範囲内のローカル IPv4 アドレスが割り当てられます。コンテナがメタデータエンドポイントにクエリを実行する場合、コンテナエージェントは、一意の IP アドレスに基づいてコンテナが属するタスクを判断し、そのタスクのメタデータと統計情報を返します。

トピック

- Fargate のタスク用の Amazon ECS タスクメタデータエンドポイントバージョン 4
- Fargate のタスク用の Amazon ECS タスクメタデータエンドポイントバージョン 3

Fargate のタスク用の Amazon ECS タスクメタデータエンドポイントバージョン 4

> **Important**
>
> Amazon EC2 インスタンスでホストされている Amazon ECS タスクを使用している場合は、「Amazon ECS タスクのメタデータエンドポイント」を参照してください。

Fargate のプラットフォームバージョン `1.4.0` 以降、`ECS_CONTAINER_METADATA_URI_V4` という名前の環境変数がタスク内の各コンテナに挿入されます。タスクメタデータエンドポイントバージョン 4 に対してクエリを実行すると、さまざまなタスクメタデータおよび Docker 統計をタスクで利用できます。

タスクメタデータエンドポイントバージョン 4 は、バージョン 3 と同じように動作しますが、コンテナとタスクに関する追加のネットワークメタデータを提供します。`/stats` エンドポイントにクエリを実行するときにも追加のネットワークメトリクスを使用できます。

タスクメタデータエンドポイントは、プラットフォームバージョン `1.4.0` 以降を使用する AWS Fargate 上で実行される、すべての Amazon ECS タスクに対してデフォルトでオンになっています。

> **Note**
>
> 新しいタスクメタデータエンドポイントバージョンを今後作成する必要がないように、追加のメタデータをバージョン 4 の出力に追加できます。既存のメタデータが削除されたり、メタデータのフィールド名が変更されたりすることはありません。

Fargate タスクメタデータエンドポイントバージョン 4 のパス

次のタスクメタデータエンドポイントをコンテナで使用できます。

`${ECS_CONTAINER_METADATA_URI_V4}`

このパスはコンテナのメタデータを返します。

`${ECS_CONTAINER_METADATA_URI_V4}/task`

このパスはタスクのメタデータを返します。これには、タスクに関連付けられたすべてのコンテナのコンテナ ID および名前のリストが含まれています。このエンドポイントのレスポンスの詳

細については、「Fargate のタスク用の Amazon ECS タスクメタデータ v4 JSON レスポンス」を参照してください。

`${ECS_CONTAINER_METADATA_URI_V4}/stats`

このパスは Docker コンテナの Docker 統計を返します。返される各統計の詳細については、Docker API ドキュメントの「ContainerStats」を参照してください。

> **Note**
>
> AWS FargateでのAmazon ECS タスクは、コンテナの統計を返す前にコンテナを約 1 秒間実行する必要があります。

`${ECS_CONTAINER_METADATA_URI_V4}/task/stats`

このパスはタスクに関連付けられたすべてのコンテナの Docker 統計を返します。返される各統計の詳細については、Docker API ドキュメントの「ContainerStats」を参照してください。

> **Note**
>
> AWS FargateでのAmazon ECS タスクは、コンテナの統計を返す前にコンテナを約 1 秒間実行する必要があります。

Fargate のタスク用の Amazon ECS タスクメタデータ v4 JSON レスポンス

次のメタデータは、タスクメタデータエンドポイント (`${ECS_CONTAINER_METADATA_URI_V4}/task`) JSON レスポンスで返されます。

`Cluster`

タスクが属する Amazon ECS クラスターの Amazon リソースネーム (ARN)または短縮名。

`ServiceName`

タスクが属しているサービスの名前。ServiceName は、Fargate タスクがサービスに関連付けられている場合に表示されます。

`VPCID`

Amazon EC2 コンテナインスタンスの VPC ID。このフィールドは、Amazon EC2 インスタンスでのみ表示されます。

> **Note**
>
> この VPCID メタデータは、Amazon ECS コンテナエージェントのバージョン 1.63.1、もしくはそれ以降を使用している場合にのみ含まれます。

`TaskARN`

コンテナが属しているタスクの Amazon リソースネーム (ARN)。

`Family`

タスクの Amazon ECS タスク定義のファミリー。

`Revision`

タスクの Amazon ECS タスク定義のリビジョン。

`DesiredStatus`

Amazon ECSからのタスクの望ましいステータス。

`KnownStatus`

Amazon ECS からのタスクの既知のステータス。

`Limits`

CPU (vCPU で表される) やメモリなど、タスクレベルで指定されたリソースの制限。リソースの制限が定義されていない場合はこのパラメータは省略されます。

`PullStartedAt`

最初のコンテナイメージのプル開始時のタイムスタンプ。

`PullStoppedAt`

最後のコンテナイメージのプル終了時のタイムスタンプ。

`AvailabilityZone`

タスクがあるアベイラビリティーゾーン。

> **Note**
>
> アベイラビリティーゾーンのメタデータは、プラットフォームバージョン 1.4 以降 (Linux) または 1.0.0 (Windows) を使用する Fargate タスクでのみ使用できます。

`LaunchType`

タスクが使用している起動タイプ。クラスター容量プロバイダーを使用する場合、タスクが Fargate インフラストラクチャと EC2 インフラストラクチャのいずれを使用しているかを示します。

> **Note**
>
> このLaunchTypeメタデータは、Amazon ECS Linux コンテナエージェントのバージョン1.45.0 以降 (Linux) または 1.0.0 以降 (Windows) を使用している場合にのみ含まれます。

`EphemeralStorageMetrics`

このタスクのエフェメラルストレージの予約サイズと現在の使用量。

> **Note**
>
> Fargate はディスク上のスペースを予約します。スペースは Fargate によってのみ使用されます。これには課金されることはありません。これらのメトリクスには表示されません。ただし、この追加ストレージは、df などの他のツールでも確認できます。

`Utilized`

このタスクの現在のエフェメラルストレージ使用量 (MiB 単位)。

`Reserved`

このタスクの予約済みエフェメラルストレージ (MiB 単位)。エフェメラルストレージのサイズは、実行中のタスクでは変更できません。タスク定義で ephermalStorage オブジェクトを指定して、エフェメラルストレージの量を変更できます。ephermalStorage は MiB ではなく GiB 単位で指定されます。ephermalStorage および EphemeralStorageMetrics は、Fargate Linux プラットフォームのバージョン 1.4.0 以降でのみ使用できます。

`Containers`

タスクに関連付けられている各コンテナのコンテナメタデータのリスト。

`DockerId`

コンテナの Docker ID。

Fargate を使用する場合、IDは32桁の16進数に10桁の数字が続きます。

`Name`

タスク定義で指定されたコンテナの名前。

`DockerName`

Docker に提供されたコンテナの名前。Amazon ECSコンテナエージェントはコンテナに一意の名前を生成し、同じタスク定義の複数のコピーが単一のインスタンスで実行される場合に名前の競合を回避します。

`Image`

コンテナのイメージ。

`ImageID`

イメージマニフェストの SHA-256 ダイジェスト。これは、`repository-url/image@sha256:digest` 形式を使用してイメージをプルするために使用できるダイジェストです。

`Ports`

コンテナに対して公開されている任意のポート。公開ポートがない場合はこのパラメータは省略されます。

`Labels`

コンテナに適用された任意のラベル。ラベルが適用されていない場合はこのパラメータは省略されます。

`DesiredStatus`

Amazon ECSからのコンテナの望ましいステータス。

`KnownStatus`

Amazon ECS からのコンテナの既知のステータス。

`ExitCode`

コンテナの終了コード。このパラメータは、コンテナが終了していない場合は省略されます。

`Limits`

CPU (CPU 単位で表される) やメモリなど、コンテナレベルで指定されたリソースの制限。リソースの制限が定義されていない場合はこのパラメータは省略されます。

`CreatedAt`

コンテナ作成時のタイムスタンプ。このパラメータは、コンテナがまだ作成されていない場合は省略されます。

`StartedAt`

コンテナ起動時のタイムスタンプ。このパラメータは、コンテナがまだ起動していない場合は省略されます。

`FinishedAt`

コンテナ停止時のタイムスタンプ。このパラメータは、コンテナがまだ停止していない場合は省略されます。

`Type`

コンテナのタイプ。タスク定義で指定されているコンテナのタイプは `NORMAL` です。他のコンテナタイプは無視してかまいません。これらはAmazon ECS コンテナエージェントによってプロビジョニングされる内部タスクリソースで使用されます。

`LogDriver`

コンテナに使用されるログドライバー。

> **Note**
>
> この`LogDriver`メタデータは、Amazon ECS Linux コンテナエージェントのバージョン`1.45.0`、もしくはそれ以降を使用している場合にのみ含まれます。

`LogOptions`

コンテナに定義されたログドライバオプション。

> **Note**
>
> この`LogOptions`メタデータは、Amazon ECS Linux コンテナエージェントのバージョン`1.45.0`、もしくはそれ以降を使用している場合にのみ含まれます。

`ContainerARN`

コンテナの Amazon リソースネーム (ARN)。

> **Note**
>
> このContainerARNメタデータは、Amazon ECS Linux コンテナエージェントのバージョン1.45.0、もしくはそれ以降を使用している場合にのみ含まれます。

`Networks`

ネットワークモードや IP アドレスなど、コンテナのネットワーク情報。ネットワーク情報が定義されていない場合はこのパラメータは省略されます。

`Snapshotter`

このコンテナイメージをダウンロードするために containerd によって使用された snapshotter。有効な値はデフォルトの `overlayfs` です。`soci` は SOCI インデックスによる遅延読み込み時に使用されます。このパラメータは、Linux プラットフォームバージョン `1.4.0` で実行されるタスクにのみ使用できます。

`RestartCount`

コンテナが再起動された回数。

> **Note**
>
> `RestartCount` メタデータが含まれるのは、コンテナの再起動ポリシーが有効になっている場合のみです。詳細については、「コンテナ再起動ポリシーを使用して Amazon ECS タスク内の個々のコンテナを再起動する」を参照してください。

`ClockDrift`

基準時刻とシステム時刻の違いに関する情報。この機能では、Amazon Time Sync Service を使用してクロックの精度を測定し、コンテナのクロックエラー範囲を特定します。詳細については、「Linux インスタンス用 Amazon EC2 ユーザーガイド」の「Linux インスタンスの時刻の設定」を参照してください。

`ReferenceTime`

クロック精度の基礎。Amazon ECS は、`2021-09-07T16:57:44Z` などの NTP を通じて協定世界時 (UTC) の世界標準を使用しています。

ClockErrorBound

UTC へのオフセットとして定義されるクロックエラーの対策。この誤差は、基準時刻とシステム時刻の差 (ミリ秒単位) です。

ClockSynchronizationStatus

最新のシステム時刻と基準時刻間の同期試行における成功状況を示します。

有効な値は SYNCHRONIZED および NOT_SYNCHRONIZED です。

ExecutionStoppedAt

タスクの DesiredStatus が STOPPED に移行した時のタイムスタンプ。これは、必須コンテナが STOPPED に移行したときに発生します。

Fargate でのタスクの Amazon ECS タスクメタデータ v4 の例

以下の例は、AWS Fargateで実行されるAmazon ECS タスクのタスクメタデータエンドポイントからの出力例を示しています。

コンテナから、タスクメタデータのエンドポイントに続けて curl を使用すると、curl ${ECS_CONTAINER_METADATA_URI_V4}/task などのエンドポイントをクエリできます。

コンテナメタデータレスポンスの例

${ECS_CONTAINER_METADATA_URI_V4} エンドポイントにクエリを実行すると、コンテナ自体に関するメタデータのみが返されます。以下に出力例を示します。

```
{
    "DockerId": "cd189a933e5849daa93386466019ab50-2495160603",
    "Name": "curl",
    "DockerName": "curl",
    "Image": "111122223333.dkr.ecr.us-west-2.amazonaws.com/curltest:latest",
    "ImageID":
 "sha256:25f3695bedfb454a50f12d127839a68ad3caf91e451c1da073db34c542c4d2cb",
    "Labels": {
        "com.amazonaws.ecs.cluster": "arn:aws:ecs:us-west-2:111122223333:cluster/
default",
        "com.amazonaws.ecs.container-name": "curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/default/
cd189a933e5849daa93386466019ab50",
        "com.amazonaws.ecs.task-definition-family": "curltest",
        "com.amazonaws.ecs.task-definition-version": "2"
```

```
    },
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "Limits": {
        "CPU": 10,
        "Memory": 128
    },
    "CreatedAt": "2020-10-08T20:09:11.44527186Z",
    "StartedAt": "2020-10-08T20:09:11.44527186Z",
    "Type": "NORMAL",
    "Networks": [
        {
            "NetworkMode": "awsvpc",
            "IPv4Addresses": [
                "192.0.2.3"
            ],
            "AttachmentIndex": 0,
            "MACAddress": "0a:de:f6:10:51:e5",
            "IPv4SubnetCIDRBlock": "192.0.2.0/24",
            "DomainNameServers": [
                "192.0.2.2"
            ],
            "DomainNameSearchList": [
                "us-west-2.compute.internal"
            ],
            "PrivateDNSName": "ip-10-0-0-222.us-west-2.compute.internal",
            "SubnetGatewayIpv4Address": "192.0.2.0/24"
        }
    ],
    "ContainerARN": "arn:aws:ecs:us-west-2:111122223333:container/05966557-
f16c-49cb-9352-24b3a0dcd0e1",
    "LogOptions": {
        "awslogs-create-group": "true",
        "awslogs-group": "/ecs/containerlogs",
        "awslogs-region": "us-west-2",
        "awslogs-stream": "ecs/curl/cd189a933e5849daa93386466019ab50"
    },
    "LogDriver": "awslogs",
    "Snapshotter": "overlayfs"
}
```

Fargate でのタスクの Amazon ECS タスクメタデータ v4 の例

${ECS_CONTAINER_METADATA_URI_V4}/task エンドポイントにクエリを実行すると、コンテナが属しているタスクに関するメタデータが返されます。以下に出力例を示します。


```
{
  "Cluster": "arn:aws:ecs:us-east-1:123456789012:cluster/MyEmptyCluster",
  "TaskARN": "arn:aws:ecs:us-east-1:123456789012:task/MyEmptyCluster/
bfa2636268144d039771334145e490c5",
  "Family": "sample-fargate",
  "Revision": "5",
  "DesiredStatus": "RUNNING",
  "KnownStatus": "RUNNING",
  "Limits": {
    "CPU": 0.25,
    "Memory": 512
  },
  "PullStartedAt": "2023-07-21T15:45:33.532811081Z",
  "PullStoppedAt": "2023-07-21T15:45:38.541068435Z",
  "AvailabilityZone": "us-east-1d",
  "Containers": [
    {
      "DockerId": "bfa2636268144d039771334145e490c5-1117626119",
      "Name": "curl-image",
      "DockerName": "curl-image",
      "Image": "curlimages/curl",
      "ImageID":
 "sha256:daf3f46a2639c1613b25e85c9ee4193af8a1d538f92483d67f9a3d7f21721827",
      "Labels": {
        "com.amazonaws.ecs.cluster": "arn:aws:ecs:us-east-1:123456789012:cluster/
MyEmptyCluster",
        "com.amazonaws.ecs.container-name": "curl-image",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-east-1:123456789012:task/
MyEmptyCluster/bfa2636268144d039771334145e490c5",
        "com.amazonaws.ecs.task-definition-family": "sample-fargate",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
      "DesiredStatus": "RUNNING",
      "KnownStatus": "RUNNING",
      "Limits": { "CPU": 128 },
      "CreatedAt": "2023-07-21T15:45:44.91368314Z",
      "StartedAt": "2023-07-21T15:45:44.91368314Z",
      "Type": "NORMAL",
      "Networks": [
```

```
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": ["172.31.42.189"],
          "AttachmentIndex": 0,
          "MACAddress": "0e:98:9f:33:76:d3",
          "IPv4SubnetCIDRBlock": "172.31.32.0/20",
          "DomainNameServers": ["172.31.0.2"],
          "DomainNameSearchList": ["ec2.internal"],
          "PrivateDNSName": "ip-172-31-42-189.ec2.internal",
          "SubnetGatewayIpv4Address": "172.31.32.1/20"
        }
      ],
      "ContainerARN": "arn:aws:ecs:us-east-1:123456789012:container/MyEmptyCluster/
bfa2636268144d039771334145e490c5/da6cccf7-1178-400c-afdf-7536173ee209",
      "Snapshotter": "overlayfs"
    },
    {
      "DockerId": "bfa2636268144d039771334145e490c5-3681984407",
      "Name": "fargate-app",
      "DockerName": "fargate-app",
      "Image": "public.ecr.aws/docker/library/httpd:latest",
      "ImageID":
 "sha256:8059bdd0058510c03ae4c808de8c4fd2c1f3c1b6d9ea75487f1e5caa5ececa02",
      "Labels": {
        "com.amazonaws.ecs.cluster": "arn:aws:ecs:us-east-1:123456789012:cluster/
MyEmptyCluster",
        "com.amazonaws.ecs.container-name": "fargate-app",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-east-1:123456789012:task/
MyEmptyCluster/bfa2636268144d039771334145e490c5",
        "com.amazonaws.ecs.task-definition-family": "sample-fargate",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
      "DesiredStatus": "RUNNING",
      "KnownStatus": "RUNNING",
      "Limits": { "CPU": 2 },
      "CreatedAt": "2023-07-21T15:45:44.954460255Z",
      "StartedAt": "2023-07-21T15:45:44.954460255Z",
      "Type": "NORMAL",
      "Networks": [
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": ["172.31.42.189"],
          "AttachmentIndex": 0,
          "MACAddress": "0e:98:9f:33:76:d3",
```

```
          "IPv4SubnetCIDRBlock": "172.31.32.0/20",
          "DomainNameServers": ["172.31.0.2"],
          "DomainNameSearchList": ["ec2.internal"],
          "PrivateDNSName": "ip-172-31-42-189.ec2.internal",
          "SubnetGatewayIpv4Address": "172.31.32.1/20"
        }
      ],
      "ContainerARN": "arn:aws:ecs:us-east-1:123456789012:container/MyEmptyCluster/
bfa2636268144d039771334145e490c5/f65b461d-aa09-4acb-a579-9785c0530cbc",
      "Snapshotter": "overlayfs"
    }
  ],
  "LaunchType": "FARGATE",
  "ClockDrift": {
    "ClockErrorBound": 0.446931,
    "ReferenceTimestamp": "2023-07-21T16:09:17Z",
    "ClockSynchronizationStatus": "SYNCHRONIZED"
  },
  "EphemeralStorageMetrics": {
    "Utilized": 261,
    "Reserved": 20496
  }
}
```

タスク統計情報のレスポンス例

${ECS_CONTAINER_METADATA_URI_V4}/task/stats エンドポイントにクエリを実行すると、コンテナが属しているタスクに関するネットワークメトリクスが返されます。以下に出力例を示します。

```
{
  "3d1f891cded94dc795608466cce8ddcf-464223573": {
    "read": "2020-10-08T21:24:44.938937019Z",
    "preread": "2020-10-08T21:24:34.938633969Z",
    "pids_stats": {},
    "blkio_stats": {
      "io_service_bytes_recursive": [
        {
          "major": 202,
          "minor": 26368,
          "op": "Read",
          "value": 638976
        },
```

```
        {
          "major": 202,
          "minor": 26368,
          "op": "Write",
          "value": 0
        },
        {
          "major": 202,
          "minor": 26368,
          "op": "Sync",
          "value": 638976
        },
        {
          "major": 202,
          "minor": 26368,
          "op": "Async",
          "value": 0
        },
        {
          "major": 202,
          "minor": 26368,
          "op": "Total",
          "value": 638976
        }
      ],
      "io_serviced_recursive": [
        {
          "major": 202,
          "minor": 26368,
          "op": "Read",
          "value": 12
        },
        {
          "major": 202,
          "minor": 26368,
          "op": "Write",
          "value": 0
        },
        {
          "major": 202,
          "minor": 26368,
          "op": "Sync",
          "value": 12
        },
```

```
        {
          "major": 202,
          "minor": 26368,
          "op": "Async",
          "value": 0
        },
        {
          "major": 202,
          "minor": 26368,
          "op": "Total",
          "value": 12
        }
      ],
      "io_queue_recursive": [],
      "io_service_time_recursive": [],
      "io_wait_time_recursive": [],
      "io_merged_recursive": [],
      "io_time_recursive": [],
      "sectors_recursive": []
    },
    "num_procs": 0,
    "storage_stats": {},
    "cpu_stats": {
      "cpu_usage": {
        "total_usage": 1137691504,
        "percpu_usage": [
          696479228,
          441212276,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0
        ],
        "usage_in_kernelmode": 80000000,
        "usage_in_usermode": 810000000
```

```
      },
      "system_cpu_usage": 9393210000000,
      "online_cpus": 2,
      "throttling_data": {
        "periods": 0,
        "throttled_periods": 0,
        "throttled_time": 0
      }
    },
    "precpu_stats": {
      "cpu_usage": {
        "total_usage": 1136624601,
        "percpu_usage": [
          695639662,
          440984939,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0
        ],
        "usage_in_kernelmode": 80000000,
        "usage_in_usermode": 810000000
      },
      "system_cpu_usage": 9373330000000,
      "online_cpus": 2,
      "throttling_data": {
        "periods": 0,
        "throttled_periods": 0,
        "throttled_time": 0
      }
    },
    "memory_stats": {
      "usage": 6504448,
      "max_usage": 8458240,
      "stats": {
```

```
        "active_anon": 1675264,
        "active_file": 557056,
        "cache": 651264,
        "dirty": 0,
        "hierarchical_memory_limit": 536870912,
        "hierarchical_memsw_limit": 9223372036854772000,
        "inactive_anon": 0,
        "inactive_file": 3088384,
        "mapped_file": 430080,
        "pgfault": 11034,
        "pgmajfault": 5,
        "pgpgin": 8436,
        "pgpgout": 7137,
        "rss": 4669440,
        "rss_huge": 0,
        "total_active_anon": 1675264,
        "total_active_file": 557056,
        "total_cache": 651264,
        "total_dirty": 0,
        "total_inactive_anon": 0,
        "total_inactive_file": 3088384,
        "total_mapped_file": 430080,
        "total_pgfault": 11034,
        "total_pgmajfault": 5,
        "total_pgpgin": 8436,
        "total_pgpgout": 7137,
        "total_rss": 4669440,
        "total_rss_huge": 0,
        "total_unevictable": 0,
        "total_writeback": 0,
        "unevictable": 0,
        "writeback": 0
      },
      "limit": 9223372036854772000
    },
    "name": "curltest",
    "id": "3d1f891cded94dc795608466cce8ddcf-464223573",
    "networks": {
      "eth1": {
        "rx_bytes": 2398415937,
        "rx_packets": 1898631,
        "rx_errors": 0,
        "rx_dropped": 0,
        "tx_bytes": 1259037719,
```

```
        "tx_packets": 428002,
        "tx_errors": 0,
        "tx_dropped": 0
      }
    },
    "network_rate_stats": {
      "rx_bytes_per_sec": 43.298687872232854,
      "tx_bytes_per_sec": 215.39347269466413
    }
  }
}
```

Fargate のタスク用の Amazon ECS タスクメタデータエンドポイントバージョン 3

> **Important**
>
> タスクメタデータのバージョン 3 エンドポイントは、現在アクティブにメンテナンスされていません。タスクメタデータバージョン 4 エンドポイントを更新して、最新のメタデータエンドポイント情報を取得することをお勧めします。詳細については、「the section called "Fargate のタスク用のタスクメタデータエンドポイントバージョン 4"」を参照してください。

Fargate のプラットフォームバージョン `1.1.0` 以降、`ECS_CONTAINER_METADATA_URI` という名前の環境変数がタスク内の各コンテナに挿入されます。タスクメタデータバージョン 3 エンドポイントに対してクエリを実行すると、さまざまなタスクメタデータおよび Docker 統計 をタスクで利用できます。

タスクメタデータエンドポイント機能は、プラットフォームバージョン `1.1.0` 以降を使用するFargate上でホストされるAmazon ECS タスクに対してデフォルトで有効になります。詳細については、「Amazon ECS 向け Fargate プラットフォームバージョン」を参照してください。

Fargate のタスク用のタスクメタデータエンドポイントのパス

次の API エンドポイントをコンテナで使用できます。

`${ECS_CONTAINER_METADATA_URI}`

このパスはコンテナのメタデータ JSON を返します。

`${ECS_CONTAINER_METADATA_URI}/task`

このパスはタスクのメタデータ JSON を返します。これには、タスクに関連付けられたすべてのコンテナのコンテナ ID および名前のリストが含まれています。このエンドポイントのレスポンスの詳細については、「Fargate のタスク用の Amazon ECS タスクメタデータ v3 JSON レスポンス」を参照してください。

`${ECS_CONTAINER_METADATA_URI}/stats`

このパスは特定の Docker コンテナの Docker 統計 JSON を返します。返される各統計の詳細については、Docker API ドキュメントの「ContainerStats」を参照してください。

`${ECS_CONTAINER_METADATA_URI}/task/stats`

このパスはタスクに関連付けられたすべてのコンテナの Docker 統計 JSON を返します。返される各統計の詳細については、Docker API ドキュメントの「ContainerStats」を参照してください。

Fargate のタスク用の Amazon ECS タスクメタデータ v3 JSON レスポンス

次の情報が、タスクメタデータエンドポイント (`${ECS_CONTAINER_METADATA_URI}/task`) JSON レスポンスから返されます。

`Cluster`

タスクが属する Amazon ECS クラスターの Amazon リソースネーム (ARN)または短縮名。

`TaskARN`

コンテナが属しているタスクの Amazon リソースネーム (ARN)。

`Family`

タスクの Amazon ECS タスク定義のファミリー。

`Revision`

タスクの Amazon ECS タスク定義のリビジョン。

`DesiredStatus`

Amazon ECSからのタスクの望ましいステータス。

`KnownStatus`

Amazon ECS からのタスクの既知のステータス。

`Limits`

CPU (vCPU で表される) やメモリなど、タスク レベルで指定されたリソースの制限。リソースの制限が定義されていない場合はこのパラメータは省略されます。

`PullStartedAt`

最初のコンテナイメージのプル開始時のタイムスタンプ。

`PullStoppedAt`

最後のコンテナイメージのプル終了時のタイムスタンプ。

`AvailabilityZone`

タスクがあるアベイラビリティーゾーン。

> **Note**
>
> アベイラビリティーゾーンのメタデータは、プラットフォームバージョン 1.4 以降 (Linux) または 1.0.0 以降 (Windows) を使用する Fargate タスクでのみ使用できます。

`Containers`

タスクに関連付けられている各コンテナのコンテナメタデータのリスト。

`DockerId`

コンテナの Docker ID。

`Name`

タスク定義で指定されたコンテナの名前。

`DockerName`

Docker に提供されたコンテナの名前。Amazon ECSコンテナエージェントはコンテナに一意の名前を生成し、同じタスク定義の複数のコピーが単一のインスタンスで実行される場合に名前の競合を回避します。

`Image`

コンテナのイメージ。

`ImageID`

イメージマニフェストの SHA-256 ダイジェスト。これは、`repository-url/image@sha256:digest` 形式を使用してイメージをプルするために使用できるダイジェストです。

`Ports`

コンテナに対して公開されている任意のポート。公開ポートがない場合はこのパラメータは省略されます。

`Labels`

コンテナに適用された任意のラベル。ラベルが適用されていない場合はこのパラメータは省略されます。

`DesiredStatus`

Amazon ECSからのコンテナの望ましいステータス。

`KnownStatus`

Amazon ECS からのコンテナの既知のステータス。

`ExitCode`

コンテナの終了コード。このパラメータは、コンテナが終了していない場合は省略されます。

`Limits`

CPU (CPU 単位で表される) やメモリなど、コンテナレベルで指定されたリソースの制限。リソースの制限が定義されていない場合はこのパラメータは省略されます。

`CreatedAt`

コンテナ作成時のタイムスタンプ。このパラメータは、コンテナがまだ作成されていない場合は省略されます。

`StartedAt`

コンテナ起動時のタイムスタンプ。このパラメータは、コンテナがまだ起動していない場合は省略されます。

`FinishedAt`

コンテナ停止時のタイムスタンプ。このパラメータは、コンテナがまだ停止していない場合は省略されます。

`Type`

コンテナのタイプ。タスク定義で指定されているコンテナのタイプは `NORMAL` です。他のコンテナタイプは無視してかまいません。これらはAmazon ECS コンテナエージェントによってプロビジョニングされる内部タスクリソースで使用されます。

`Networks`

ネットワークモードや IP アドレスなど、コンテナのネットワーク情報。ネットワーク情報が定義されていない場合はこのパラメータは省略されます。

`ClockDrift`

基準時刻とシステム時刻の違いに関する情報。これは、Linux オペレーティングシステムに適用されます。この機能では、Amazon Time Sync Service を使用してクロックの精度を測定し、コンテナのクロックエラー範囲を特定します。詳細については、「Linux インスタンス用 Amazon EC2 ユーザーガイド」の「Linux インスタンスの時刻の設定」を参照してください。

`ReferenceTime`

クロック精度の基礎。Amazon ECS は、`2021-09-07T16:57:44Z` などの NTP を通じて協定世界時 (UTC) の世界標準を使用しています。

`ClockErrorBound`

UTC へのオフセットとして定義されるクロックエラーの対策。この誤差は、基準時刻とシステム時刻の差 (ミリ秒単位) です。

`ClockSynchronizationStatus`

最新のシステム時刻と基準時刻間の同期試行における成功状況を示します。

有効な値は `SYNCHRONIZED` および `NOT_SYNCHRONIZED` です。

`ExecutionStoppedAt`

タスクの `DesiredStatus` が `STOPPED` に移行した時のタイムスタンプ。これは、必須コンテナが `STOPPED` に移行したときに発生します。

Fargate でのタスクの Amazon ECS タスクメタデータ v3 の例

1 つのコンテナタスクの JSON レスポンスを次に示します。

```
{
  "Cluster": "default",
```

```
  "TaskARN": "arn:aws:ecs:us-east-2:012345678910:task/9781c248-0edd-4cdb-9a93-
f63cb662a5d3",
  "Family": "nginx",
  "Revision": "5",
  "DesiredStatus": "RUNNING",
  "KnownStatus": "RUNNING",
  "Containers": [
    {
      "DockerId": "731a0d6a3b4210e2448339bc7015aaa79bfe4fa256384f4102db86ef94cbbc4c",
      "Name": "~internal~ecs~pause",
      "DockerName": "ecs-nginx-5-internalecspause-acc699c0cbf2d6d11700",
      "Image": "amazon/amazon-ecs-pause:0.1.0",
      "ImageID": "",
      "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "~internal~ecs~pause",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
        "com.amazonaws.ecs.task-definition-family": "nginx",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
      "DesiredStatus": "RESOURCES_PROVISIONED",
      "KnownStatus": "RESOURCES_PROVISIONED",
      "Limits": {
        "CPU": 0,
        "Memory": 0
      },
      "CreatedAt": "2018-02-01T20:55:08.366329616Z",
      "StartedAt": "2018-02-01T20:55:09.058354915Z",
      "Type": "CNI_PAUSE",
      "Networks": [
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": [
            "10.0.2.106"
          ]
        }
      ]
    },
    {
      "DockerId": "43481a6ce4842eec8fe72fc28500c6b52edcc0917f105b83379f88cac1ff3946",
      "Name": "nginx-curl",
      "DockerName": "ecs-nginx-5-nginx-curl-ccccb9f49db0dfe0d901",
      "Image": "nrdlngr/nginx-curl",
```

```
      "ImageID":
 "sha256:2e00ae64383cfc865ba0a2ba37f61b50a120d2d9378559dcd458dc0de47bc165",
      "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "nginx-curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
        "com.amazonaws.ecs.task-definition-family": "nginx",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
      "DesiredStatus": "RUNNING",
      "KnownStatus": "RUNNING",
      "Limits": {
        "CPU": 512,
        "Memory": 512
      },
      "CreatedAt": "2018-02-01T20:55:10.554941919Z",
      "StartedAt": "2018-02-01T20:55:11.064236631Z",
      "Type": "NORMAL",
      "Networks": [
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": [
            "10.0.2.106"
          ]
        }
      ]
    }
  ],
  "PullStartedAt": "2018-02-01T20:55:09.372495529Z",
  "PullStoppedAt": "2018-02-01T20:55:10.552018345Z",
  "AvailabilityZone": "us-east-2b"
}
```

Amazon ECS コンテナの詳細分析

Amazon ECS コンテナエージェントは、エージェントが実行しているコンテナインスタンスおよびインスタンスで実行されている関連タスクに関する詳細を収集する API オペレーションを提供します。コンテナインスタンス内から curl コマンドを使用して Amazon ECS コンテナエージェント (ポート 51678) をクエリし、コンテナインスタンスのメタデータまたはタスク情報を受け取ることができます。

> **Important**
>
> メタデータを取得するには、Amazon ECS にアクセスするための IAM; ロールがコンテナインスタンスに必要です。詳細については、「Amazon ECS コンテナインスタンスの IAM ロール」を参照してください。

コンテナインスタンスのメタデータを表示するには、SSH 経由でコンテナインスタンスにログインし、以下のコマンドを実行します。メタデータには、Amazon ECS コンテナインスタンス ID、コンテナインスタンスが登録されているクラスター、 Amazon ECS コンテナエージェントのバージョン情報が含まれます。

```
curl -s http://localhost:51678/v1/metadata | python3 -mjson.tool
```

出力:

```
{
    "Cluster": "cluster_name",
    "ContainerInstanceArn": "arn:aws:ecs:region:aws_account_id:container-
instance/cluster_name/container_instance_id",
    "Version": "Amazon ECS Agent - v1.30.0 (02ff320c)"
}
```

コンテナインスタンスで実行されているすべてのタスクの情報を表示するには、SSH 経由でコンテナインスタンスにログインし、以下のコマンドを実行します。

```
curl http://localhost:51678/v1/tasks
```

出力:

```
{
    "Tasks": [
        {
            "Arn": "arn:aws:ecs:us-west-2:012345678910:task/default/example5-58ff-46c9-
ae05-543f8example",
            "DesiredStatus": "RUNNING",
            "KnownStatus": "RUNNING",
            "Family": "hello_world",
            "Version": "8",
```

```
            "Containers": [
                {
          "DockerId":
 "9581a69a761a557fbfce1d0f6745e4af5b9dbfb86b6b2c5c4df156f1a5932ff1",
          "DockerName": "ecs-hello_world-8-mysql-fcae8ac8f9f1d89d8301",
          "Name": "mysql",
          "CreatedAt": "2023-10-08T20:09:11.44527186Z",
          "StartedAt": "2023-10-08T20:09:11.44527186Z",
          "ImageID":
 "sha256:2ae34abc2ed0a22e280d17e13f9c01aaf725688b09b7a1525d1a2750e2c0d1de"
        },
        {
          "DockerId":
 "bf25c5c5b2d4dba68846c7236e75b6915e1e778d31611e3c6a06831e39814a15",
          "DockerName": "ecs-hello_world-8-wordpress-e8bfddf9b488dff36c00",
          "Name": "wordpress"
        }
    ]
}
    ]
}
```

コンテナインスタンスで実行されている特定のタスクの情報を表示できます。特定のタスクまたはコンテナを指定するには、以下のいずれかをリクエストに追加します。

- タスク ARN (?taskarn=*task_arn*)
- コンテナの Docker ID (?dockerid=*docker_id*)

コンテナの Docker ID を使用してタスクの情報を取得するには、SSH 経由でコンテナインスタンスにログインし、以下のコマンドを実行します。

> **Note**
>
> バージョン 1.14.2 より前の Amazon ECS コンテナエージェントでは、docker ps で表示される短縮バージョンではなく、イントロスペクション API 用の完全な Docker コンテナ ID が必要です。コンテナの完全な Docker ID は、コンテナインスタンスで docker ps --no-trunc コマンドを実行して取得できます。

```
curl http://localhost:51678/v1/tasks?dockerid=79c796ed2a7f
```

出力:

```
{
    "Arn": "arn:aws:ecs:us-west-2:012345678910:task/default/e01d58a8-151b-40e8-
bc01-22647b9ecfec",
    "Containers": [
        {
            "DockerId":
 "79c796ed2a7f864f485c76f83f3165488097279d296a7c05bd5201a1c69b2920",
            "DockerName": "ecs-nginx-efs-2-nginx-9ac0808dd0afa495f001",
            "Name": "nginx",
            "CreatedAt": "2023-10-08T20:09:11.44527186Z",
            "StartedAt": "2023-10-08T20:09:11.44527186Z",
            "ImageID":
 "sha256:2ae34abc2ed0a22e280d17e13f9c01aaf725688b09b7a1525d1a2750e2c0d1de"
        }
    ],
    "DesiredStatus": "RUNNING",
    "Family": "nginx-efs",
    "KnownStatus": "RUNNING",
    "Version": "2"
}
```

ランタイムモニタリングを使用して不正な動作を特定する

Amazon GuardDuty は、AWS 環境内のアカウント、コンテナ、ワークロード、データを保護する脅威検知サービスです。GuardDuty は、機械学習（ML）モデル、異常および脅威検出機能を使用して、さまざまなログソースとランタイムアクティビティを継続的に監視し、環境内の潜在的なセキュリティリスクと悪意のあるアクティビティを特定して優先順位を付けます。

GuardDuty のランタイムモニタリングは、AWS ログとネットワークアクティビティを継続的にモニタリングして悪意のある、または不正な動作を特定することで、Fargate および EC2 コンテナインスタンスで実行されているワークロードを保護します。ランタイムモニタリングは、軽量でフルマネージド型の GuardDuty セキュリティエージェントを使用して、ファイルアクセス、プロセス実行、ネットワーク接続などのホスト上動作を分析します。これは、Amazon EC2 インスタンスおよびコンテナワークロードでの権限の昇格、流出した認証情報の使用、悪意のある IP アドレスやドメインとの通信、マルウェアの存在などの問題に対応しています。詳細については、「GuardDuty ユーザーガイド」の「GuardDuty ランタイムモニタリング」を参照してください。

セキュリティ管理者が GuardDuty 用 AWS Organizations での 1 つまたは複数のアカウントでランタイムモニタリングを有効にします。また、Fargate を使用するときに GuardDuty が GuardDuty セキュリティエージェントを自動的にデプロイするかどうかも選択します。クラスターはすべて自動的に保護され、GuardDuty がユーザーに代わってセキュリティエージェントを管理します。

次の場合は、GuardDuty セキュリティエージェントを手動で設定することもできます。

- EC2 コンテナインスタンスを使用する
- ランタイムモニタリングをクラスターレベルで有効にするのにきめ細かな制御が必要

ランタイムモニタリングを使用するには、保護対象のクラスターを設定し、EC2 コンテナインスタンスに GuardDuty セキュリティエージェントをインストールして管理する必要があります。

Amazon ECS でのランタイムモニタリングの仕組み

ランタイムモニタリングは、軽量な GuardDuty セキュリティエージェントを使用して、アプリケーションが基盤となるシステムリソースに対してどのようにリクエストし、アクセスし、消費しているかについて、Amazon ECS ワークロードのアクティビティをモニタリングします。

Fargate タスクでは、GuardDuty セキュリティエージェントは各タスクのサイドカーコンテナとして実行されます。

EC2 コンテナインスタンスでは、GuardDuty セキュリティエージェントはインスタンス上のプロセスとして実行されます。

GuardDuty セキュリティエージェントは、以下のリソースからデータを収集し、そのデータを GuardDuty に送信して処理します。検出結果は GuardDuty コンソールに表示できます。また、AWS Security Hub CSPM などのほかの AWS のサービス やサードパーティのセキュリティベンダーに送信して、集計や修復を行うこともできます。検出結果を表示および管理する方法については、「Amazon GuardDuty ユーザーガイド」の「Amazon GuardDuty 検出結果の管理」を参照してください。

- 次の Amazon ECS API コールからのレスポンス:
  - DescribeClusters

    `--include TAGS` オプションを使用すると、レスポンスパラメータにランタイムモニタリングタグ (タグが設定されている場合) が含まれます。
  - DescribeTasks

Fargate では、レスポンスパラメータに GuardDuty サイドカーコンテナが含まれます。

- ListAccountSettings

  レスポンスパラメータには、セキュリティ管理者が設定するランタイムモニタリングアカウント設定が含まれます。

- コンテナエージェントのイントロスペクションデータ。詳細については、「Amazon ECS コンテナの詳細分析」を参照してください。
- コンピューティングオプションのタスクメタデータエンドポイント:
  - Amazon ECS タスクメタデータエンドポイントバージョン 4
  - Fargate のタスク用の Amazon ECS タスクメタデータエンドポイントバージョン 4

考慮事項

ランタイムモニタリングを使用する際は、次の点を考慮してください。

- ランタイムモニタリングにはコストが関連付けられています。詳細については、「Amazon GuardDuty の料金」を参照してください。
- ランタイムモニタリングは Amazon ECS Anywhere ではサポートされていません。
- ランタイムモニタリングは Windows オペレーティングシステムではサポートされていません。
- Fargate で Amazon ECS Exec を使用する場合、GuardDuty セキュリティエージェントはサイドカーコンテナとして実行されるため、コンテナ名を指定する必要があります。
- GuardDuty セキュリティエージェントのサイドカーコンテナでは Amazon ECS Exec を使用することはできません。
- クラスターレベルでランタイムモニタリングを制御する IAM ユーザーには、タグ付け用の適切な IAM アクセス許可が必要です。詳細については、「IAM ユーザーガイド」の「IAM チュートリアル: タグに基づいて AWS リソースにアクセスするためのアクセス許可を定義する」を参照してください。
- Fargate タスクはタスク実行ロールを使用する必要があります。このロールは、ユーザーに代わって Amazon ECR プライベートリポジトリに保存されている GuardDuty セキュリティエージェントを取得、更新、管理するアクセス許可をタスクに付与します。
- Runtime Monitoring は、Amazon ECS マネージドインスタンスで実行されているアプリケーションではサポートされていません。

リソース使用率

クラスターに追加したタグは、クラスタータグクォータに加算されます。

GuardDuty エージェントサイドカーコンテナは、タスク定義クォータあたりのコンテナ数には可算されません。

ほとんどのセキュリティソフトウェアと同様に、GuardDuty には若干のオーバーヘッドがあります。Fargate のメモリ制限については、「GuardDuty ユーザーガイド」の「CPU とメモリの制限」を参照してください。Amazon EC2 のメモリ制限については、「GuardDuty エージェントの CPU とメモリの制限」を参照してください。

Amazon ECS Fargate ワークロードのランタイムモニタリング

EC2 コンテナインスタンスを使用する場合は、ランタイムモニタリングを手動で設定する必要があります。詳細については、「Amazon ECS での EC2 ワークロードのランタイムモニタリング」を参照してください。

GuardDuty にコンテナインスタンスのセキュリティエージェントを管理させることができます。このオプションは Fargate でのみ使用できます。このオプション (GuardDuty エージェント管理) は GuardDuty で使用できます。

GuardDuty エージェント管理を使用すると、GuardDuty は次の操作を実行します。

- クラスターをホストする VPC ごとに GuardDuty の VPC エンドポイントを作成します。
- 最新の GuardDuty セキュリティエージェントを取得し、すべての新しいスタンドアロン Fargate タスクと新しいサービスデプロイのサイドカーコンテナとしてインストールします。

  新しいサービスデプロイは、サービスを初めて起動したとき、または既存のサービスを [新しいデプロイの強制] オプションで更新したときに実行されます。

Amazon ECS のランタイムモニタリングを有効にする

GuardDuty を設定して、すべての Fargate クラスターのセキュリティエージェントが自動的に管理されるようにすることができます。

前提条件

ランタイムモニタリングを使用するための前提条件は次のとおりです。

- Fargate プラットフォームバージョンは Linux では `1.4.0` 以降である必要があります。
- Amazon ECS の IAM ロールとアクセス許可:
  - Fargate タスクはタスク実行ロールを使用する必要があります。このロールは、ユーザーに代わって GuardDuty セキュリティエージェントを取得、更新、管理するアクセス許可をタスクに付与します。詳細については、「Amazon ECS タスク実行IAM ロール」を参照してください。
  - 事前定義されたタグを使用してクラスターのランタイムモニタリングを制御します。アクセスポリシーによりタグに基づいてアクセスが制限されている場合は、クラスターにタグを付けるための明示的なアクセス許可を IAM ユーザーに付与する必要があります。詳細については、「IAM ユーザーガイド」の「IAM チュートリアル: タグに基づいて AWS リソースにアクセスするためのアクセス許可を定義する」を参照してください。
- Amazon ECR リポジトリへの接続:

  GuardDuty セキュリティエージェントは Amazon ECR リポジトリに保存されます。スタンドアロンタスクとサービスタスクにはそれぞれリポジトリへのアクセスが必要です。次のオプションの 1 つを使用できます。
  - パブリックサブネット内のタスクでは、タスクにパブリック IP アドレスを使用するか、タスクが実行されるサブネットに Amazon ECR の VPC エンドポイントを作成することができます。詳細については、Amazon Elastic コンテナレジストリ ユーザーガイド の Amazon ECR Interface VPC エンドポイント(AWS PrivateLink) を参照してください。
  - プライベートサブネットのタスクでは、ネットワークアドレス変換 (NAT) ゲートウェイを使用するか、タスクが実行されるサブネットに Amazon ECR の VPC エンドポイントを作成することができます。

    詳細については、「プライベートサブネットと NAT ゲートウェイ」を参照してください。
- GuardDuty の `AWSServiceRoleForAmazonGuardDuty` ロールが必要です。詳細については、「Amazon GuardDuty ユーザーガイド」の「GuardDuty でのサービスにリンクされたロールのアクセス許可」を参照してください。
- ランタイムモニタリングで保護するファイルは、すべてルートユーザーによってアクセスできる必要があります。ファイルのアクセス許可を手動で変更した場合は、`755` に設定する必要があります。

EC2 コンテナインスタンスでランタイムモニタリングを使用するための前提条件は次のとおりです。

- Amazon ECS-AMI のバージョン `20230929` 以降を使用する必要があります。

- コンテナインスタンスで Amazon ECS エージェントをバージョン 1.77 以降で実行する必要があります。
- カーネルのバージョン 5.10 以降を使用する必要があります。
- サポートされている Linux オペレーティングシステムとアーキテクチャについては、「GuardDuty ランタイムモニタリングがサポートしているオペレーティングシステムとワークロードはどれですか」を参照してください。
- Systems Manager を使用してコンテナインスタンスを管理できます。詳細については、「AWS Systems Manager Session Manager ユーザーガイド」の「Systems Manager の EC2 インスタンスのセットアップ」を参照してください。

手順

GuardDuty でランタイムモニタリングを有効にします。この機能を有効にする方法については、「Amazon GuardDuty ユーザーガイド」の「ランタイムモニタリングを有効にする」を参照してください。

既存の Amazon ECS Fargate タスクにランタイムモニタリングを追加する

ランタイムモニタリングを有効にすると、クラスター内のすべての新しいスタンドアロンタスクと新しいサービスデプロイが自動的に保護されます。イミュータビリティの制約を維持するために、既存のタスクは影響を受けません。

前提条件

1. ランタイムモニタリングを有効にします。詳細については、「Amazon ECS のランタイムモニタリングを有効にする」を参照してください。
2. Fargate タスクはタスク実行ロールを使用する必要があります。このロールは、ユーザーに代わって GuardDuty セキュリティエージェントを取得、更新、管理するアクセス許可をタスクに付与します。詳細については、「Amazon ECS タスク実行IAM ロール」を参照してください。

手順

- タスクをすぐに保護するには、次のいずれかのアクションを実行する必要があります。
  - スタンドアロンタスクの場合は、タスクを停止してその後開始します。詳細については、「Amazon ECS タスクの停止」および「Amazon ECS タスクとしてのアプリケーションの実行」を参照してください。

- サービスの一部であるタスクの場合は、「新しいデプロイの強制」オプションを使用してサービスを更新します。詳細については、「Amazon ECS サービスを更新する」を参照してください。

Amazon ECS クラスターからランタイムモニタリングを削除する

テストに使用するクラスターなど、特定のクラスターを保護対象から除外したい場合があります。そうすると、GuardDuty はクラスター内のリソース上で次のオペレーションを実行します。

- GuardDuty セキュリティエージェントを新しいスタンドアロンの Fargate タスクや新しいサービスデプロイにデプロイしなくなります。

  イミュータビリティの制約を維持するために、ランタイムモニタリングが有効になっている既存のタスクやデプロイは影響を受けません。
- 課金を停止し、タスクの実行時イベントを受け付けなくなります。

手順

以下の手順を実行して、ランタイムモニタリングをクラスターから削除します。

1. Amazon ECS コンソール または AWS CLI を使用して、クラスターの `GuardDutyManaged` タグキーを `false` に設定します。詳細については、「クラスターの更新」または「CLI または API を使用したタグの操作」を参照してください。タグに以下の値を使用します。

   > **Note**
   >
   > Key と Value は大文字と小文字が区別され、この文字列と完全に一致する必要があります。

   Key = `GuardDutyManaged`、Value = `false`
2. クラスターの GuardDuty VPC エンドポイントを削除します。VPC エンドポイントの削除方法の詳細については、「AWS PrivateLink ユーザーガイド」の「インターフェイスエンドポイントの削除」を参照してください。

アカウントから Amazon ECS のランタイムモニタリングを削除する

以降ランタイムモニタリングを使用しない場合は、GuardDuty の機能を無効にします。この機能を無効にする方法については、「Amazon GuardDuty ユーザーガイド」の「ランタイムモニタリングを有効にする」を参照してください。

GuardDuty では以下のオペレーションを実行します。

- クラスターをホストする各 VPC の GuardDuty の VPC エンドポイントを削除します。
- GuardDuty セキュリティエージェントを新しいスタンドアロンの Fargate タスクや新しいサービスのデプロイにデプロイしなくなります。

  イミュータビリティの制約を維持するために、既存のタスクやデプロイは、停止、複製、またはスケーリングされるまで影響を受けません。
- 課金を停止し、タスクの実行時イベントを受け付けなくなります。

Amazon ECS での EC2 ワークロードのランタイムモニタリング

このオプションは、キャパシティーとして EC2 インスタンスを使用する場合や、Fargate でランタイムモニタリングをクラスターレベルできめ細かく制御する必要がある場合に使用します。

事前定義されたタグを追加することで、クラスターをランタイムモニタリング用にプロビジョニングします。

EC2 コンテナインスタンスでは、ユーザーが GuardDuty セキュリティエージェントをダウンロード、インストール、管理します。

Fargate では、GuardDuty がユーザーに代わってセキュリティエージェントを管理します。

Amazon ECS のランタイムモニタリングを有効にする

EC2 インスタンスを使用するクラスターや、Fargate でランタイムモニタリングをクラスターレベルできめ細かく制御する必要がある場合に有効にできます。

ランタイムモニタリングを使用するための前提条件は次のとおりです。

- Fargate プラットフォームバージョンは Linux では `1.4.0` 以降である必要があります。
- Amazon ECS の IAM ロールとアクセス許可:

  - Fargate タスクはタスク実行ロールを使用する必要があります。このロールは、ユーザーに代わって GuardDuty セキュリティエージェントを取得、更新、管理するアクセス許可をタスクに付与します。詳細については、「Amazon ECS タスク実行IAM ロール」を参照してください。
  - 事前定義されたタグを使用してクラスターのランタイムモニタリングを制御します。アクセスポリシーによりタグに基づいてアクセスが制限されている場合は、クラスターにタグを付けるための明示的なアクセス許可を IAM ユーザーに付与する必要があります。詳細については、「IAM ユーザーガイド」の「IAM チュートリアル: タグに基づいて AWS リソースにアクセスするためのアクセス許可を定義する」を参照してください。
- Amazon ECR リポジトリへの接続:

  GuardDuty セキュリティエージェントは Amazon ECR リポジトリに保存されます。スタンドアロンタスクとサービスタスクにはそれぞれリポジトリへのアクセスが必要です。次のオプションの 1 つを使用できます。

  - パブリックサブネット内のタスクでは、タスクにパブリック IP アドレスを使用するか、タスクが実行されるサブネットに Amazon ECR の VPC エンドポイントを作成することができます。詳細については、Amazon Elastic コンテナレジストリ ユーザーガイド の Amazon ECR Interface VPC エンドポイント(AWS PrivateLink) を参照してください。
  - プライベートサブネットのタスクでは、ネットワークアドレス変換 (NAT) ゲートウェイを使用するか、タスクが実行されるサブネットに Amazon ECR の VPC エンドポイントを作成することができます。

    詳細については、「プライベートサブネットと NAT ゲートウェイ」を参照してください。
- GuardDuty の `AWSServiceRoleForAmazonGuardDuty` ロールが必要です。詳細については、「Amazon GuardDuty ユーザーガイド」の「GuardDuty でのサービスにリンクされたロールのアクセス許可」を参照してください。
- ランタイムモニタリングで保護するファイルは、すべてルートユーザーによってアクセスできる必要があります。ファイルのアクセス許可を手動で変更した場合は、755 に設定する必要があります。

EC2 コンテナインスタンスでランタイムモニタリングを使用するための前提条件は次のとおりです。

- Amazon ECS-AMI のバージョン `20230929` 以降を使用する必要があります。
- コンテナインスタンスで Amazon ECS エージェントをバージョン `1.77` 以降で実行する必要があります。

- カーネルのバージョン 5.10 以降を使用する必要があります。
- サポートされている Linux オペレーティングシステムとアーキテクチャについては、「GuardDuty ランタイムモニタリングがサポートしているオペレーティングシステムとワークロードはどれですか」を参照してください。
- Systems Manager を使用してコンテナインスタンスを管理できます。詳細については、「AWS Systems Manager Session Manager ユーザーガイド」の「Systems Manager の EC2 インスタンスのセットアップ」を参照してください。

GuardDuty でランタイムモニタリングを有効にします。この機能を有効にする方法については、「Amazon GuardDuty ユーザーガイド」の「ランタイムモニタリングを有効にする」を参照してください。

Amazon ECS クラスターのランタイムモニタリングを追加する

クラスターのランタイムモニタリングを設定し、EC2 コンテナインスタンスに GuardDuty セキュリティエージェントをインストールします。

前提条件

1. ランタイムモニタリングを有効にします。詳細については、「Amazon ECS のランタイムモニタリングを有効にする」を参照してください。
2. 事前定義されたタグを使用してクラスターのランタイムモニタリングを制御します。アクセスポリシーによりタグに基づいてアクセスが制限されている場合は、クラスターにタグを付けるための明示的なアクセス許可を IAM ユーザーに付与する必要があります。詳細については、「IAM ユーザーガイド」の「IAM チュートリアル: タグに基づいて AWS リソースにアクセスするためのアクセス許可を定義する」を参照してください。

手順

以下の操作を実行して、ランタイムモニタリングをクラスターに追加します。

1. クラスター VPC ごとに GuardDuty 用の VPC エンドポイントを作成します。詳細については、「GuardDuty ユーザーガイド」の「Amazon VPC エンドポイントの手動作成」を参照してください。
2. EC2 コンテナインスタンスを設定します。

a. クラスター内の EC2 コンテナインスタンスの Amazon ECS エージェントをバージョン 1.77 以降に更新します。詳細については、「Amazon ECS コンテナエージェントをアップデートする」を参照してください。

b. クラスターの EC2 コンテナインスタンスに GuardDuty セキュリティエージェントをインストールします。詳細については、「GuardDuty ユーザーガイド」の「Amazon EC2 インスタンスのセキュリティエージェントの手動管理」を参照してください。

   GuardDuty セキュリティエージェントは EC2 コンテナインスタンス上のプロセスとして実行されるため、新規および既存のタスク、デプロイはすべて即座に保護されます。

3. Amazon ECS コンソール または AWS CLI を使用して、クラスターの `GuardDutyManaged` タグキーを `true` に設定します。詳細については、「クラスターの更新」または「CLI または API を使用したタグの操作」を参照してください。タグに以下の値を使用します。

> **Note**
>
> Key と Value は大文字と小文字が区別され、この文字列と完全に一致する必要があります。

Key = `GuardDutyManaged`、Value = `true`

既存の Amazon ECS タスクにランタイムモニタリングを追加する

ランタイムモニタリングを有効にすると、クラスター内のすべての新しいスタンドアロンタスクと新しいサービスデプロイが自動的に保護されます。イミュータビリティの制約を維持するために、既存のタスクは影響を受けません。

前提条件

- ランタイムモニタリングを有効にします。詳細については、「Amazon ECS のランタイムモニタリングを有効にする」を参照してください。

手順

- タスクをすぐに保護するには、次のいずれかのアクションを実行する必要があります。

- スタンドアロンタスクの場合は、タスクを停止してその後開始します。詳細については、「Amazon ECS タスクの停止」および「Amazon ECS タスクとしてのアプリケーションの実行」を参照してください。
- サービスの一部であるタスクの場合は、「新しいデプロイの強制」オプションを使用してサービスを更新します。詳細については、「Amazon ECS サービスを更新する」を参照してください。

Amazon ECS クラスターからランタイムモニタリングを削除する

ランタイムモニタリングはクラスターから削除できます。そうすると、GuardDuty はクラスター内のすべてのリソースのモニタリングを停止します。

ランタイムモニタリングをクラスターから削除するには

1. Amazon ECS コンソール または AWS CLI を使用して、クラスターの `GuardDutyManaged` タグキーを `false` に設定します。詳細については、「クラスターの更新」または「CLI または API を使用したタグの操作」を参照してください。

   > **Note**
   >
   > Key と Value は大文字と小文字が区別され、この文字列と完全に一致する必要があります。

   Key = `GuardDutyManaged`、Value = `false`
2. クラスター内の EC2 コンテナインスタンスの GuardDuty セキュリティエージェントをアンインストールします。

   詳細については、「GuardDuty ユーザーガイド」の「セキュリティエージェントの手動アンインストール」を参照してください。
3. 各クラスター VPC の GuardDuty VPC エンドポイントを削除します。VPC エンドポイントの削除方法の詳細については、「AWS PrivateLink ユーザーガイド」の「インターフェイスエンドポイントの削除」を参照してください。

Amazon ECS コンテナインスタンスの GuardDuty セキュリティエージェントを更新する

EC2 コンテナインスタンスの GuardDuty セキュリティエージェントを更新する方法については、「Amazon GuardDuty ユーザーガイド」の「GuardDuty セキュリティエージェントの更新」を参照してください。

アカウントから Amazon ECS のランタイムモニタリングを削除する

以降ランタイムモニタリングを使用しない場合は、GuardDuty の機能を無効にします。この機能を無効にする方法については、「Amazon GuardDuty ユーザーガイド」の「ランタイムモニタリングを有効にする」を参照してください。

すべてのクラスターからランタイムモニタリングを削除します。詳細については、「Amazon ECS クラスターからランタイムモニタリングを削除する」を参照してください。

ランタイムモニタリングのトラブルシューティング

トラブルシューティングをして、タスクやコンテナでランタイムモニタリングが有効になっていて実行されていることを確認する必要がある場合があります。

トピック

- 自分のアカウントでランタイムモニタリングがアクティブかどうかはどうすればわかりますか?
- クラスターでランタイムモニタリングがアクティブかどうかはどうすればわかりますか?
- GuardDuty セキュリティエージェントが Fargate タスクで実行されているかどうかはどうすればわかりますか?
- GuardDuty セキュリティエージェントが EC2 コンテナインスタンスで実行されているかどうかはどうすればわかりますか?
- クラスターで実行されているタスクにタスク実行ロールがない場合はどうなりますか?
- ランタイムモニタリング対応のクラスターにタグを付けるための正しいアクセス許可を持っているかどうかはどうすればわかりますか?
- Amazon ECR に接続されていない場合はどうなりますか?
- ランタイムモニタリングを有効にした後、Fargate タスクのメモリ不足エラーに対処するにはどうすればよいですか?

自分のアカウントでランタイムモニタリングがアクティブかどうかはどうすればわかりますか?

その情報は、Amazon ECS コンソールの [アカウント設定] ページにあります。

`effective-settings` オプションを使用して `list-account-settings` を実行することもできます。

```
aws ecs list-account-settings --effective-settings
```

Output

[name] が `guardDutyActivate` に設定され、[value] が on に設定された設定は、アカウントが設定済みであることを示します。GuardDuty 管理者に問い合わせて、管理が自動か手動かを確認する必要があります。

```
{
    "setting": {
        "name": "guardDutyActivate",
        "value": "enabled",
        "principalArn": "arn:aws:iam::123456789012:root",
        "type": "aws-managed"
    }
}
```

クラスターでランタイムモニタリングがアクティブかどうかはどうすればわかりますか?

カバレッジ統計は GuardDuty コンソールで確認できます。これには、自分のアカウントまたはメンバーアカウントに関連付けられた Amazon ECS リソース情報が含まれています。カバレッジ統計は、選択した AWS リージョン 内のすべての ECS クラスターに対する正常な ECS クラスターの割合です。この統計には、Fargate および EC2 を使用するクラスターのカバレッジが含まれます。詳細については、「Amazon GuardDuty ユーザーガイド」の「カバレッジ統計の確認」を参照してください。

GuardDuty セキュリティエージェントが Fargate タスクで実行されているかどうかはどうすればわかりますか?

Fargate タスクでは、GuardDuty セキュリティエージェントはサイドカーコンテナとして実行されます。

サイドカーは、Amazon ECS コンソールの [タスクの詳細] ページの [コンテナ] に表示されます。

describe-tasks を実行することで、[name] が aws-gd-agent に設定され、[lastStatus] が RUNNING に設定されているコンテナを探すことができます。

次の例は、タスク aws:ecs:us-east-1:123456789012:task/0b69d5c0-d655-4695-98cd-5d2d5EXAMPLE のデフォルトのクラスターの出力を示しています。

```
aws ecs describe-tasks --cluster default --tasks aws:ecs:us-
east-1:123456789012:task/0b69d5c0-d655-4695-98cd-5d2d5EXAMPLE
```

Output

gd-agent という名前のコンテナは RUNNING 状態にあります。

```
"containers": [
      {
        "containerArn": "arn:aws:ecs:us-east-1:123456789012:container/4df26bb4-
f057-467b-a079-96167EXAMPLE",
        "taskArn": "arn:aws:ecs:us-east-1:123456789012:task/0b69d5c0-
d655-4695-98cd-5d2d5EXAMPLE",
        "lastStatus": "RUNNING",
        "healthStatus": "UNKNOWN",
        "memory": "string",
        "name": "aws-gd-agent"
      }
    ]
```

GuardDuty セキュリティエージェントが EC2 コンテナインスタンスで実行されているかどうかはどうすればわかりますか?

以下のコマンドを実行して、ステータスを表示します。

```
sudo systemctl status amazon-guardduty-agent
```

ログファイルは次の場所にあります。

```
/var/log/amzn-guardduty-agent
```

クラスターで実行されているタスクにタスク実行ロールがない場合はどうなりますか?

Fargate タスクでは、タスクは GuardDuty セキュリティエージェントのサイドカーコンテナなしで開始されます。GuardDuty ダッシュボードには、カバレッジ統計ダッシュボードにタスクが保護されていないことが表示されます。

ランタイムモニタリング対応のクラスターにタグを付けるための正しいアクセス許可を持っているかどうかはどうすればわかりますか?

クラスターにタグを付けるには、`CreateCluster` と `UpdateCluster` の両方に対する `ecs:TagResource` アクションが必要です。

以下はポリシーの例の一部です。

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
         "ecs:TagResource"
      ],
      "Resource": "*",
      "Condition": {
         "StringEquals": {
             "ecs:CreateAction" : [ "CreateCluster", "UpdateCluster" ]
          }
       }
    }
  ]
}
```

Amazon ECR に接続されていない場合はどうなりますか?

Fargate タスクでは、タスクは GuardDuty セキュリティエージェントのサイドカーコンテナなしで開始されます。GuardDuty ダッシュボードには、カバレッジ統計ダッシュボードにタスクが保護されていないことが表示されます。

ランタイムモニタリングを有効にした後、Fargate タスクのメモリ不足エラーに対処するにはどうすればよいですか?

GuardDuty セキュリティエージェントは軽量なプロセスです。しかし、プロセスはそれでもワークロードのサイズに応じてリソースを消費します。クラスターに GuardDuty のデプロイをステージングするには、Amazon CloudWatch Container Insights などのコンテナリソーストラッキングツールを使用することをお勧めします。このようなツールは、アプリケーションの GuardDuty セキュリティエージェントの消費プロファイルを確認するのに役立ちます。そして、必要に応じて Fargate タスクのサイズを調整して、潜在的なメモリ不足状態を回避できます。

ECS Exec を使用して Amazon ECS コンテナをモニタリングする

Amazon ECS Exec を使用すれば、最初にホストコンテナのオペレーティングシステムとやり取りしたり、インバウンドポートを開いたり、SSH キーを管理したりすることなく、コンテナと直接やり取りできます。ECS Exec を使用して、Amazon EC2 インスタンスまたは AWS Fargate で実行されているコンテナでコマンドを実行したり、シェルを取得したりできます。これにより、診断情報を収集し、エラーを迅速にトラブルシューティングすることが容易になります。例えば、開発コンテキストでは、ECS を使用して、コンテナ内のさまざまなプロセスと簡単にやり取りし、アプリケーションのトラブルシューティングを行うことができます。また本番稼働シナリオでは、これを使用することで、コンテナへのブレークグラスアクセスを行って問題をデバッグできます。

Amazon ECS API、AWS Command Line Interface (AWS CLI)、AWS、SDK、または AWS Copilot CLI から ECS Exec を使用して、実行中の Linux または Windows コンテナでコマンドを実行できます。ECS Exec の使用方法と AWS Copilot CLI を使用した動画チュートリアルの詳細については、Copilot に関する GitHub ドキュメントを参照してください。

ECS Exec を使用すれば、より厳格なアクセスコントロールポリシーを維持することもできます。この機能を選択的に有効にすることで、コマンドを実行できるユーザーと、それらのコマンドを実行できるタスクを制御できます。各コマンドとその出力のログを使用すると、ECS Exec を使用して、実行されたタスクを確認したり、CloudTrail を使用して、コンテナにアクセスしたユーザーを監査したりできます。

考慮事項

ECS Exec を使用する際は、次のことを配慮してください。

- ECS Exec は、Systems Manager がサポートしないオペレーティングシステムでの実行時に期待どおりに動作しない場合があります。サポートされているオペレーティングシステムについては、

「AWS Systems Manager ユーザーガイド」で「オペレーティングシステムのタイプ」を参照してください。

- ECS Exec は、次のインフラストラクチャで実行されるタスクでサポートされています。
  - Amazon EC2 上の Linux コンテナを任意の Amazon ECS に最適化された AMI (Bottlerocketを含む)
  - 外部インスタンス (Amazon ECS Anywhere) 上の Linux および Windows コンテナ
  - AWS Fargate 上の Linux および Windows コンテナ
  - 次の Windows Amazon ECS に最適化された AMI 上のAmazon EC2 上の Windows コンテナ (コンテナエージェントバージョン 1.56 以降を使用) :
    - Amazon ECS に最適化された Windows Server 2022 Full AMI
    - Amazon ECS に最適化された Windows Server 2022 Core AMI
    - Amazon ECS に最適化された Windows Server 2019 Full AMI
    - Amazon ECS に最適化された Windows Server 2019 Core AMI
    - Amazon ECS に最適化された Windows Server 20H2 Core AMI
- タスクに HTTP プロキシを設定した場合は、EC2 インスタンスメタデータと IAM ロールトラフィックのプロキシをバイパスするために、NO_PROXY 環境変数を "NO_PROXY=169.254.169.254,169.254.170.2" に設定します。NO_PROXY 環境変数を設定しない場合、コンテナ内のメタデータエンドポイントからインスタンスメタデータまたは IAM ロールの認証情報を取得するときに失敗する可能性があります。NO_PROXY 環境変数を推奨として設定すると、169.254.169.254 and 169.254.170.2 へのリクエストが HTTP プロキシを通過しないように、メタデータと IAM トラフィックがフィルタリングされます。
- ECS Exec および Amazon VPC
  - Amazon ECS にインターフェイス Amazon VPC エンドポイントを使用している場合は、Systems Manager Session Manager (ssmmessages) のインターフェイス Amazon VPC エンドポイントを作成する必要があります。Systems Manager VPC エンドポイントの詳細については、「AWS Systems Manager ユーザーガイド」の「Session Manager に VPC エンドポイントを設定するために AWS PrivateLink を使用する」を参照してください。
  - Amazon ECS にインターフェイス Amazon VPC エンドポイントを使用していて、暗号化に AWS KMS key を使用している場合には、AWS KMS key 用のインターフェイス Amazon VPC エンドポイントを作成する必要があります。詳細については、「AWS Key Management Service デベロッパーガイド」の「VPC エンドポイントを介した AWS KMS key への接続」を参照してください。

  - Amazon EC2 インスタンスで実行されるタスクがある場合は、awsvpc ネットワークモードを使用してください。インターネットにアクセスできない (NAT ゲートウェイを使用するように設定されていないなど) 場合は、Systems Manager Session Manager 用のインターフェイス Amazon VPC エンドポイント (ssmmessages) を作成する必要があります。awsvpc ネットワークモードに関する考慮事項の詳細については、「考慮事項」を参照してください。Systems Manager VPC エンドポイントの詳細については、「AWS Systems Manager ユーザーガイド」の「Session Manager に VPC エンドポイントを設定するために AWS PrivateLink を使用する」を参照してください。
- Amazon ECS Exec は、IPv6-only 設定で実行されているタスクではサポートされていません。IPv6-only 設定でタスクを実行する方法の詳細については、「Fargate における Amazon ECS タスクのネットワークオプション」および「EC2 の Amazon ECS タスクネットワークオプション」を参照してください。
- ECS Exec および SSM
  - タスクには、SSM セッションマネージャーに必要なアクセス許可が含まれているタスク IAM ロールが必要です。Amazon EC2 のタスクについては、タスクロールが設定されていない場合、基盤となる Amazon EC2 インスタンスのインスタンスロールが代わりに使用されます。詳細については、「ECS Exec のアクセス許可」を参照してください。
  - ユーザーが ECS Exec を使用してコンテナ上でコマンドを実行すると、これらのコマンドは root ユーザーとして実行されます。コンテナにユーザー ID を指定しても、SSM エージェントとその子プロセスは root として実行されます。
  - SSM エージェントは、必要なディレクトリやファイルを作成するために、コンテナのファイルシステムに書き込みができる必要があります。したがって、readonlyRootFilesystemタスク定義パラメータ、またはその他の方法を使ってルートファイルシステムを読み取り専用にすることは、サポートされません。
  - execute-commandアクション外部の SSM セッションを開始することは可能ですが、これにより、セッションはログに記録されず、セッション制限に対してカウントされません。IAM ポリシーを使用した ssm:start-session操作を拒否して、このアクセスを制限することをお勧めします。詳細については、「[Start Session (セッション開始)] 操作へのアクセス制限」を参照してください。
- 以下の機能はサイドカーコンテナとして実行されます。そのため、コマンドを実行するコンテナ名を指定する必要があります。
  - Runtime Monitoring
  - Service Connect

- ユーザーは、コンテナコンテキスト内で使用可能なすべてのコマンドを実行できます。一部の操作 (コンテナのメインプロセスの終了、コマンドエージェントの終了、依存関係の削除) によって、孤立プロセスとゾンビプロセスが発生する可能性があります。ゾンビプロセスをクリーンアップするには、タスク定義に `initProcessEnabled` フラグを追加することをお勧めします。
- ECS Exec はある程度の CPU とメモリを使用します。タスク定義で CPU とメモリリソースの割り当てを指定する場合は、この点を考慮する必要があります。
- AWS CLI バージョン `1.22.3` 以降、または AWS CLI バージョン `2.3.6` 以降を使用する必要があります。AWS CLI をアップデートする情報については、AWS Command Line Interface ユーザーガイドバージョン 2 の「AWS CLI の最新バージョンのインストールまたはアップデート」を参照してください。
- プロセス ID (PID) 名前空間ごとに作成できる ECS Exec セッションは 1 つのみです。タスク内で PID 名前空間を共有している 場合は、1 つのコンテナ内でのみ ECS Exec セッションを開始できます。
- ECS Exec セッションのアイドルタイムアウト時間は 20 分です。この値は変更できません。
- 既存のタスクに対して ECS Exec をオンにすることはできません。新しいタスクに対してのみオンにできます。
- `run-task` を使用して、非同期配置でマネージドスケーリングを使用するクラスターでタスクを起動 (インスタンスなしでタスクを起動) する場合、ECS Exec は使用できません。
- Microsoft Nano Server コンテナに対しては ECS Exec を実行できません。

アーキテクチャ

ECS Exec は、AWS Systems Manager (SSM) セッションマネージャーを使用して実行中のコンテナとの接続を確立し、AWS Identity and Access Management (IAM) ポリシーを使用して実行中のコンテナで実行中のコマンドへのアクセスを制御します。これは、必要な SSM Agent バイナリをコンテナにバインドマウントすることによって実現されます。Amazon ECS または AWS Fargate エージェントは、アプリケーションコードと一緒にコンテナ内で SSM コアエージェントをスタートする責任があります。詳細については、 Systems Manager のセッションマネージャーを参照してください。

AWS CloudTrail の `ExecuteCommand` を使用してコンテナにアクセスしたユーザーを監査し、各コマンド (およびその出力) を Amazon S3 または Amazon CloudWatch Logs に記録できます。独自の暗号化キーを使用してローカルクライアントとコンテナ間のデータを暗号化するには、AWS Key Management Service (AWS KMS) キーを指定する必要があります。

ECS Exec の設定

ECS Exec を使用するには、まずタスクとサービスの機能をオンにしてから、コンテナでコマンドを実行する必要があります。

オプションのタスク定義の変更

タスク定義パラメータ `initProcessEnabled` を `true` に設定すると、コンテナ内で init プロセスが開始されます。この結果、見つかったゾンビ SSM エージェントの子プロセスはすべて削除されます。以下に例を示します。

```
{
    "taskRoleArn": "ecsTaskRole",
    "networkMode": "awsvpc",
    "requiresCompatibilities": [
        "EC2",
        "FARGATE"
    ],
    "executionRoleArn": "ecsTaskExecutionRole",
    "memory": ".5 gb",
    "cpu": ".25 vcpu",
    "containerDefinitions": [
        {
            "name": "amazon-linux",
            "image": "amazonlinux:latest",
            "essential": true,
            "command": ["sleep","3600"],
            "linuxParameters": {
                "initProcessEnabled": true
            }
        }
    ],
    "family": "ecs-exec-task"
}
```

タスクとサービスに対する ECS Exec をオンにする

次の AWS CLI コマンド:(`create-service`、`update-service`、`start-task`、または `run-task`) のいずれかを使用するときに、`--enable-execute-command` フラグを指定することにより、サービスおよびスタンドアロンタスクの ECS Exec 機能をオンにすることができます。

例えば、次のコマンドを実行すると、ECS Exec 機能が Fargate で実行される新しく作成されたサービスに対してオンになります。サービス作成の詳細については、create-service を参照してください。

```
aws ecs create-service \
    --cluster cluster-name \
    --task-definition task-definition-name \
    --enable-execute-command \
    --service-name service-name \
    --launch-type FARGATE \
     --network-configuration
 "awsvpcConfiguration={subnets=[subnet-12344321],securityGroups=[sg-12344321],assignPublicIp=EN
 \
    --desired-count 1
```

タスクに対して ECS Exec をオンにしたら、次のコマンドを実行して、タスクが使用可能な状態であることを確認できます。lastStatus の ExecuteCommandAgent プロパティが RUNNING として表示され、enableExecuteCommand プロパティが true に設定されている場合、タスクの準備が整います。

```
aws ecs describe-tasks \
    --cluster cluster-name \
    --tasks task-id
```

以下の出力スニペットは、表示される可能性があるものの例です。

```
{
    "tasks": [
        {
            ...
            "containers": [
                {
                    ...
                    "managedAgents": [
                        {
                            "lastStartedAt": "2021-03-01T14:49:44.574000-06:00",
                            "name": "ExecuteCommandAgent",
                            "lastStatus": "RUNNING"
                        }
                    ]
                }
```

```
            ],
            ...
            "enableExecuteCommand": true,
            ...
        }
    ]
}
```

ECS Exec を使用してコマンド実行

ECS Exec を使用したログ記録

監査やトラブルシューティングのために、ECS Exec セッションのログを設定してコマンドとその出力を記録できます。

タスクとサービスでのログ記録を有効にする

> **Important**
>
> CloudWatch の料金の詳細については、CloudWatch の料金をご覧ください。Amazon ECS は、追加料金なしで提供されているモニタリングメトリクスも提供します。詳細については、「CloudWatch を使用して Amazon ECS をモニタリングする」を参照してください。

Amazon ECS には、ECS Exec で実行されたコマンドのログ記録がデフォルトで有効になっています。デフォルトでは、タスク定義で設定されたログドライバーを使用して CloudWatch Logs に `awslogs` ログを送信します。カスタム設定を提供する場合、AWS CLI は `create-cluster` コマンドと `update-cluster` コマンドの両方に対して `--configuration` フラグをサポートします。コマンドログを Amazon S3 または CloudWatch Logs に正しくアップロードするには、コンテナイメージに `script` と `cat` をインストールする必要があります。クラスター作成の詳細については、create-cluster を参照してください。

> **Note**
>
> この設定では、`execute-command` セッションのログ記録のみを処理します。アプリケーションのログには影響しません。

以下の例では、クラスターを作成し、出力を cloudwatch-log-group-name という名前の CloudWatch Logs LogGroup と s3-bucket-name という名前の Amazon S3 バケットに記録します。

CloudWatchEncryptionEnabledオプションをtrueに設定した場合、ロググループの暗号化にAWS KMSカスタマーマネージドキーを使用する必要があります。ロググループを暗号化する方法については、「Amazon CloudWatch Logsユーザーガイド」のAWS Key Management Serviceを使用してCloudWatch Logsのログデータを暗号化するを参照してください。

```
aws ecs create-cluster \
    --cluster-name cluster-name \
    --configuration executeCommandConfiguration="{ \
        kmsKeyId=string, \
        logging=OVERRIDE, \
        logConfiguration={ \
            cloudWatchLogGroupName=cloudwatch-log-group-name, \
            cloudWatchEncryptionEnabled=true, \
            s3BucketName=s3-bucket-name, \
            s3EncryptionEnabled=true, \
            s3KeyPrefix=demo \
        } \
    }"
```

logging プロパティにより、ECS Exec のログ機能の動作が決まります:

- NONE: ログ記録はオフになっています。
- DEFAULT: ログは設定された awslogs ドライバーに送信されます。ドライバーが設定されていない場合、ログは保存されません。
- OVERRIDE: ログは、指定された Amazon CloudWatch Logs LogGroup、Amazon S3 バケット、またはその両方に送信されます。

Amazon CloudWatch Logs または Amazon S3 ロギングに必要な IAM アクセス許可

ログ記録を有効にするには、 タスク定義で参照されるAmazon ECSタスクロールに追加のアクセス許可が必要です。これらの追加アクセス許可は、ポリシーとしてタスクロールに追加することが可能です。ログの送信先が Amazon CloudWatch Logs と Amazon S3 のどちらであるかによって異なります。

Amazon CloudWatch Logs

次のポリシー例では、Amazon CloudWatch Logs について必須のアクセス許可を追加しています。

```
{
   "Version":"2012-10-17",
   "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
               "logs:CreateLogStream",
               "logs:DescribeLogStreams",
               "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:us-east-1:111122223333:log-group:/aws/
ecs/cloudwatch-log-group-name:*"
        }
   ]
}
```

Amazon S3

次のインラインポリシー例では、Amazon S3 について必須のアクセス許可を追加しています。

```
{
   "Version":"2012-10-17",
   "Statement": [
        {
            "Effect": "Allow",
            "Action": [
               "s3:GetBucketLocation"
            ],
            "Resource": "*"
```

```
        },
        {
           "Effect": "Allow",
           "Action": [
               "s3:GetEncryptionConfiguration"
           ],
           "Resource": "arn:aws:s3:::s3-bucket-name"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::s3-bucket-name/*"
        }
    ]
   }
```

独自のAWS KMS key (KMS キー) を使用した暗号化に必要な IAM アクセス許可

デフォルトでは、ローカルクライアントとコンテナ間で転送されるデータは、AWS が提供する TLS 1.2 暗号化を使用します。独自の KMS キー を使用してデータをさらに暗号化するには、KMS キー を作成し、タスク IAM ロールに `kms:Decrypt` アクセス権限を追加する必要があります。このアクセス許可は、データを復号化するためにコンテナによって使用されます。KMS キーの 作成の詳細については、キーを作成する を参照してください。

AWS KMS アクセス許可を必要とするタスク IAM ロールに次のインラインポリシーを追加します。詳細については、「ECS Exec のアクセス許可」を参照してください。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:us-
east-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
```

```
        }
    ]
}
```

独自の KMS キー を使用してデータを暗号化するには、execute-command アクションを使用するユーザーまたはグループに kms:GenerateDataKey アクセス許可が付与されている必要があります。

以下のユーザーまたはグループのポリシー例では、独自の KMS キー を使用するために必要なアクセス許可が含まれています。KMS キーのAmazon リソースネーム (ARN) を指定する必要があります。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey"
            ],
            "Resource": "arn:aws:kms:us-
east-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ]
}
```

IAM ポリシーを使用して ECS Exec へのアクセスを制限する

次の IAM ポリシー条件キーの 1 つ以上を使用して、execute-command API アクションへのユーザーアクセスを制限します。

- aws:ResourceTag/*clusterTagKey*
- ecs:ResourceTag/*clusterTagKey*
- aws:ResourceTag/*taskTagKey*
- ecs:ResourceTag/*taskTagKey*
- ecs:container-name

- ecs:cluster
- ecs:task
- ecs:enable-execute-command

以下の IAM ポリシーの例では、ユーザーは environment キーと development 値を持つタグのあるタスク内で実行されているコンテナと、cluster-nameという名前のクラスターでコマンドを実行できます。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:ExecuteCommand",
                "ecs:DescribeTasks"
            ],
            "Resource": [
                   "arn:aws:ecs:us-east-1:111122223333:task/cluster-name/*",
                   "arn:aws:ecs:us-east-1:111122223333:cluster/cluster-name"
            ],
            "Condition": {
                "StringEquals": {
                    "ecs:ResourceTag/environment": "development"
                }
            }
        }
    ]
}
```

次の IAM ポリシーの例では、コンテナ名が production-app の場合、ユーザーは execute-command API を使用できません。

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "ecs:ExecuteCommand"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "ecs:container-name": "production-app"
                }
            }
        }
    ]
}
```

次の IAM ポリシーを使用するユーザーは、ECS Exec がオフ状態の場合にのみタスクを起動できます。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:RunTask",
                "ecs:StartTask",
                "ecs:CreateService",
                "ecs:UpdateService"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "ecs:enable-execute-command": "false"
                }
            }
        }
    ]
```

```
}
```

> **Note**
>
> `execute-command` API 操作には、リクエスト内のタスクとクラスターリソースのみが含まれるため、クラスタータグとタスクタグのみが評価されます。

IAM ポリシー条件キーの詳細については、「Service Authorization Reference」の「Actions, resources, and condition keys for Amazon Elastic Container Service」を参照してください。

[Start Session (セッション開始)] 操作へのアクセス制限

ECS Exec の外部コンテナで SSM セッションを開始することは可能ですが、セッションがログに記録されない可能性があります。ECS Exec 以外で開始されたセッションも、セッションクォータに対してカウントされません。IAM ポリシーを使用して Amazon ECS タスクに対する `ssm:start-session` アクションを直接拒否することで、このアクセスを制限することをお勧めします。使用されているタグに基づいて、すべての Amazon ECS タスクまたは特定のタスクへのアクセスを拒否できます。

以下は、指定されたクラスター名を持つすべてのリージョンのタスクに対する `ssm:start-session`アクション へのアクセスを拒否する IAM ポリシーの例です。オプションで、*`cluster-name`* にワイルドカードを含めることができます。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": "ssm:StartSession",
            "Resource": [
                   "arn:aws:ecs:us-east-1:111122223333:task/cluster-name/*"
            ]
        }
    ]
}
```

以下は、タグキー Task-Tag-Key とタグ値 Exec-Task でタグ付けされたすべてのリージョンのリソースに対する ssm:start-session アクションへのアクセスを拒否する IAM ポリシーの例です。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": "ssm:StartSession",
            "Resource": "arn:aws:ecs:*:*:task/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Task-Tag-Key": "Exec-Task"
                }
            }
        }
    ]
}
```

ECS Exec のトラブルシューティング

その他のトラブルシューティングについては、「Amazon ECS Exec に関する問題のトラブルシューティング」を参照してください。

ECS Exec を使用してコマンド実行

Amazon ECS Exec を使用して、コンテナに関連する診断情報を収集し、コンテナのライフサイクル全体で発生するエラーのトラブルシューティングを行うことができます。

前提条件

ECS Exec の使用を開始する前に、次の操作が完了していることを確認してください。

- 考慮事項を確認します。詳細については、考慮事項を参照してください。
- タスクとサービスに対する ECS Exec を設定する 詳細については、ECS Exec の設定を参照してください。

- をインストールして設定します。AWS CLI詳細については、「AWS CLI の使用を開始する」を参照してください。
- のセッションマネージャープラグインをインストールします。AWS CLI詳細については、に セッション マネージャー プラグインのインストールAWS CLI を参照してください。
- 適切なアクセス許可を持つタスクロールを設定します。ECS Exec の適切なアクセス許可のあるタスクロールを使用する必要があります。詳細については、「タスク IAM ロール」を参照してください。
- バージョンの要件を確認します。ECS Exec には、タスクが Amazon EC2 でホストされているか AWS Fargate でホストされているかに応じて、バージョン要件があります:
  - Amazon EC2 を使用している場合は、2021 年 1 月 20 日以降にリリースされた Amazon ECS 最適化 AMI を、エージェントバージョン 1.50.2 以上で使用する必要があります。詳細については、Amazon ECS 最適化 AMI を参照してください。
  - AWS Fargate を使用している場合、プラットフォームバージョン `1.4.0` 以上 (Linux) または `1.0.0` (Windows) を使用する必要があります。詳細については、の「AWS Fargateプラットフォームバージョン」を参照してください。

サービスタスクにコンソールを使用する

コンソールを使用して、ECS Exec でコマンドを実行できます。

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
2. [クラスター] ページで、クラスターを選択します。
3. [クラスターの詳細] ページの [サービス] セクションで、サービスを選択します。

   [サービスの詳細] ページが表示されます。
4. [サービスの詳細] ページで、[タスク] を選択します。次に、タスクを選択します。
5. [Containers] で、ECS Exec を使用するコンテナを選択します。
6. コマンドを実行するには、次のいずれかの方法を使用します。

   - 接続 を選択します。

     CloudShell セッションが表示され、そこでコマンドを実行できます。
   - 矢印を選択し、[AWS CLI コマンドをコピー] を選択します。

     その後、コマンドをローカル環境で実行できます。

予想される結果

接続が成功すると、コンテナからインタラクティブなシェルプロンプトが表示されます。コマンドをコンテナ環境で直接実行できるようになりました。セッションを終了するには、[セッションの終了] を選択します。

スタンドアロンタスクにコンソールを使用する

コンソールを使用して、ECS Exec でコマンドを実行できます。

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
2. [クラスター] ページで、クラスターを選択します。
3. [クラスターの詳細] ページの [タスク] セクションで、サービスを選択します。

   [タスクの詳細] ページが表示されます。
4. [Containers] で、ECS Exec を使用するコンテナを選択します。
5. コマンドを実行するには、次のいずれかの方法を使用します。

   - 接続 を選択します。

     CloudShell セッションが表示され、そこでコマンドを実行できます。
   - 矢印を選択し、[AWS CLI コマンドをコピー] を選択します。

     その後、コマンドをローカル環境で実行できます。

予想される結果

接続が成功すると、コンテナからインタラクティブなシェルプロンプトが表示されます。コマンドをコンテナ環境で直接実行できるようになりました。セッションを終了するには、[セッションの終了] を選択します。

コマンドシェルの使用

コマンドシェルを使用して、ECS Exec でコマンドを実行できます。

`ExecuteCommandAgent` が実行されていることを確認したら、以下のコマンドを使用してコンテナ上でインタラクティブシェルを開くことができます。タスクに複数のコンテナが含まれている場合は、`--container` フラグを使うコンテナ名を指定する必要があります。Amazon ECS はインタラクティブセッションの開始のみをサポートするため、`--interactive` フラグを使用する必要があります。

次のコマンドでは、*task-id* という ID のタスクで、*container-name* という名前のコンテナに対して、インタラクティブな /bin/sh コマンドが実行されます。

task-id は、タスクの Amazon リソースネーム (ARN) です。

```
aws ecs execute-command --cluster cluster-name \
    --task task-id \
    --container container-name \
    --interactive \
    --command "/bin/sh"
```

予想される結果

コマンドが成功すると、コンテナからインタラクティブなシェルプロンプトが表示されます。コマンドをコンテナ環境で直接実行できるようになりました。セッションを終了するには、exit を入力するか、[Ctrl+D] を押します。

AWS Compute OptimizerAmazon ECS に関する 推奨事項

AWS Compute Optimizer は、Amazon ECS のタスクとコンテナのサイズに関する推奨事項を生成します。詳細については、AWS Compute Optimizer ユーザーガイド の「What is AWS Compute Optimizer? (とは ?)」 を参照してください。

AWS Fargate の Amazon ECS サービスでの、タスクとコンテナサイズに関する推奨事項

AWS Compute Optimizer は、AWS Fargate の Amazon ECS サービスについての推奨事項を生成します。 AWS Compute Optimizer からは、タスクの CPU とメモリサイズおよびコンテナの CPU、コンテナメモリ、およびその予約サイズに関する推奨事項が得られます。これらの推奨事項は、Compute Optimizer コンソールの以下のページにそれぞれ表示されます。

- [Fargate の Amazon ECS サービスに関する推奨事項] ページ
- [Fargate の Amazon ECS サービスの詳細] ページ

詳細については、「AWS Compute Optimizer ユーザーガイド」の「Fargate の Amazon ECS サービスに関する推奨事項の表示」を参照してください。

Amazon ECS MCP サーバー

Amazon ECS MCP サーバーは、AI を活用した開発および運用のエクスペリエンスを実現するフルマネージドサービスです。モデルコンテキストプロトコル (MCP) は、Amazon ECS ワークロードに関するコンテキストに応じたリアルタイムの知識により AI エージェントとアプリケーションを強化する標準化されたインターフェイスを提供します。これにより、より正確でコンテキストに対応したレスポンスと AI を活用したワークフローで、Amazon ECS ワークロードを検査、トラブルシューティング、最適化できます。

> **Note**
>
> Amazon ECS MCP サーバーはプレビューリリースであり、変更される可能性があります。

概要

Amazon ECS MCP サーバーは、Kiro などの MCP 互換 AI コーディングアシスタント、または Claude Code、Cursor、Cline などのサードパーティーツールと簡単に統合でき、開発ワークフローを強化できます。

デプロイ中、サービスのデプロイとタスクのヘルスステータスをリアルタイムで可視化します。開発と運用では、このサーバーは Amazon ECS 管理を簡素化するため、コンテナのヘルスステータスをモニタリングし、サービスイベントを分析し、デプロイの問題を調査するための高レベルのワークフローを提供します。デバッグとトラブルシューティングでは、タスクの障害、コンテナログ、ネットワーク設定を分析する包括的な診断ツールを通じて迅速に問題を解決します。これらの機能は自然言語インタラクションを通じてアクセスできるため、複雑なコンテナオペレーションをより直感的かつ効率的に実行できます。

Amazon ECS MCP サーバーには、以下に使用できるツールがいくつか用意されています。

- デプロイとサービスをモニタリングする

  デプロイのステータスを確認し、サービスイベントを追跡し、コンテナのヘルスステータスをリアルタイムでモニタリングします。
- コンテナの問題をトラブルシューティングする

  カスタマイズ可能な時間枠を使用して、タスクの失敗を分析し、イメージのプルエラーを調査し、コンテナログを調べます。

- ネットワーク設定を検査する

  Amazon ECS サービスの VPC、サブネット、およびセキュリティグループ設定を確認します。
- リソースの依存関係を解決する

  タスク定義の削除を妨げるブロック要因を特定し、リソースの関係を把握します。
- セキュリティ体制を評価する

  クラスターセキュリティ設定データを収集し、AI を活用した分析と AWS のベストプラクティスの比較を行い、修復ステップを優先します。
- 最新のドキュメントにアクセスする

  Amazon ECS の機能と設定に関する正確かつタイムリーなガイダンスについては、最新の AWS ドキュメントを参照してください。

フルマネージド Amazon ECS MCP サーバーは AWS クラウドでホストされているため、ローカルでのインストールやメンテナンスは必要ありません。自動更新とパッチ適用、IAM 統合による一元化されたセキュリティ、CloudTrail による包括的な監査ログ記録、AWS の実績あるスケーラビリティ、信頼性、サポートなど、エンタープライズグレードの機能を備えています。

開始方法

開始するには、Amazon ECS MCP サーバーの使用開始 を参照してください。

Amazon ECS MCP サーバーの使用開始

このガイドでは、AI コードアシスタントで Amazon ECS MCP サーバーをセットアップして使用する手順について説明します。環境を設定し、サーバーに接続し、自然言語インタラクションを通じて Amazon ECS クラスターの管理を開始する方法が理解できます。

> **Note**
>
> Amazon ECS MCP サーバーはプレビューリリースであり、変更される可能性があります。

前提条件

開始する前に、以下があることを確認してください。

- Amazon ECS にアクセスできる AWS アカウントが作成されていること
- AWS CLI がインストールされ、認証情報を使用して設定されていること
- Python 3.10+ 以上がインストールされていること
- uv がインストールされていること

セットアップ

前提条件を確認する

Python のバージョンが 3.10 以上であることを確認します。

```
python3 --version
```

uv のインストールを確認します。

```
uv --version
```

AWS CLI の設定を確認します。

```
aws configure list
```

IAM 許可をセットアップする

AWS サービスへの読み取り専用リクエストを行い、MCP サーバーを操作するには、IAM アクセス許可が必要です。AWS 管理ポリシー、または単一のカスタムポリシーに基づいて、より詳細な制御を行うことができます。

オプション 1: マネージドポリシーとカスタムポリシーの組み合わせ

1. すべての AWS サービスへの読み取り専用アクセス用に AWS マネージドポリシー ReadOnlyAccess をアタッチします。
2. MCP アクセス許可のための追加のカスタムポリシーを作成してアタッチします (以下の MCP アクセス許可 JSON を参照)。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
      "Sid": "MCPServerAccess",
      "Effect": "Allow",
      "Action": [
        "ecs-mcp:InvokeReadOnlyTools",
        "ecs-mcp:UseMcp"
      ],
      "Resource": "*"
    }
  ]
}
```

オプション 2: 単一のカスタムポリシー (最も詳細なコントロール)

または、AWS サービスアクセス許可と MCP アクセス許可の両方を含む単一のカスタム JSON ポリシーを作成してアタッチすることもできます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "MCPServerAccess",
      "Effect": "Allow",
      "Action": [
        "ecs-mcp:InvokeReadOnlyTools",
        "ecs-mcp:UseMcp"
      ],
      "Resource": "*"
    },
    {
      "Sid": "ECSReadOnlyAccess",
      "Effect": "Allow",
      "Action": [
        "ecs:ListAccountSettings",
        "ecs:ListClusters",
        "ecs:ListContainerInstances",
        "ecs:ListTaskDefinitions",
        "ecs:ListServices",
        "ecs:ListServiceDeployments",
        "ecs:ListTasks",
        "ecs:DescribeClusters",
        "ecs:DescribeCapacityProviders",
        "ecs:DescribeContainerInstances",
        "ecs:DescribeTaskDefinition",
```

```
        "ecs:DescribeServices",
        "ecs:DescribeServiceDeployments",
        "ecs:DescribeServiceRevisions",
        "ecs:DescribeTaskSets",
        "ecs:DescribeTasks"
      ],
      "Resource": "*"
    },
    {
      "Sid": "CloudWatchLogsReadOnlyAccess",
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogGroups",
        "logs:DescribeLogStreams",
        "logs:GetLogEvents",
        "logs:FilterLogEvents"
      ],
      "Resource": "*"
    },
    {
      "Sid": "ELBReadOnlyAccess",
      "Effect": "Allow",
      "Action": [
        "elasticloadbalancing:DescribeLoadBalancers",
        "elasticloadbalancing:DescribeTargetGroups",
        "elasticloadbalancing:DescribeListeners"
      ],
      "Resource": "*"
    },
    {
      "Sid": "EC2ReadOnlyAccess",
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeVpcs",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeRouteTables",
        "ec2:DescribeNetworkInterfaces"
      ],
      "Resource": "*"
    },
    {
      "Sid": "ECRReadOnlyAccess",
      "Effect": "Allow",
```

```
      "Action": [
        "ecr:DescribeRepositories",
        "ecr:DescribeImages"
      ],
      "Resource": "*"
    },
    {
      "Sid": "IAMReadOnlyAccessForSecurityAnalysis",
      "Effect": "Allow",
      "Action": [
        "iam:GetRole",
        "iam:ListAttachedRolePolicies",
        "iam:GetPolicy",
        "iam:GetPolicyVersion",
        "iam:ListRolePolicies",
        "iam:GetRolePolicy"
      ],
      "Resource": "*"
    }
  ]
}
```

AI アシスタントを選択して設定する

MCP 互換 AI アシスタントまたは MCP 互換ツールをインストールします。例えば、Kiro、Cline、Cursor、Claude Code などを使用できます。次に、AWS の MCP Proxy を介して Amazon ECS MCP サーバーを使用するように AI コードアシスタントをセットアップします。これは、Amazon ECS MCP サーバーへの安全で認証されたアクセスのために必要です。プロキシはクライアント側のブリッジとして機能し、ローカル AWS 認証情報を使用して AWS SigV4 認証を処理します。次の例では、Kiro CLI を使用しています。Kiro での MCP のセットアップの詳細については、この リンク からご確認ください。

MCP 設定ファイルを見つける

- macOS/Linux:

```
~/.kiro/settings/mcp.json
```

- Windows:

```
%USERPROFILE%\.kiro\settings\mcp.json
```

設定ファイルが存在しない場合は、作成します。

MCP サーバー設定を追加する

リージョン ({region}) プレースホルダーは、ご使用のリージョン (us-west-2 など) に置き換えてください。リージョンの詳細なリストについては、「the section called “AWS Fargate 上の Linux コンテナ”」を参照してください。また、{profile} プレースホルダーを default などの AWS CLI プロファイル名 に置き換えてください。

Mac/Linux の場合:

```
{
  "mcpServers": {
    "ecs-mcp": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws@latest",
        "https://ecs-mcp.{region}.api.aws/mcp",
        "--service",
        "ecs-mcp",
        "--profile",
        "{profile}",
        "--region",
        "{region}"
      ]
    }
  }
}
```

Windows の場合 –

```
{
  "mcpServers": {
    "ecs-mcp": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "--from",
        "mcp-proxy-for-aws@latest",
```

```
        "mcp-proxy-for-aws.exe",
        "https://ecs-mcp.{region}.api.aws/mcp",
        "--service",
        "ecs-mcp",
        "--profile",
        "{profile}",
        "--region",
        "{region}"
      ]
    }
  }
}
```

設定を確認する

Kiro CLI (`kiro-cli`) を再起動し、MCP サーバーがロードされていることを確認し (`/mcp`)、使用可能なツール (`/tools`) を確認します。

設定を検証する

接続をテストする

AI アシスタントに簡単な質問をして、接続を確認します。

```
List all ECS clusters in my AWS account
```

Amazon ECS クラスターのリストが表示されます。

Amazon ECS MCP サーバーを使用する AI アシスタントと会話する

例 1: デプロイのモニタリング

```
Check deployment status for my web-service in production-cluster
Show me the ALB URL for my deployed service
Get service events for the last hour
```

例 2: コンテナのヘルスステータスの調査

```
Show me all tasks that failed in the last 2 hours
Why are my containers failing health checks?
Display container logs for my api-service
```

例 3: 障害のトラブルシューティング

```
Analyze task failures in my production cluster
Check for image pull errors in the last 30 minutes
Why is my task definition stuck in DELETE_IN_PROGRESS state?
```

例 4: 設定の検査

```
Show me the network configuration for my web-service
What security groups are attached to my service?
List all VPC and subnet details for my ECS service
```

例 5: セキュリティ体制を評価する

```
Are there any security issues with my production cluster?
Analyze IAM roles and policies for my web-service
Check for overly permissive security groups in my ECS cluster
```

一般的な設定とベストプラクティス

複数の AWS プロフィール

複数の AWS アカウントを使用する場合は、個別の MCP サーバー設定を作成します。

Mac/Linux の場合:

```
{
  "mcpServers": {
    "ecs-mcp-prod": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws@latest",
        "https://ecs-mcp.{region}.api.aws/mcp",
        "--service",
        "ecs-mcp",
        "--profile",
        "production",
        "--region",
        "us-west-2"
```

```
      ]
    },
    "ecs-mcp-dev": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws@latest",
        "https://ecs-mcp.{region}.api.aws/mcp",
        "--service",
        "ecs-mcp",
        "--profile",
        "development",
        "--region",
        "us-east-1"
      ]
    }
  }
}
```

セキュリティのベストプラクティス

許可された入力メカニズムを介してシークレットや機密情報を渡さない:

- シークレットや認証情報を設定ファイルに記述しない
- 機密情報をプロンプトでモデルに直接渡さない
- シークレットをタスク定義やサービス設定に含めない
- 機密情報をアプリケーションログに記録しない
- Parameter Store を使用して機密情報を保存する

ツール設定

ツールと設定の完全なリストについては、「Amazon ECS MCP サーバーのツール設定」を参照してください。

Amazon ECS MCP サーバーのツール設定

このガイドでは、フルマネージド Amazon ECS MCP サーバーで使用できるすべての設定について説明します。

> **Note**
>
> Amazon ECS MCP サーバーはプレビューリリースであり、変更される可能性があります。

環境変数

AWS_PROFILE (オプション)

使用する AWS の認証情報プロファイル名。`--profile` コマンドライン引数で上書きできます。

例:`export AWS_PROFILE=dev`

AWS_REGION (オプション)

SigV4 署名用の AWS リージョン。設定されていない場合は `us-west-2` がデフォルトになります。

例:`export AWS_REGION=us-west-2`

引数

SigV4 MCP エンドポイント URL (必須)

接続先の MCP エンドポイント URL です。

例:`https://ecs-mcp.us-west-2.api.aws/mcp`

--service (オプション)

AWSSigV4 署名用の のサービス名。指定されていない場合は、エンドポイントホスト名から自動検出されます。

例:`--service ecs-mcp`

--profile (オプション)

AWS使用する の認証情報プロファイル。指定されていない場合は、`AWS_PROFILE` 環境変数がデフォルトになります。

例:`--profile dev`

--region

AWS使用する リージョン。設定されていない場合は、AWS_REGION 環境変数を使用します。デフォルトは us-east-1 です。

例:--region us-west-2

--read-only (オプション)

書き込みアクセス許可が必要なツールを無効にします (書き込みアクセス許可を必要としないツールには readOnlyHint=true という注釈が付けられます)。デフォルトでは、すべてのツールが有効になっています。

例:--read-only

その他の設定オプションについては、「Configuration parameters」を参照してください。

ツール

Amazon ECS MCP サーバーは、次の MCP ツール を公開しています。

読み取り専用ツール

get_deployment_status

このツールでは、特定の Amazon ECS クラスターとサービスの Amazon ECS デプロイステータスをチェックできます。

- 必須 IAM アクション:
  - ecs:DescribeServices
  - elasticloadbalancing:DescribeTargetGroups
  - elasticloadbalancing:DescribeLoadBalancers
- 必須パラメータ:
  - cluster_name (文字列): Amazon ECS クラスター名
  - service_name (文字列): Amazon ECS サービス名
- 応答: このツールは、デプロイのステータスとデプロイに関連付けられた詳細を返します。

fetch_service_events

このツールでは、カスタマイズ可能な時間枠で診断用の Amazon ECS サービスイベントを取得できます。

- 必須 IAM アクション:
  - `ecs:DescribeServices`
- 必須パラメータ:
  - `cluster_name` (文字列): Amazon ECS クラスター名
  - `service_name` (文字列): Amazon ECS サービス名
- 任意指定のパラメータ:
  - `time_window` (整数): 時間枠を秒単位で指定できます (デフォルトは 1 時間または 3600 秒)
  - `start_time` (文字列): ISO 形式のカスタム開始時刻
  - `end_time` (文字列): ISO 形式のカスタム終了時刻
- 応答: このツールは、サマリーを含むサービスのイベントのリストを返します。

fetch_task_failures

このツールは、サマリーを使用して Amazon ECS タスクの失敗を取得および分析するのに役立ちます。このツールを使用して、タスクの失敗のパターンを特定できます。

- 必須 IAM アクション:
  - `ecs:ListTasks`
  - `ecs:DescribeTasks`
- 必須パラメータ:
  - `cluster_name` (文字列): タスクが存在する Amazon ECS クラスターの名前
- 任意指定のパラメータ:
  - `service_name` (文字列): 特定のサービスでフィルタリングできます
  - `time_window` (整数): 時間枠を秒単位で指定できます (デフォルトは 1 時間または 3600 秒)
  - `start_time` (文字列): ISO 形式のカスタム開始時刻
  - `end_time` (文字列): ISO 形式のカスタム終了時刻
- 応答: このツールは、失敗したタスクとサマリーを含む TaskFailureResult を返します。

fetch_task_logs

このツールでは、柔軟な時間範囲オプションを使用して Amazon ECS タスクの CloudWatch ログを取得できます。このツールは、ランタイムの問題のトラブルシューティングに使用します。

- 必須 IAM アクション:
  - `ecs:DescribeServices`
  - `ecs:DescribeTaskDefinition`

    - `logs:FilterLogEvents`
- 必須パラメータ:
    - `cluster_name` (文字列): Amazon ECS クラスター名
    - `service_name` (文字列): Amazon ECS サービス名
- 任意指定のパラメータ:
    - `time_window` (整数): 時間枠を秒単位で指定できます (デフォルトは 1 時間または 3600 秒)
    - `start_time` (文字列): ISO 形式のカスタム開始時刻
    - `end_time` (文字列): ISO 形式のカスタム終了時刻
    - `log_group_name` (文字列): CloudWatch ロググループを指定します
- 応答: ツールはログエントリを含む TaskLogsResult を返します。

get_task_definition_deletion_blockers

このツールでは、タスク定義の削除を妨げる可能性のある依存関係を識別できます。このツールは、クリーンアップオペレーションを実行していて、Amazon ECS リソースの削除を妨げている原因がわかっている場合に適しています。

- 必須 IAM アクション:
    - `ecs:ListClusters`
    - `ecs:ListServices`
    - `ecs:DescribeServices`
    - `ecs:ListTasks`
    - `ecs:DescribeTasks`
- 必須パラメータ:
    - `task_definition_arn` (文字列): 分析するタスク定義の Amazon リソースネーム (ARN)
- 応答: ツールはブロッカーと削除ステータスを返します。

detect_image_pull_failures

このツールは、コンテナイメージのプルエラーを検出して分類するのに役立ちます。

- 必須 IAM アクション:
    - `ecs:ListTasks`
    - `ecs:DescribeTasks`
- 必須パラメータ:

    - `cluster_name` (文字列): Amazon ECS クラスター名
- 任意指定のパラメータ:
    - `service_name` (文字列): 特定のサービスでフィルタリングできます
    - `time_window` (整数): 時間枠を秒単位で指定できます (デフォルトは 1 時間または 3600 秒)
    - `start_time` (文字列): ISO 形式のカスタム開始時刻
    - `end_time` (文字列): ISO 形式のカスタム終了時刻
- 応答: ツールはエラーとサマリーを返します。

fetch_network_configuration

このツールでは、Amazon ECS サービスのネットワーク設定の詳細を取得できます。このツールは、Amazon ECS セットアップの Amazon VPC、サブネット、セキュリティグループ設定を把握するために使用できます。

- 必須 IAM アクション:
    - `ecs:DescribeServices`
    - `ec2:DescribeSecurityGroups`
    - `ec2:DescribeSubnets`
- 必須パラメータ:
    - `cluster_name` (文字列): Amazon ECS クラスター名
    - `service_name` (文字列): Amazon ECS サービス名
- 応答: ツールはネットワークの詳細を含む NetworkConfigurationResult のリストを返します。

get_security_recommendations

このツールは、AI を活用したセキュリティ分析を行うために Amazon ECS クラスターのセキュリティ設定データを収集します。IAM ロールとポリシー、ネットワーク設定 (セキュリティグループ、サブネット、パブリック IP)、タスク定義コンテナ設定、クラスターレベルの設定を収集します。AI アシスタントは、このデータを AWS のセキュリティのベストプラクティスと比較し、修復手順を含む、優先順位付けされた検出結果レポートを生成します。

- 必須 IAM アクション:
    - `ecs:DescribeClusters`
    - `ecs:ListServices`
    - `ecs:DescribeServices`
    - `ecs:DescribeTaskDefinition`
    - `ec2:DescribeSecurityGroups`

  - ec2:DescribeSubnets
  - iam:GetRole
  - iam:ListAttachedRolePolicies
  - iam:GetPolicy
  - iam:GetPolicyVersion
  - iam:ListRolePolicies
  - iam:GetRolePolicy
- 必須パラメータ:
  - cluster_name (文字列): 分析する Amazon ECS クラスター名
- 任意指定のパラメータ:
  - service_name (文字列): 分析を 1 つのサービスにスコープします。省略した場合、クラスター内のすべてのサービスが分析されます。
  - max_services (整数): 分析するサービスの最大数 (デフォルトは 25、上限は 50)
- レスポンス: このツールは、クラスター設定、サービス設定、タスク定義設定、ネットワークセキュリティの詳細、IAM ロールとポリシーデータ、コレクションの警告を返します。AI アシスタントはこのデータを使用して、修復ガイダンスを含む優先順位付けされたセキュリティ検出結果レポートを生成します。

fetch_action_logs

このツールを使用して、Amazon ECS アクションログを取得し、コンテナライフサイクルオペレーションのトラブルシューティングを行います。Action Logs を使用すると、ENI アタッチメント、イメージプル、ロードバランサー登録、スロットリングの再試行、アクセス許可エラーなど、Amazon ECS がユーザーに代わって実行するサービス側のオペレーションを可視化できます。このツールは、タスクの起動に失敗した場合、デプロイが停止した場合、またはオペレーション中に Amazon ECS が何を実行したかを確認する必要がある場合に使用します。このツールは、クラスターで Action Logs が有効になっている場合はアクションログエントリを返し、Action Logs が設定されていない場合はステータスメッセージを返します。

- 必須 IAM アクション:
  - ecs:DescribeClusters
  - logs:DescribeDeliverySources
  - logs:DescribeDeliveries
  - logs:DescribeDeliveryDestinations
  - logs:FilterLogEvents

- 必須パラメータ:
  - `cluster_name`: Amazon ECS クラスターの名前または ARN。
- 任意指定のパラメータ:
  - `service_name`: アクションログのフィルタリングに使用する Amazon ECS サービス名。
  - `task_id`: アクションログのフィルタリングに使用するタスク ARN または ID。
  - `deployment_id`: アクションログのフィルタリングに使用するデプロイ ID。
  - `log_level`: ログレベルのフィルター。有効な値は、ERROR、WARN、または INFO です。
  - `time_window`: 秒単位の時間ウィンドウ。デフォルト値は 3,600 秒です。
  - `start_time`: ISO 形式のカスタム開始時刻。
  - `end_time`: ISO 形式のカスタム終了時刻。
- レスポンス:

  このツールは、クラスターのアクションログ検出ステータスを返します。アクションログが CloudWatch Logs の送信先で有効になっている場合、ツールは一致するログエントリも返します。各エントリには、タイムスタンプ、ログレベル、イベント名、リソース ARN、イベントの詳細が含まれます。結果は 1,000 エントリに制限されます。より絞り込んだデータを取得するには、より狭い時間枠または追加のフィルターを使用してください。

ナレッジツール

これらのツールは、最新の AWS ドキュメントへのアクセスを提供します。AWS ナレッジ MCP サーバーを使用しており、発信者からの IAM アクセス許可は必要ありません。Amazon ECS MCP サービスはユーザーに代わってナレッジエンドポイントに接続します。

aws_knowledge_aws___search_documentation

このツールは、クエリに一致する関連ページを AWS ドキュメントで検索します。

- 必須 IAM アクション: なし
- 必須パラメータ:
  - `search_phrase` (文字列): AWS ドキュメントの検索クエリ
- 任意指定のパラメータ:
  - `topics` (配列): 最大 3 つのトピック領域を検索 (デフォルトは `["general"]`)
  - `limit` (整数): トピックあたりの結果の最大数 (デフォルトは 5)

- レスポンス: このツールは、タイトル、URL、コンテキストスニペットを含む一致するドキュメントページを返します。

aws_knowledge_aws__read_documentation

このツールは、AWS ドキュメントページの内容を読み取ってマークダウン形式で返します。

- 必須 IAM アクション: なし
- 必須パラメータ:
  - `requests` (配列): リクエストオブジェクトの配列。各オブジェクトには以下が含まれます。
    - `url` (文字列、必須): 読み取るドキュメントページの URL
    - `max_length` (整数、オプション): 返す最大文字数 (デフォルトは 10,000)
    - `start_index` (整数、オプション): 読み取りを開始する文字の位置 (デフォルトは 0)
- レスポンス: ツールはドキュメントページの内容をマークダウン形式で返します。長いページは、ナビゲーションをサポートするチャンクで返されます。

Amazon ECS トラブルシューティング

ロードバランサー、タスク、サービス、またはコンテナインスタンスの問題のトラブルシューティングが必要な場合があります。この章は、Amazon ECS コンソールで Amazon ECS コンテナエージェント、コンテナインスタンス上の Docker デーモン、サービスイベントログから診断情報を見つけるのに役立ちます。

停止したタスクの詳細については、次のいずれかを参照してください。

アクション	詳細情報
停止したタスクのエラーを解決します。	Amazon ECS の停止したタスクのエラーを表示する
停止したタスクのエラーを表示します。	Amazon ECS の停止したタスクのエラーを解決する
停止したタスクのエラーコードを確認します。	Amazon ECS の停止したタスクのエラーメッセージ
CannotPullContainer タスクのエラーを確認します。	Amazon ECS での CannotPullContainer タスクエラー
タスクの IAM ロールリクエストを表示します。	Amazon ECS タスクの IAM ロールリクエストの表示
タスクイベントを使用してトラブルシューティングを実行します。	コンソールの Amazon ECS イベントキャプチャ

サービスエラーの詳細については、次のいずれかを参照してください。

アクション	詳細情報
サービスイベントメッセージを表示します。	Amazon ECS のサービスイベントメッセージを表示する

アクション	詳細情報	
サービスイベントメッセージを確認します。	Amazon ECS のサービスイベントメッセージ	
ロードバランサーの問題を確認します。	Amazon ECS のサービスロードバランサーのトラブルシューティング	
サービスの自動スケーリングの問題を確認します。	Amazon ECS のサービス自動スケーリングのトラブルシューティング	

タスク定義のエラーの詳細については、次のいずれかを参照してください。

アクション	詳細情報	
タスク定義のメモリエラーを解決します。	Amazon ECS タスク定義の無効な CPU またはメモリエラーをトラブルシューティングする	

Amazon ECS エージェントのエラーの詳細については、次のいずれかを参照してください。

アクション	詳細情報	
Amazon ECS コンテナエージェントログを表示します。	Amazon ECS コンテナエージェントログの表示	
Amazon ECS ログを収集する方法について説明します。	Amazon ECS ログコレクターを使用したコンテナログの収集	
Amazon ECS エージェントを使用して診断の詳細を取得します。	エージェントのイントロスペクションを使用して Amazon ECS 診断の詳細を取得する	

Docker エラーの一般的な情報については、次のいずれかを参照してください。

アクション	詳細情報
Docker 診断を使用します。	Amazon ECS の Docker 診断
Docker デバッグモードを有効にします。	Amazon ECS の Docker デーモンからの詳細な出力の設定
Docker API エラー 500 のトラブルシューティングを実行します。	Amazon ECS の Docker API error (500): devmapper のトラブルシューティング

ECS Exec および Amazon ECS Anywhere のエラーの詳細については、次のいずれかを参照してください。

アクション	詳細情報
ECS Exec のトラブルシューティングを実行します。	Amazon ECS Exec に関する問題のトラブルシューティング
Amazon ECS Anywhere のトラブルシューティングを実行します。	Amazon ECS Anywhere に関する問題のトラブルシューティング

Amazon EBS ボリュームの Amazon ECS タスクへのアタッチに関する問題については、以下を参照してください。

アクション	詳細情報
Amazon ECS タスクへの Amazon EBS ボリュームのアタッチに関するトラブルシューティング。	Amazon ECS タスクへの Amazon EBS ボリュームのアタッチに関するトラブルシューティング

アクション	詳細情報	
Amazon ECS タスクへの Amazon EBS ボリュームのアタッチに関するステータスの理由。	Amazon ECS タスクへの Amazon EBS ボリュームアタッチメントのステータス理由	

Amazon ECS Service Connect で共有 AWS Cloud Map 名前空間を使用する際の問題については、次のいずれかを参照してください。

アクション	詳細情報	
共有 AWS Cloud Map 名前空間を使用した Amazon ECS Service Connect のトラブルシューティング。	共有 AWS Cloud Map 名前空間を使用した Amazon ECS Service Connect のトラブルシューティング	

スロットリングの問題を解決するための情報については、次のいずれかを参照してください。

アクション	詳細情報	
Fargate のスロットリングクォータについて説明します。	AWS Fargate スロットリングのクォータ	
Amazon ECS スロットリングのベストプラクティスについて説明します。	Amazon ECS のスロットリングに関する問題を処理する	

API エラーの詳細については、次のいずれかを参照してください。

アクション	詳細情報	
API エラーを解決します。	Amazon ECS での API エラーの原因	

AI を活用したトラブルシューティングについては、以下を参照してください。

アクション	詳細情報	
Amazon Q Developer に関するコンソールのトラブルシューティング。	Amazon Q Developer のトラブルシューティング	
Amazon ECS MCP サーバーを使用して、AI アシスタントを用いたトラブルシューティングを行います。	Amazon ECS MCP サーバー	

Amazon ECS の停止したタスクのエラーを解決する

タスクの開始に失敗すると、コンソールと `describe-tasks` 出力パラメータ (`stoppedReason` および `stopCode`) にエラーメッセージが表示されます。

停止したタスクは、コンソールで 1 時間表示できます。停止したタスクを表示するには、フィルターオプションを変更する必要があります。詳細については、「Amazon ECS の停止したタスクのエラーを表示する」を参照してください。

次のページには、エラーコードに関する情報が記載されています。

- 停止したタスクのエラーメッセージの変更について説明します。

  Amazon ECS の停止したタスクのエラーメッセージの更新
- 停止したタスクを表示して、原因に関する情報を取得します。

  Amazon ECS の停止したタスクのエラーを表示する
- 停止したタスクのエラーメッセージと、考えられるエラーの原因について説明します。

  Amazon ECS の停止したタスクのエラーメッセージ
- 停止したタスク接続を検証し、エラーを修正する方法について説明します。

  Amazon ECS の停止したタスクの接続を検証する

Amazon ECS の停止したタスクのエラーメッセージの更新

2024 年 6 月 14 日より、Amazon ECS チームは、以下の表に示すように停止したタスクのエラーメッセージを変更します。stopCode の変更はありません。アプリケーションが正確なエラーメッセージ文字列に依存している場合は、新しい文字列でアプリケーションを更新する必要があります。ご質問や問題については、AWS サポート にお問い合わせください。

> **Note**
>
> このエラーメッセージは変更される可能性があるため、オートメーションではエラーメッセージに依存しないことをお勧めします。

CannotPullContainerError

以前のエラーメッセージ	新しいエラーメッセージ
CannotPullContainerError: Error response from daemon: pull access denied for *repository* , repository does not exist or may require 'docker login': denied: User: *roleARN*	• CannotPullContainerError: The task can't pull the image. ロールにレジストリからイメージをプルするためのアクセス許可があることを確認します。Error response from daemon: pull access denied for *repository* , repository does not exist or may require 'docker login': denied: User: *roleARN* is not authorized to perform: ecr:BatchGetImage on resource: *image* because no identity-based policy allows the ecr:BatchGetImage action. • CannotPullContainerError: The task can't pull the

以前のエラーメッセージ	新しいエラーメッセージ	
	image. イメージが存在することを確認します。Error response from daemon: pull access denied for *repository* , repository does not exist or may require 'docker login': denied: requested access to the resource is denied.	
CannotPullContainerError: Error response from daemon: Get *imageURI*: net/http: request canceled while waiting for connection	CannotPullContainerError: The task can't pull the image. ネットワーク設定を確認します。Error response from daemon: Get *image*: net/http: request canceled while waiting for connection (Client.Timeout exceeded while awaiting headers)	
CannotPullContainerError: ref pull has been retried 5 time(s): failed to copy: httpReadSeeker: failed open: failed to do request: Get *registry-uri* : dial tcp <ip>:<port> i/o timeout	CannotPullContainerError: The task cannot pull *image-uri* from the registry *registry-uri* . There is a connection issue between the task and the registry. Check your task network configuration. : failed to copy: httpReadSeeker: failed open: failed to do request: Get *registry-uri* : dial tcp <ip>:<port> i/o timeout	

ResourceNotFoundException

以前のエラーメッセージ	新しいエラーメッセージ	
Fetching secret data from AWS Secrets Manager in region *region*: secret *sercretARN* : ResourceNotFoundException: Secrets Manager can't find the specified secret.	ResourceNotFoundException: The task can't retrieve the secret with ARN '*sercretARN* ' from AWS Secrets Manager. Check whether the secret exists in the specified Region. ResourceNotFoundException: Fetching secret data from AWS Secrets Manager in region *region*: secret *sercretARN* : ResourceNotFoundException: Secrets Manager can't find the specified secret.	

ResourceInitializationError

以前のエラーメッセージ	新しいエラーメッセージ	
ResourceInitializationError : unable to pull secrets or registry auth: execution resource retrieval failed: unable to retrieve ecr registry auth: service call has been retried 3 time(s): RequestError: send request failed caused by: Post "https://api.ecr.us-east-1.amazonaws.com/": dial tcp <ip>:<port>: i/	ResourceInitializationError : unable to pull secrets or registry auth: The task cannot pull registry auth from Amazon ECR: There is a connection issue between the task and Amazon ECR. Check your task network configuration. RequestError: send request failed caused by: Post "https://api.ecr.us-east-1.amazonaws	

以前のエラーメッセージ	新しいエラーメッセージ	
o timeout. Please check your task network configuration.	.com": dial tcp <ip>:<port>: i/o timeout	
ResourceInitializationError : unable to pull secrets or registry auth: execution resource retrieval failed: unable to retrieve secrets from ssm: service call has been retried 5 time(s): RequestCa nceled: request context canceled caused by: context deadline exceeded	ResourceInitializationError : unable to pull secrets or registry auth: unable to retrieve secrets from ssm: The task cannot pull secrets from AWS Systems Manager. There is a connectio n issue between the task and AWS Systems Manager Parameter Store. Check your task network configuration. RequestCanceled: request context canceled caused by: context deadline exceeded	
ResourceInitializationError: failed to download env files: file download command: non empty error stream: RequestCanceled: request context canceled caused by: context deadline exceeded	ResourceInitializationError: failed to download env files: The task can't download the environment variable files from Amazon S3. There is a connection issue between the task and Amazon S3. Check your task network configuration. service call has been retried 5 time(s): RequestCanceled: request context canceled caused by: context deadline exceeded	

以前のエラーメッセージ	新しいエラーメッセージ	
ResourceInitializationError : failed to validate logger args::signal:killed	ResourceInitializationError : failed to validate logger args: The task cannot find the Amazon CloudWatch log group defined in the task definition. There is a connectio n issue between the task and Amazon CloudWatch. Check your network configuration. : signal: killed	
ResourceInitializationError : unable to pull secrets or registry auth: pull command failed: : signal: killed	ResourceInitializationError : unable to pull secrets or registry auth: Check your task network configuration. : signal: killed	

Amazon ECS の停止したタスクのエラーを表示する

タスクの開始に問題がある場合、アプリケーションエラーまたは設定エラーのためにタスクが停止している可能性があります。例えば、タスクを実行するとタスクが `PENDING` ステータスを表示して消えるとします。

タスクが Amazon ECS サービスによって作成された場合、Amazon ECS がサービスを維持するために行うアクションはサービスイベントで公開されます。イベントは、AWS マネジメントコンソール、AWS CLI、AWS SDK、Amazon ECS API または SDK と API を使用するツールで表示できます。これらのイベントには、タスク内のコンテナの実行が停止したり、Elastic Load Balancing によるヘルスチェックに何度も失敗したりしたことが原因で、Amazon ECS が停止してタスクが置き換えられることが含まれます。

また、タスクが Amazon EC2 にあるコンテナインスタンスまたは外部コンピュータで実行された場合、コンテナランタイムと Amazon ECS エージェントのログを確認することもできます。これらのログは、ホスト Amazon EC2 インスタンスまたは外部コンピュータにあります。詳細については、「Amazon ECS コンテナエージェントログの表示」を参照してください。

手順

Console

AWS マネジメントコンソール

次の手順に従って、コンソールを使用して停止されたタスクにエラーがないかどうかを確認することができます。停止したタスクを表示するには、フィルターオプションを変更する必要があります。

停止したタスクは 1 時間だけコンソールに表示されます。

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。
3. [Clusters] (クラスター) ページで、クラスターを選択します。
4. [Cluster: ***name***] (クラスター: 名前) ページで、[Tasks] (タスク) タブを選択します。
5. 停止済みのタスクを表示するようにフィルターを設定します。希望ステータスのフィルターでは、停止 を選択します。

   [停止済み] オプションには停止されたタスクが表示され、[任意の必要なステータス] にはすべてのタスクが表示されます。
6. 停止したタスクを選択して検査します。
7. 停止済みのタスクの行の [前回のステータス] 列で、[停止済み] を選択します。

   ポップアップウィンドウに停止した理由が表示されます。

AWS CLI

1. クラスターで停止したタスクを一覧表示します。出力には、タスクの Amazon リソースネーム (ARN) が含まれますが、この名前は、タスクを説明するものである必要があります。

   ```
   aws ecs list-tasks \
        --cluster cluster_name \
        --desired-status STOPPED \
        --region region
   ```
2. 停止したタスクを記述して情報を取得します。詳細については、「AWS Command Line Interfaceリファレンス」の「describe-tasks」を参照してください。

```
aws ecs describe-tasks \
     --cluster cluster_name \
     --tasks arn:aws:ecs:region:account_id:task/cluster_name/task_ID \
     --region region
```

次の出力パラメータを使用します。

- `stopCode` - ResourceInitializationError などの停止コードは、タスクが停止された理由を示します。
- `StoppedReason` - タスクが停止した理由を示します。
- `reason` (`containers` 構造の場合) - 理由には、停止したコンテナに関するその他の詳細が含まれます。

次のステップ

停止したタスクを表示して、原因に関する情報を取得します。詳細については、「Amazon ECS の停止したタスクのエラーメッセージ」を参照してください。

Amazon ECS の停止したタスクのエラーメッセージ

タスクが予期せず停止した場合に表示される可能性のあるエラーメッセージは、以下のとおりです。

停止したタスクのエラーメッセージを AWS マネジメントコンソール で確認する方法については、「Amazon ECS の停止したタスクのエラーを表示する」を参照してください。

> **Tip**
>
> Amazon ECS MCP サーバー と AI アシスタントを使用することで、タスクの失敗とコンテナログの分析を自然言語で実行できます。

停止したタスクのエラーコードには、「ResourceInitializationError」などのカテゴリが関連付けられています。各カテゴリの詳細については、以下を参照してください。

Category	詳細情報
TaskFailedToStart	Amazon ECS TaskFailedToStart エラーのトラブルシューティング
ResourceInitializationError	Amazon ECS での ResourceInitializationError エラーのトラブルシューティング
ResourceNotFoundException	Amazon ECS ResourceNotFoundException エラーのトラブルシューティング
SpotInterruptionError	Amazon ECS での SpotInterruption エラーのトラブルシューティング
InternalError	Amazon ECS での InternalError エラーのトラブルシューティング
OutOfMemoryError	Amazon ECS での OutOfMemoryError エラーのトラブルシューティング
ContainerRuntimeError	Amazon ECS での ContainerRuntimeError エラーのトラブルシューティング
ContainerRuntimeTimeoutError	Amazon ECS での ContainerRuntimeTimeoutError エラーのトラブルシューティング
CannotStartContainerError	Amazon ECS での CannotStartContainerError エラーのトラブルシューティング

Category	詳細情報	
CannotStopContainerError	Amazon ECS での CannotStopContainerError エラーのトラブルシューティング	
CannotInspectContainerError	Amazon ECS での CannotInspectContainerError エラーのトラブルシューティング	
CannotCreateVolumeError	Amazon ECS での CannotCreateVolumeError エラーのトラブルシューティング	
CannotPullContainer	Amazon ECS での CannotPullContainer タスクエラー	

Amazon ECS TaskFailedToStart エラーのトラブルシューティング

以下に、`TaskFailedToStart` エラーメッセージおよび、そのエラーを修正するために取れる措置を示します。

停止したタスクのエラーメッセージを AWS マネジメントコンソール で確認する方法については、「Amazon ECS の停止したタスクのエラーを表示する」を参照してください。

Unexpected EC2 error while attempting to Create Network Interface with public IP assignment enabled in subnet '***subnet-id***

これは、`awsvpc` ネットワークモードを使用し、パブリック IP アドレスを持つサブネットで実行される Fargate タスクにおいて、サブネットに十分な IP アドレスがない場合に発生します。

利用可能な IP アドレスの数は、Amazon EC2 コンソールのサブネットの詳細ページ、または `describe-subnets` を使用して確認できます。詳細については、「Amazon VPC ユーザーガイド」の「サブネットを表示する」を参照してください。

この問題を修正するには、タスクを実行するための新しいサブネットを作成します。

InternalError: *<reason>*

このエラーは、ENI アタッチメントが要求されたときに発生します。Amazon EC2 は ENI のプロビジョニングを非同期で処理します。プロビジョニングプロセスには時間がかかります。Amazon ECS では、待ち時間が長かったり、エラーが報告されない場合に備えてタイムアウトを設けています。ENI がプロビジョニングされても、レポートは障害タイムアウト後に Amazon ECS に送られる場合があります。この場合、Amazon ECS は使用中の ENI で報告されたタスク障害を確認します。

The selected task definition is not compatible with the selected compute strategy

このエラーは、起動タイプがクラスターのキャパシティタイプと一致しないタスク定義を選択した場合に発生します。クラスターに割り当てられたキャパシティープロバイダーと一致するタスク定義を選択する必要があります。

Unable to attach network interface to unused device index

このエラーは、`awsvpc` ネットワーキングタイプを使用しているときに、タスク用に十分な CPU/メモリがない場合に発生します。まず、インスタンスの CPU を確認します。詳細については、「Amazon EC2 インスタンスタイプ」の「Amazon EC2 インスタンスタイプの仕様」を参照してください。インスタンスの CPU 値をインスタンスの ENI の数で乗算します。その値をタスク定義で使用します。

AGENT

タスクを起動しようとしたコンテナインスタンスに、現在接続されていないエージェントがあります。タスク配置の待ち時間が長くならないように、リクエストは拒否されました。

切断されたエージェントをトラブルシューティングする方法については、「切断された Amazon ECS エージェントをトラブルシューティングするにはどうすればよいですか?」を参照してください。

Amazon ECS での ResourceInitializationError エラーのトラブルシューティング

以下に、`ResourceInitialization` エラーメッセージおよび、そのエラーを修正するために取れる措置を示します。

停止したタスクのエラーメッセージを AWS マネジメントコンソール で確認する方法については、「Amazon ECS の停止したタスクのエラーを表示する」を参照してください。

エラー

- The task cannot pull registry authentication from Amazon ECR. There is a connection issue between the task and Amazon ECR. Check your task network configuration.
- The task can't download the environment variable files from Amazon S3. There is a connection issue between the task and Amazon S3. Check your task network configuration.
- The task cannot pull secrets from AWS Systems Manager Parameter Store. Check your network connection between the task and AWS Systems Manager.
- The task can’t pull secrets from AWS Secrets Manager. There is a connection issue between the task and Secrets Manager. Check your task network configuration.
- The task can’t pull the secret from Secrets Manager. The task can't retrieve the secret with ARN ‘secretARN' from Secrets Manager. Check whether the secret exists in the specified Region.
- pull command failed: unable to pull secrets or registry auth Check your task network configuration.
- The task cannot find the Amazon CloudWatch log group defined in the task definition. There is a connection issue between the task and Amazon CloudWatch. ネットワーク設定を確認します。
- failed to initialize logging driver
- failed to invoke EFS utils commands to set up EFS volumes

The task cannot pull registry authentication from Amazon ECR. There is a connection issue between the task and Amazon ECR. Check your task network configuration.

このエラーは、タスクが Amazon ECR に接続できないことを示します。

タスクと Amazon ECR との間の接続を確認してください。詳細については、「Amazon ECS の停止したタスクの接続を検証する」を参照してください。

The task can't download the environment variable files from Amazon S3. There is a connection issue between the task and Amazon S3. Check your task network configuration.

このエラーは、タスクが Amazon S3 から環境設定ファイルをダウンロードできない場合に発生します。

タスクと Amazon S3 VPC エンドポイントとの間の接続を確認してください。詳細については、「Amazon ECS の停止したタスクの接続を検証する」を参照してください。

The task cannot pull secrets from AWS Systems Manager Parameter Store. Check your network connection between the task and AWS Systems Manager.

このエラーは、タスクが Systems Manager の認証情報を使用して、タスク定義で定義されたイメージをプルできない場合に発生します。

タスクと Systems Manager VPC エンドポイントとの間の接続を確認してください。詳細については、「Amazon ECS の停止したタスクの接続を検証する」を参照してください。

The task can't pull secrets from AWS Secrets Manager. There is a connection issue between the task and Secrets Manager. Check your task network configuration.

このエラーは、タスクが Secrets Manager の認証情報を使用して、タスク定義で定義されたイメージをプルできない場合に発生します。

エラーは、Systems Manager VPC エンドポイントとタスク間のネットワーク接続に問題があることを示します。

タスクとエンドポイント間の接続を確認する方法については、「Amazon ECS の停止したタスクの接続を検証する」を参照してください。

The task can't pull the secret from Secrets Manager. The task can't retrieve the secret with ARN '***secretARN***' from Secrets Manager. Check whether the secret exists in the specified Region.

このエラーは、タスクが Secrets Manager の認証情報を使用して、タスク定義で定義されたイメージをプルできない場合に発生します。

この問題は、次のいずれかの理由によって発生します。

エラーの原因	手順	
Secrets Manager VPC エンドポイントとタスク間のネットワーク接続の問題 エラーメッセージに次のいずれかの文字列が表示されている場合、問題の原因はネットワークです。 • dial tcp • dial udp • <ip>:<port>: i/o timeout • net/http: TLS handshake timeout	タスクと Secrets Manager エンドポイント間の接続を確認します。詳細については、「Amazon ECS の停止したタスクの接続を検証する」を参照してください。	

エラーの原因	手順	
• read: connection timed out • Client.Timeout exceeded while awaiting headers • net/http: request canceled while waiting for connection • signal: killed • context deadline exceeded		
タスク定義で定義されたタスク実行ロールに、Secrets Manager のアクセス許可がありません。	タスク実行ロールに、Secrets Manager に必要なアクセス許可を追加します。詳細については、「Secrets Manager または Systems Manager のアクセス許可」を参照してください。	
シークレット ARN が存在しません	Secrets Manager に ARN が存在することを確認します。イメージの表示については、「Secrets Manager デベロッパーガイド」の「Secrets Manager でシークレットを検索する」を参照してください。	

pull command failed: unable to pull secrets or registry auth Check your task network configuration.

このエラーは、タスクが Amazon ECR、Systems Manager、または Secrets Manager に接続できない場合に発生します。これはネットワークの設定ミスが原因です。

この問題を解決するには、タスクと Amazon ECR との間の接続を確認します。また、タスクとシークレットを保存するサービス (Systems Manager または Secrets Manager) との間の接続も確認する必要があります。詳細については、「Amazon ECS の停止したタスクの接続を検証する」を参照してください。

The task cannot find the Amazon CloudWatch log group defined in the task definition. There is a connection issue between the task and Amazon CloudWatch. ネットワーク設定を確認します。

このエラーは、タスクがタスク定義で定義した CloudWatch ロググループを見つけられない場合に発生します。

このエラーは、CloudWatch VPC エンドポイントとタスクとの間のネットワーク接続に問題があることを示します。

タスクとエンドポイント間の接続を確認する方法については、「Amazon ECS の停止したタスクの接続を検証する」を参照してください。

failed to initialize logging driver

このエラーは、タスクがタスク定義で定義した CloudWatch ロググループを見つけられない場合に発生します。

このエラーは、タスク定義内に CloudWatch グループが存在しないことを示します。

対象の CloudWatch を検索するには、次の手順を実行します。

1. 次のコマンドを実行して、タスク定義情報を取得します。

```
aws ecs describe-task-definition \
    --task-definition task-def-name
```

   各コンテナの出力を見て、awslogs-group 値を書き留めます。

```
"logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-group": "/ecs/example-group",
                    "awslogs-create-group": "true",
                    "awslogs-region": "us-east-1",
                    "awslogs-stream-prefix": "ecs"
                },
```

2. このグループが CloudWatch に存在することを確認します。詳細については、「Amazon CloudWatch Logs ユーザーガイド」の「ロググループとログストリームの操作」を参照してください。

この問題は、タスク定義で指定されたグループが正しくないことか、ロググループが存在しないことのいずれかです。

3. 問題を修正します。

問題	手順
タスク定義で正しくないロググループが指定されています。	コンテナ定義にロググループ設定が含まれるようにタスク定義を更新します。タスク定義の更新については、「Amazon Elastic Container Service API リファレンス」の「コンソールを使用した Amazon ECS タスク定義の更新」または「RegisterTaskDefinition」を参照してください。
ロググループが CloudWatch に存在しません	ロググループを作成します。詳細については、Amazon CloudWatch Logs ユーザーガイドの「CloudWatch Logs のロググループを作成する」を参照してください。

failed to invoke EFS utils commands to set up EFS volumes

以下の問題により、タスクに Amazon EFS ボリュームをマウントできない可能性があります。

- Amazon EFS ファイルシステムが正しく設定されていません。
- タスクに必要なアクセス許可がありません。
- ネットワークおよび VPC 構成に関連する問題があります。

この問題をデバッグして修正する方法については、AWS re:Post の 「AWS Fargate タスクに Amazon EFS ボリュームをマウントできない理由」を参照してください。

Amazon ECS ResourceNotFoundException エラーのトラブルシューティング

以下に、 `ResourceNotFoundException` エラーメッセージおよび、そのエラーを修正するために取れる措置を示します。

停止したタスクのエラーメッセージを AWS マネジメントコンソール で確認する方法については、「Amazon ECS の停止したタスクのエラーを表示する」を参照してください。

The task can't retrieve the secret with ARN '***sercretARN***' from AWS Secrets Manager. Check whether the secret exists in the specified Region.

このエラーは、タスクが Secrets Manager からシークレットを取得できない場合に発生します。つまり、タスク定義で指定されたシークレット (およびエラーメッセージに含まれるシークレット) は Secrets Manager に存在しないということです。

リージョンはエラーメッセージに含まれています。

Fetching secret data from AWS Secrets Manager in region *region*: secret *sercretARN*: ResourceNotFoundException: Secrets Manager can't find the specified secret.

シークレットの検索方法については、「AWS Secrets Manager ユーザーガイド」の「AWS Secrets Manager でシークレットを検索する」を参照してください。

次の表を使用して、エラーを特定して対処します。

問題	アクション	
シークレットは、タスク定義とは異なるリージョンにあります。	a. タスクと同じリージョンにシークレットを作成します。詳細については、「AWS Secrets Manager シークレットを作成する」を参照してください。 b. 新しいシークレットでタスク定義を更新します。詳細については、「Amazon Elastic Container Service API リファレンス」の「コンソールを使用した	

問題	アクション
	Amazon ECS タスク定義の更新」または「RegisterTaskDefinition」参照してください。
タスク定義に誤ったシークレット ARN が存在します。正しいシークレットは Secrets Manager に存在します。	正しいシークレットでタスク定義を更新します。詳細については、「Amazon Elastic Container Service API リファレンス」の「コンソールを使用した Amazon ECS タスク定義の更新」または「RegisterTaskDefinition」参照してください。
シークレットが存在しません。	a. タスクと同じリージョンにシークレットを作成します。詳細については、「AWS Secrets Manager シークレットを作成する」を参照してください。 b. 新しいシークレットでタスク定義を更新します。詳細については、「Amazon Elastic Container Service API リファレンス」の「コンソールを使用した Amazon ECS タスク定義の更新」または「RegisterTaskDefinition」参照してください。

Amazon ECS での SpotInterruption エラーのトラブルシューティング

`SpotInterruption` エラーの原因は、Fargate と EC2 によって異なります。

停止したタスクのエラーメッセージを AWS マネジメントコンソール で確認する方法については、「Amazon ECS の停止したタスクのエラーを表示する」を参照してください。

Fargate

`SpotInterruption` エラーは、Fargate Spot に容量がない場合や、Fargate がスポット容量を元に戻した場合に発生します。

タスクを複数のアベイラビリティーゾーンで実行することで、容量を増やすことができます。

EC2

このエラーは、使用可能なスポットインスタンスがない場合や、EC2 がスポットインスタンスの容量を元に戻した場合に発生します。

インスタンスを複数のアベイラビリティーゾーンで実行することで、容量を増やすことができます。

Amazon ECS での InternalError エラーのトラブルシューティング

適用先: Fargate

停止したタスクのエラーメッセージを AWS マネジメントコンソール で確認する方法については、「Amazon ECS の停止したタスクのエラーを表示する」を参照してください。

`InternalError` エラーは、ランタイムに関連しない予期せぬ内部エラーがエージェントに発生した場合に発生します。

このエラーは、プラットフォームバージョン `1.4` 以降を使用の場合にのみ発生します。

この問題のデバッグおよび修正方法については、Amazon ECS の停止したタスクのエラーメッセージ を参照してください。

Amazon ECS での OutOfMemoryError エラーのトラブルシューティング

以下は、OutOfMemoryError のエラーメッセージと、エラーを修正するために実行できるアクションの一部です。

停止したタスクのエラーメッセージを AWS マネジメントコンソール で確認する方法については、「Amazon ECS の停止したタスクのエラーを表示する」を参照してください。

container killed due to memory usage

このエラーは、コンテナ内のプロセスでタスク定義で割り当てられたメモリよりも多くのメモリを消費しているか、またはホストやオペレーティングシステムに制約があることが原因でコンテナが終了したときに発生します。

Amazon ECS での ContainerRuntimeError エラーのトラブルシューティング

以下は、ContainerRuntimeError のエラーメッセージと、エラーを修正するために実行できるアクションの一部です。

停止したタスクのエラーメッセージを AWS マネジメントコンソール で確認する方法については、「Amazon ECS の停止したタスクのエラーを表示する」を参照してください。

ContainerRuntimeError

このエラーは、エージェントが `containerd` がランタイム固有のオペレーションに関する予期しないエラーを受け取った場合に発生します。このエラーは通常、エージェントや `containerd` ランタイムの内部障害によって発生します。

このエラーは、プラットフォームバージョン `1.4.0` 以降 (Linux) または `1.0.0` 以降 (Windows) を使用している場合のみに発生します。

この問題をデバッグして修正する方法については、AWS re:Post の「Amazon ECS タスクが停止する理由」を参照してください。

Amazon ECS での ContainerRuntimeTimeoutError エラーのトラブルシューティング

以下は、ContainerRuntimeTimeoutError のエラーメッセージと、エラーを修正するために実行できるアクションの一部です。

停止したタスクのエラーメッセージを AWS マネジメントコンソール で確認する方法については、「Amazon ECS の停止したタスクのエラーを表示する」を参照してください。

Could not transition to running; timed out after waiting 1m or Docker timeout error

このエラーは、タイムアウト期間内にコンテナが `RUNNING`または`STOPPED`のどちらかの状態に移行できなかった場合に発生します。理由とタイムアウト値は、エラーメッセージに表示されます。

Amazon ECS での CannotStartContainerError エラーのトラブルシューティング

以下は、CannotStartContainerError のエラーメッセージと、エラーを修正するために実行できるアクションの一部です。

停止したタスクのエラーメッセージを AWS マネジメントコンソール で確認する方法については、「Amazon ECS の停止したタスクのエラーを表示する」を参照してください。

failed to get container status: *<reason>*

このエラーは、コンテナをスタートできない場合に発生します。

コンテナは、ここで指定したメモリを超えようとすると、停止されます。コンテナに提供するメモリを増やしてください。これは、タスク定義の `memory` パラメータです。詳細については、「the section called “メモリ”」を参照してください。

Amazon ECS での CannotStopContainerError エラーのトラブルシューティング

以下は、CannotStopContainerError のエラーメッセージと、エラーを修正するために実行できるアクションの一部です。

停止したタスクのエラーメッセージを AWS マネジメントコンソール で確認する方法については、「Amazon ECS の停止したタスクのエラーを表示する」を参照してください。

CannotStopContainerError

このエラーは、コンテナを停止できない場合に発生します。

この問題をデバッグして修正する方法については、AWS re:Post の「Amazon ECS タスクが停止する理由」を参照してください。

Amazon ECS での CannotInspectContainerError エラーのトラブルシューティング

以下は、CannotInspectContainerError のエラーメッセージと、エラーを修正するために実行できるアクションの一部です。

停止したタスクのエラーメッセージを AWS マネジメントコンソール で確認する方法については、「Amazon ECS の停止したタスクのエラーを表示する」を参照してください。

CannotInspectContainerError

このエラーは、コンテナエージェントがコンテナランタイムを通してコンテナを説明できない場合に発生します。

プラットフォームバージョン `1.3` 以前を使用している場合、Amazon ECS エージェントは Docker から理由を返します。

プラットフォームバージョン `1.4.0` 以降 (Linux) または `1.0.0` 以降 (Windows) を使用している場合、Fargate エージェントは `containerd` から理由を返します。

この問題をデバッグして修正する方法については、AWS re:Post の「Amazon ECS タスクが停止する理由」を参照してください。

Amazon ECS での CannotCreateVolumeError エラーのトラブルシューティング

以下は、CannotCreateVolumeError のエラーメッセージと、エラーを修正するために実行できるアクションの一部です。

停止したタスクのエラーメッセージを AWS マネジメントコンソール で確認する方法については、「Amazon ECS の停止したタスクのエラーを表示する」を参照してください。

CannotCreateVolumeError

このエラーは、エージェントがタスク定義に指定されているボリュームマウントを作成できない場合に発生します。

このエラーは、プラットフォームバージョン `1.4.0` 以降 (Linux) または `1.0.0` 以降 (Windows) を使用している場合のみに発生します。

この問題をデバッグして修正する方法については、AWS re:Post の「Amazon ECS タスクが停止する理由」を参照してください。

Amazon ECS での CannotPullContainer タスクエラー

次のエラーは、Amazon ECS が指定されたコンテナイメージを取得できないためにタスクを開始できなかったことを示しています。

> **Note**
>
> 1.4 Fargate プラットフォームバージョンでは、長いエラーメッセージが切り捨てられます。

停止したタスクのエラーメッセージを AWS マネジメントコンソール で確認する方法については、「Amazon ECS の停止したタスクのエラーを表示する」を参照してください。

> **Tip**
>
> Amazon ECS MCP サーバー と AI アシスタントを使用することで、イメージプルエラーの調査を自然言語で実行できます。

エラー

- The task can't pull the image. ロールにレジストリからイメージをプルするためのアクセス許可があることを確認します。
- The task cannot pull 'image-name' from the Amazon ECR repository 'repository URI'. There is a connection issue between the task and Amazon ECR. Check your task network configuration.
- The task can't pull the image. Check your network configuration
- CannotPullContainerError: プルイメージマニフェストが 5 回再試行されました: ref の解決に失敗しました
- API error (500): Get https://111122223333.dkr.ecr.us-east-1.amazonaws.com/v2/: net/http: request canceled while waiting for connection
- API エラー
- write /var/lib/docker/tmp/GetImageBlob111111111: no space left on device
- ERROR: toomanyrequests: Too Many Requests or You have reached your pull rate limit.
- Error response from daemon: Get url: net/http: request canceled while waiting for connection
- ref pull has been retried 1 time(s): failed to copy: httpReaderSeeker: failed open: unexpected status code
- pull access denied
- pull command failed: panic: runtime error: invalid memory address or nil pointer dereference
- error pulling image conf/error pulling image configuration
- コンテキストがキャンセルされました

The task can't pull the image. ロールにレジストリからイメージをプルするためのアクセス許可があることを確認します。

このエラーは、アクセス許可の問題により、タスク定義で指定されたイメージをタスクがプルできないことを示します。

この問題を解決するには。

1. イメージが*#####*に存在することを確認してください。イメージの表示の詳細については、「Amazon Elastic Container Registry ユーザーガイド」の「Amazon ECR でのイメージの詳細の表示」を参照してください。
2. *role-arn* にイメージをプルするための正しいアクセス許可があることを確認します。

   ロールを更新する方法については、「AWS Identity and Access Management ユーザーガイド」の「ロールに対するアクセス許可を更新する」を参照してください。

タスクには次のいずれかのロールを使用します。

- Fargate のタスクの場合、これはタスク実行ロールになります。Amazon ECR の追加のアクセス許可については、「インターフェイスエンドポイントのアクセス許可によって Amazon ECR イメージをプルする Fargate タスクです。」を参照してください。
- EC2 のタスクの場合、これはコンテナインスタンスロールになります。Amazon ECR の追加のアクセス許可については、「Amazon ECR のアクセス許可」を参照してください。

The task cannot pull '***image-name***' from the Amazon ECR repository '***repository URI***'. There is a connection issue between the task and Amazon ECR. Check your task network configuration.

このエラーは、タスクが Amazon ECR に接続できないことを示します。*##### URI* リポジトリへの接続を確認してください。

これらの問題を検証および解決する方法の詳細については、「Amazon ECS の停止したタスクの接続を検証する」を参照してください。

The task can't pull the image. Check your network configuration

このエラーは、タスクが Amazon ECR に接続できないことを示します。

これらの問題を検証および解決する方法の詳細については、「Amazon ECS の停止したタスクの接続を検証する」を参照してください。

CannotPullContainerError: プルイメージマニフェストが 5 回再試行されました: ref の解決に失敗しました

このエラーは、タスクがイメージをプルできないことを示します。

この問題を解決する方法は以下のとおりです。

- タスク定義で指定されたイメージがリポジトリ内のイメージと一致することを確認してください。
- Amazon ECS はイメージバージョンの安定性を強制します。元のイメージが使用できなくなった場合、このエラーが発生します。イメージタグは、この挙動の執行プロセスの一部です。タグとして :latest を使用するタスク定義のイメージを、特定のバージョンに変更してください。詳細については、「コンテナイメージの解決」を参照してください。

これらの問題を検証および解決する方法の詳細については、「Amazon ECS の停止したタスクの接続を検証する」を参照してください。

API error (500): Get https://111122223333.dkr.ecr.us-east-1.amazonaws.com/v2/: net/http: request canceled while waiting for connection

このエラーは、インターネットへのルートがないため、接続がタイムアウトしたことを示します。

この問題を解決するには、以下ができます。

- パブリックサブネットのタスクでは、タスクの起動時に [Auto-assign public IP] (自動割り当てパブリック IP)を [ENABLED] (有効)に指定する必要があります。詳細については、「Amazon ECS タスクとしてのアプリケーションの実行」を参照してください。
- プライベートサブネットのタスクでは、タスク起動時に [Auto-assign public IP] (自動割り当てパブリック IP) を [DISABLED] (無効) に指定し、VPC の NAT ゲートウェイを設定してリクエストをインターネットにルートします。詳細については、Amazon VPC ユーザーガイド の NAT ゲートウェイ を参照してください。

API エラー

このエラーは、Amazon ECR エンドポイントとの接続に問題があることを示します。

この問題を解決する方法については、サポート ウェブサイトの「Amazon ECS の Amazon ECR エラー "CannotPullContainerError: API error" を解決するには」を参照してください。

write /var/lib/docker/tmp/***GetImageBlob111111111***: no space left on device

このエラーは、ディスク容量が不足していることを示しています。

この問題を解決するには、ディスク容量を解放します。

Amazon ECS 最適化 AMI を使用している場合は、次のコマンドを使用してファイルシステムで 20 個の最大ファイルを取得できます。

```
du -Sh / | sort -rh | head -20
```

出力例:

```
5.7G    /var/lib/docker/
containers/50501b5f4cbf90b406e0ca60bf4e6d4ec8f773a6c1d2b451ed8e0195418ad0d2
1.2G    /var/log/ecs
594M    /var/lib/docker/devicemapper/mnt/
c8e3010e36ce4c089bf286a623699f5233097ca126ebd5a700af023a5127633d/rootfs/data/logs
...
```

場合によっては、実行中のコンテナによってルートボリュームがいっぱいになる可能性があります。コンテナが json-file 制限のないデフォルトの max-size ログドライバーを使用している場合、ログファイルが使用されているスペースの大半を占めている可能性があります。docker ps コマンドを使用して、出力からコンテナ ID にディレクトリ名をマッピングすることによって、どのコンテナがスペースを使用しているかを確認します。例えば、次のようになります。

```
CONTAINER ID   IMAGE                                COMMAND             CREATED
 STATUS              PORTS                               NAMES
50501b5f4cbf   amazon/amazon-ecs-agent:latest   "/agent"            4 days ago
 Up 4 days                                           ecs-agent
```

デフォルトでは、json-file ログドライバーを使用する場合、Docker はすべてのコンテナの標準出力 (および標準エラー) をキャプチャし、JSON 形式を使用してファイルに書き込みます。ログドライバーオプションとして max-size を設定できます。これにより、ログファイルの容量が大きくなりすぎるのを防ぐことができます。詳細については、Docker ドキュメントの「JSON ファイルロギングドライバー」を参照してください。

このオプションの使用方法を示すコンテナ定義のスニペットを次に示します。

```
{
    "log-driver": "json-file",
    "log-opts": {
        "max-size": "256m"
    }
}
```

コンテナログのディスク容量使用量が大きすぎる場合、awslogs ログドライバーを使用することもできます。awslogs ログドライバーがログを CloudWatch に送信します。これによりコンテナインスタンスのコンテナログに使用されるディスク容量が解放されます。詳細については、「Amazon ECS ログを CloudWatch に送信する」を参照してください。

Docker がアクセスできるディスクサイズに更新する必要がある場合があります。

詳細については、「CannotPullContainerError: no space left on device」を参照してください。

ERROR: toomanyrequests: Too Many Requests or You have reached your pull rate limit.

このエラーは、Docker Hub のレート制限があることを示します。

次のエラーのいずれかが表示された場合は、Docker Hub のレート制限に達している可能性があります。

Docker Hub のレート制限の詳細については、[Understanding Docker Hub rate limiting] (Docker ハブのレート制限を理解する) を参照してください。

Docker Hub のレート制限を上げ、コンテナインスタンスの Docker プルを認証する必要がある場合は、「Private registry authentication for container instances」を参照してください。

Error response from daemon: Get ***url***: net/http: request canceled while waiting for connection

このエラーは、インターネットへのルートがないため、接続がタイムアウトしたことを示します。

この問題を解決するには、以下ができます。

- パブリックサブネットのタスクでは、タスクの起動時に [Auto-assign public IP] (自動割り当てパブリック IP)を [ENABLED] (有効)に指定する必要があります。詳細については、「Amazon ECS タスクとしてのアプリケーションの実行」を参照してください。
- プライベートサブネットのタスクでは、タスク起動時に [Auto-assign public IP] (自動割り当てパブリック IP) を [DISABLED] (無効) に指定し、VPC の NAT ゲートウェイを設定してリクエストをインターネットにルートします。詳細については、Amazon VPC ユーザーガイド の NAT ゲートウェイ を参照してください。

ref pull has been retried 1 time(s): failed to copy: httpReaderSeeker: failed open: unexpected status code

このエラーは、イメージのコピー時に問題があったことを示します。

この問題を解決するには、次のいずれかの記事を確認してください。

- Fargate タスクについては、Fargate Amazon ECSタスクの「cannotpullcontainererror」エラーの解決方法を参照してください。
- その他のタスクについては、Amazon ECS タスクの「cannotpullcontainererror」エラーの解決方法を参照してください。

pull access denied

このエラーは、イメージにアクセスできないことを示します。

この問題を解決するには、Amazon ECR で Docker クライアントを認証する必要がある場合があります。詳細については、「Amazon ECR ユーザーガイド」の「プライベートレジストリ認証」を参照してください。

pull command failed: panic: runtime error: invalid memory address or nil pointer dereference

このエラーは、メモリアドレスが無効であるか、ポインターデリファレンスが nil であるため、イメージにアクセスできないことを示しています。

この問題を解決するには。

- Amazon S3 に到達するためのセキュリティグループのルールがあることを確認してください。
- ゲートウェイエンドポイントを使用するときは、エンドポイントにアクセスするためのルートをルートテーブルに追加する必要があります。

error pulling image conf/error pulling image configuration

このエラーは、レート制限に達したか、ネットワークエラーが発生したことを示します。

この問題を解決するには、「Amazon ECS EC2 起動タイプタスクで CannotPullContainerError エラーを解決する方法」を参照してください。

コンテキストがキャンセルされました

このエラーは、コンテキストがキャンセルされたことを示しています。

このエラーの一般的な原因は、タスクが使用している VPC にコンテナイメージを Amazon ECR から取得するルートがないためです。

Amazon ECS の停止したタスクの接続を検証する

ネットワーク接続の問題により、タスクが停止することがあります。断続的な問題である可能性がありますが、タスクがエンドポイントに接続できないことが原因である可能性が最も高いです。

タスクの接続をテストする

`AWSSupport-TroubleshootECSTaskFailedToStart` ランブックを使用して、タスクの接続をテストできます。ランブックを使用する場合は、次のリソースに関する情報が必要です。

- タスク ID

  最後に失敗したタスク のID を使用します。
- タスクが属していたクラスター

ランブックの使用方法については、「AWS Systems Manager Automation ランブックリファレンス」の「AWSSupport-TroubleshootECSTaskFailedToStart」を参照してください。

ランブックはタスクを分析します。タスクの開始を妨げる可能性のある次の問題については、「出力」セクションで結果を確認できます。

- 設定済みのコンテナレジストリーへのネットワーク接続
- VPC エンドポイント接続
- セキュリティグループのルール設定

VPC エンドポイントの問題を修正する

AWSSupport-TroubleshootECSTaskFailedToStart ランブックの結果に VPC エンドポイントの問題が表示された場合は、次の設定を確認します。

- エンドポイントと VPC エンドポイントを作成する VPC では、プライベート DNS を使用する必要があります。
- タスクが接続できないサービスの AWS PrivateLink エンドポイントが、タスクと同じ VPC 内にあることを確認してください。詳細については、以下のいずれかを参照してください。

サービス	サービスの VPC エンドポイント情報	
Amazon ECR	Amazon ECS およびインターフェイスの VPC エンドポイント (AWS PrivateLink)	
Systems Manager	Systems Manager のために VPC エンドポイントを使用して EC2 インスタンスのセキュリティを強化する	
Secrets Manager	AWS Secrets Manager VPC エンドポイントを使用する	
CloudWatch	CloudWatch VPC エンドポイント	

サービス	サービスの VPC エンドポイント情報	
Amazon S3	AWS PrivateLinkAmazon S3 用	

- ポート 443 DNS (TCP) トラフィックで HTTPS を許可するタスクサブネットのアウトバウンドルールを設定します。詳細については、「Amazon Elastic Compute Cloud ユーザーガイド」の「セキュリティグループのルールを設定する」を参照してください。
- カスタムネームドメインサーバーを使用する場合は、DNS クエリの設定を確認します。クエリは、53 番ポートで外部への通信を行うことができ、UDP および TCP プロトコルを使用する必要があります。また、443 番ポートでの HTTPS 通信も必要です。詳細については、「Amazon Elastic Compute Cloud ユーザーガイド」の「セキュリティグループのルールを設定する」を参照してください。
- サブネットにネットワーク ACL がある場合は、次の ACL ルールが必要です。
  - ポート 1024-65535 でトラフィックを許可するアウトバウンドルール
  - ポート 443 の TCP トラフィックを許可するインバウンドルール

  ルールの設定方法については、「Amazon Virtual Private Cloud ユーザーガイド」の「ネットワーク ACL を使用してサブネットへのトラフィックを制御する」を参照してください。

ネットワークの問題を修正する

`AWSSupport-TroubleshootECSTaskFailedToStart` ランブックの結果にネットワークの問題が表示されたら、次の設定を確認します。

パブリックサブネットで awsvpc ネットワークモードを使用するタスク

ランブックに基づいて次の設定を実行します。

- パブリックサブネットのタスクでは、タスクの起動時に [Auto-assign public IP] (自動割り当てパブリック IP)を [ENABLED] (有効)に指定する必要があります。詳細については、「Amazon ECS タスクとしてのアプリケーションの実行」を参照してください。
- インターネットトラフィックを処理するにはゲートウェイが必要です。タスクサブネットのルートテーブルには、ゲートウェイへのトラフィック用のルートが必要です。

  詳細については、「Amazon Virtual Private Cloud ユーザーガイド」の「ルートテーブルのルートの追加と削除」を参照してください。

ゲートウェイタイプ	ルートテーブルの送信先	ルートテーブルのターゲット
NAT	0.0.0.0/0	NAT ゲートウェイ ID
インターネットゲートウェイ	0.0.0.0/0	インターネットゲートウェイ ID

- タスクサブネットにネットワーク ACL がある場合は、次の ACL ルールが必要です。
  - ポート 1024-65535 でトラフィックを許可するアウトバウンドルール
  - ポート 443 の TCP トラフィックを許可するインバウンドルール

  ルールの設定方法については、「Amazon Virtual Private Cloud ユーザーガイド」の「ネットワーク ACL を使用してサブネットへのトラフィックを制御する」を参照してください。

プライベートサブネットで awsvpc ネットワークモードを使用するタスク

ランブックに基づいて次の設定を実行します。

- タスクの起動時に、[自動割り当てパブリック IP] で [無効] を選択します。
- リクエストがインターネットにルーティンされるように VPC の NAT ゲートウェイを設定します。詳細については、「Amazon Virtual Private Cloud ユーザーガイド」の「NAT ゲートウェイ」を参照してください。
- タスクサブネットのルートテーブルには、NAT ゲートウェイへのトラフィック用のルートが必要です。

  詳細については、「Amazon Virtual Private Cloud ユーザーガイド」の「ルートテーブルのルートの追加と削除」を参照してください。

ゲートウェイタイプ	ルートテーブルの送信先	ルートテーブルのターゲット
NAT	0.0.0.0/0	NAT ゲートウェイ ID

- タスクサブネットにネットワーク ACL がある場合は、次の ACL ルールが必要です。
  - ポート 1024-65535 でトラフィックを許可するアウトバウンドルール
  - ポート 443 の TCP トラフィックを許可するインバウンドルール

ルールの設定方法については、「Amazon Virtual Private Cloud ユーザーガイド」の「ネットワーク ACL を使用してサブネットへのトラフィックを制御する」を参照してください。

パブリックサブネットで awsvpc ネットワークモードを使用しないタスク

ランブックに基づいて次の設定を実行します。

- クラスターの作成時に、[Amazon EC2 インスタンスのネットワーク] の [IP の自動割り当て] で、[オンにする] を選択します。

  このオプションにより、インスタンスのプライマリネットワークインターフェイスにパブリック IP アドレスを割り当てます。
- インターネットトラフィックを処理するにはゲートウェイが必要です。インスタンスサブネットのルートテーブルには、ゲートウェイへのトラフィック用のルートが必要です。

  詳細については、「Amazon Virtual Private Cloud ユーザーガイド」の「ルートテーブルのルートの追加と削除」を参照してください。

ゲートウェイタイプ	ルートテーブルの送信先	ルートテーブルのターゲット
NAT	0.0.0.0/0	NAT ゲートウェイ ID
インターネットゲートウェイ	0.0.0.0/0	インターネットゲートウェイ ID

- インスタンスサブネットにネットワーク ACL がある場合は、次の ACL ルールが必要です。
  - ポート 1024-65535 でトラフィックを許可するアウトバウンドルール
  - ポート 443 の TCP トラフィックを許可するインバウンドルール

  ルールの設定方法については、「Amazon Virtual Private Cloud ユーザーガイド」の「ネットワーク ACL を使用してサブネットへのトラフィックを制御する」を参照してください。

プライベートサブネットで awsvpc ネットワークモードを使用しないタスク

ランブックに基づいて次の設定を実行します。

- クラスターの作成時に、[Amazon EC2 インスタンスのネットワーク] の [IP の自動割り当て] で、[オフにする] を選択します。

- リクエストがインターネットにルーティンされるように VPC の NAT ゲートウェイを設定します。詳細については、Amazon VPC ユーザーガイド の NAT ゲートウェイ を参照してください。
- インスタンスサブネットのルートテーブルには、NAT ゲートウェイへのトラフィック用のルートが必要です。

  詳細については、「Amazon Virtual Private Cloud ユーザーガイド」の「ルートテーブルのルートの追加と削除」を参照してください。

ゲートウェイタイプ	ルートテーブルの送信先	ルートテーブルのターゲット
NAT	0.0.0.0/0	NAT ゲートウェイ ID

- タスクサブネットにネットワーク ACL がある場合は、次の ACL ルールが必要です。
  - ポート 1024-65535 でトラフィックを許可するアウトバウンドルール
  - ポート 443 の TCP トラフィックを許可するインバウンドルール

  ルールの設定方法については、「Amazon Virtual Private Cloud ユーザーガイド」の「ネットワーク ACL を使用してサブネットへのトラフィックを制御する」を参照してください。

Amazon ECS タスクの IAM ロールリクエストの表示

IAM ロールでタスク認証情報にプロバイダーを使用すると、プロバイダーは監査ログに保存されたものをリクエストします。監査ログは、コンテナエージェントログと同じログローテーション設定を継承します。コンテナエージェントの設定変数 `ECS_LOG_ROLLOVER_TYPE`、`ECS_LOG_MAX_FILE_SIZE_MB`、および `ECS_LOG_MAX_ROLL_COUNT` を設定して、監査ログの動作に影響を与えることもできます。詳細については、「Amazon ECS コンテナエージェントのログ設定パラメータ」を参照してください。

コンテナエージェントバージョン 1.36.0 以降の場合、監査ログは `/var/log/ecs/audit.log` にあります。ログがローテーションされると、*YYYY*-*MM*-*DD*-*HH* 形式のタイムスタンプがログファイル名の最後に追加されます。

コンテナエージェントバージョン 1.35.0 以前の場合、監査ログは `/var/log/ecs/audit.log.`*YYYY*-*MM*-*DD*-*HH* にあります。

ログエントリの形式は以下のとおりです。

- タイムスタンプ

- HTTP レスポンスコード
- リクエスト元の IP アドレスとポート番号
- 認証情報プロバイダーの相対 URI
- リクエストを行ったユーザーエージェント
- リクエスト元のコンテナが属するタスクの ARN
- `GetCredentials` API 名とバージョン番号
- コンテナインスタンスが登録されている Amazon ECS クラスターの名前
- コンテナインスタンス ARN

ログファイルを表示するには、次のコマンドが使用できます。

```
cat /var/log/ecs/audit.log.2016-07-13-16
```

出力:

```
2016-07-13T16:11:53Z 200 172.17.0.5:52444 "/v1/credentials" "python-requests/2.7.0
 CPython/2.7.6 Linux/4.4.14-24.50.amzn1.x86_64" TASK_ARN GetCredentials
 1 CLUSTER_NAME CONTAINER_INSTANCE_ARN
```

Amazon ECS のサービスイベントメッセージを表示する

サービスに関する問題をトラブルシューティングする際は、まず最初にサービスイベントログの診断情報を確認します。サービスイベントは、`DescribeServices` API、AWS CLI、または AWS マネジメントコンソール を使って表示できます。

Amazon ECS API を使用して、サービスイベントメッセージを表示する場合、サービススケジューラからのイベントのみが返されます。これには、最新のタスク配置とインスタンスの健全性イベントが含まれます。ただし、Amazon ECS コンソールには、次のソースからのサービスイベントが表示されます。

- Amazon ECS サービススケジューラからのタスク配置およびインスタンスのヘルスイベント。これらのイベントには、service ***(service-name)*** のプレフィックスがついています。このイベントビューが役立つ情報を提供するために、最新の 100 件のイベントのみを表示します。重複したサービスイベントメッセージは、原因が解決するか、6 時間が経過するまで表示されません。6 時間以内に原因が解決されない場合、その原因に関する別のサービスイベントメッセージが表示されます。

- サービスの自動スケーリングイベント。これらのイベントにはMessageというプレフィックスが付き、サービスが Application Auto Scaling スケーリングポリシーで構成されている場合にのみ発生します。

> **Tip**
>
> Amazon ECS MCP サーバー と AI アシスタントを使用することで、サービスイベントの分析を自然言語で実行できます。

現在のサービスイベントメッセージを表示するには、次の手順を実行します。

Console

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。
3. [Clusters] (クラスター) ページで、クラスターを選択します。
4. 検査するサービスを選択します。
5. [イベント] タブでメッセージを表示します。

AWS CLI

指定したサービスのサービスイベントメッセージを表示するには、describe-service コマンドを使用します。

次の AWS CLI 例では、*default* クラスター内の *service-name* サービスが記述されます。ここには、最新のサービスイベントメッセージが表示されます。

```
aws ecs describe-services \
    --cluster default \
    --services service-name \
    --region us-west-2
```

Amazon ECS のサービスイベントメッセージ

以下は、Amazon ECS コンソールで確認できる可能性があるサービスイベントメッセージの例です。

サービス (*service-name*) が定常状態に達しました。

サービススケジューラは、サービスが正常で、必要な数のタスクが定常状態になったときに、`service (service-name) has reached a steady state.` サービスイベントを送信します。

サービススケジューラは定期的にステータスを報告するため、このメッセージを複数回受信する場合があります。

サービス (*service-name*) で、すべての要件を満たしたコンテナインスタンスがないため、タスクを配置できませんでした。

サービススケジューラは、別のタスクを追加するために利用可能なリソースが見つからなかったときに、このイベントメッセージを送信します。以下に示しているのは、その考えられる原因です。

キャパシティープロバイダーを使用して、EC2 インスタンスを自動的にスケーリングします。詳細については、「EC2 ワークロード用の Amazon ECS キャパシティプロバイダー」を参照してください。

> キャパシティープロバイダーを使用する場合は、キャパシティープロバイダー戦略を渡すか、クラスターに関連付けられたデフォルトのキャパシティープロバイダー戦略があり、起動タイプとキャパシティープロバイダー戦略を入力として渡していないことを確認してください

クラスターにコンテナインスタンスが見つかなかった

> タスクを実行しようとしているクラスターにコンテナインスタンスが登録されていない場合は、このエラーが返されます。コンテナインスタンスをクラスターに追加する必要があります。詳細については、「Amazon ECS Linux コンテナインスタンスの起動」を参照してください。

ポートが足りない

> タスクで固定ホストポートマッピングを使用している場合 (タスクでウェブサーバーのホスト上のポート 80 を使用している場合など)、1 つのコンテナが一度に使用できるホストポートは 1 つのみであるため、タスクごとに 1 つ以上のコンテナインスタンスが必要です。コンテナインスタンスをクラスターに追加するか、タスクの必要数を減らす必要があります。

登録されたポートが多すぎる

> タスク配置に最も近いコンテナインスタンスは、コンテナインスタンスごとに許可される最大予約ホストポート数である 100 を超えることはできません。ホストポートの動的マッピングを使用すると、問題を解決できる場合があります。

ポートは既に使用中です

このタスクのタスク定義は、選択されたコンテナインスタンスで既に実行されているタスクと同じポートをポートマッピングで使用します。サービスイベントメッセージは、次のメッセージの一部としてコンテナインスタンス ID を本来選択していました。

```
The closest matching container-instance is already using a port required by your
 task.
```

メモリが足りない

タスク定義で 1,000 MiB のメモリを指定しており、クラスター内の各コンテナインスタンスのメモリが 1,024 MiB の場合、コンテナインスタンスごとにこのタスクのコピーを 1 つのみ実行できます。タスク定義でメモリを減らしてコンテナインスタンスごとに複数のタスクを起動できるようにするか、クラスターで起動するコンテナインスタンスを増やすことができます。

> **Note**
>
> リソースの使用率を最大化することを目的に、特定のインスタンスタイプにおいて、タスクにできるだけ多くのメモリを提供する場合には、「Amazon ECS Linux コンテナインスタンスのメモリを予約する」を参照してください。

CPU が足りない

コンテナインスタンスには、CPU コアごとに 1,024 個の CPU ユニットがあります。タスク定義で 1,000 個の CPU ユニットを指定しており、クラスター内の各コンテナインスタンスの CPU ユニットが 1,024 個である場合、コンテナインスタンスごとにこのタスクのコピーを 1 つのみ実行できます。タスク定義で CPU ユニット数を減らしてコンテナインスタンスごとに複数のタスクを起動できるようにするか、クラスターで起動するコンテナインスタンスを増やすことができます。

十分な数の ENI アタッチメントポイントを利用できない

awsvpc ネットワークモードを使用するタスクには、それぞれ独自の Elastic Network Interface (ENI) が提供されます。この ENI は、ENI をホストするコンテナインスタンスに添付されています。Amazon EC2 インスタンスは添付できる ENI の数には制限があり、クラスター内に利用可能なENI 容量があるコンテナインスタンスはありません。

個別のコンテナインスタンスの ENI 制限は、以下の条件によって異なります。

- awsvpcTrunking アカウント設定をオプトインしていない場合、各コンテナインスタンスの ENI 制限は、インスタンスタイプによって異なります。詳細については、Amazon EC2 ユーザーガイドの「各インスタンスタイプのネットワークインターフェイスあたりの IP アドレス」を参照してください。
- awsvpcTrunking アカウント設定をオプトインしているが、オプトイン後にサポートされているインスタンスタイプを使用して新しいコンテナインスタンスを起動していない場合、各コンテナインスタンスの ENI 制限はデフォルト値のままです。詳細については、Amazon EC2 ユーザーガイドの「各インスタンスタイプのネットワークインターフェイスあたりの IP アドレス」を参照してください。
- awsvpcTrunking アカウント設定をオプトインしていて、かつオプトイン後にサポートされているインスタンスタイプを使用して新しいコンテナインスタンスを起動している場合、追加の ENI オプトインを利用できます。詳細については、「Amazon ECS コンテナネットワークインターフェイスの増加でサポートされるインスタンス」を参照してください。

  awsvpcTrunking アカウント設定のオプトインの詳細については、「Amazon ECS Linux コンテナインスタンスのネットワークインターフェイスを増やす」を参照してください。

  クラスターにコンテナインスタンスを追加することで、利用できるネットワークアダプタの数を増やすことができます。

コンテナインスタンスに必須の属性がない

一部のタスク定義パラメータでは、特定バージョンの Docker リモート API をコンテナインスタンスにインストールする必要があります。ロギングドライバーオプションなどのその他のパラメータでは、コンテナインスタンスに ECS_AVAILABLE_LOGGING_DRIVERS エージェント設定変数を使用して、それらのロギングドライバーを登録する必要があります。タスク定義に特定のコンテナインスタンス属性を必要とするパラメータが含まれており、この要件を満たすことができるコンテナインスタンスがない場合、そのタスクは配置できません。

このエラーの一般的な原因は、サービスが awsvpc ネットワークおよび EC2 を使用している場合です。指定したクラスターには、サービスの作成時に awsvpcConfiguration で指定されたものと同じサブネットにコンテナインスタンスが登録されていません。

トラブルシューティングには、AWSSupport-TroubleshootECSContainerInstance ランブックを使用できます。ランブックでは、インスタンスのユーザーデータに正しいクラスター情報が含まれているかどうか、インスタンスプロファイルに必要なアクセス権限が含まれているかどうか、およびネットワーク設定の問題が確認されます。詳細については、「AWS Systems Manager オートメーションランブックリファレンスユーザーガイド」の「AWSSupport-TroubleshootECSContainerInstance」を参照してください。

特定のタスク定義パラメータとエージェント設定変数に必要な属性の詳細については、「Fargate での Amazon ECS タスク定義パラメータ」と「Amazon ECS コンテナエージェントの設定」を参照してください。

サービス (*service-name*) で、すべての要件を満たしたコンテナインスタンスがないため、タスクを配置できませんでした。container-instance-id に最も近い *container-instance-id* には、使用可能な CPU ユニットがありません。

タスク配置に最も一致するコンテナインスタンスに、タスク定義の要件を満たす十分な CPU ユニットがありません。タスク定義のタスクサイズとコンテナ定義の両方のパラメータで、CPU の要件を確認します。

サービス (*service-name*) で、すべての要件を満たしたコンテナインスタンスがないため、タスクを配置できませんでした。container-instance-id に最も近い *container-instance-id* で、エラー「AGENT」が発生しました。

タスク配置に最も一致するコンテナインスタンス上の Amazon ECS コンテナエージェントが切断されています。コンテナインスタンスに SSH で接続できる場合は、エージェントログを調べることができます。詳細については、「Amazon ECS コンテナエージェントのログ設定パラメータ」を参照してください。エージェントがインスタンスで実行されていることも確認する必要があります。Amazon ECS に最適化された AMI を使用している場合、次のコマンドでエージェントを停止して再開始する試みができます。

- Amazon ECS に最適化された Amazon Linux 2 AMI および Amazon ECS に最適化された Amazon Linux 2023 AMI の場合

```
sudo systemctl restart ecs
```

- Amazon ECS に最適化された Amazon Linux AMI の場合:

```
sudo stop ecs && sudo start ecs
```

サービス (*service-name*) (タスク *task-id*) (インスタンス *instance-id*) は、(reason インスタンスが、少なくとも UnhealthyThreshold 回のヘルスチェックに連続して失敗した) ため、(elb *elb elb-name*) で正常な状態になっていません

このサービスはロードバランサーに登録されており、ロードバランサーのヘルスチェックは失敗しています。メッセージには、どの特定のタスクがヘルスチェックに失敗しているかを識別するのに役立つタスク ID が含まれています。詳細については、「Amazon ECS のサービスロードバランサーのトラブルシューティング」を参照してください。

サービス (*service-name*) は、一貫してタスクを正常に開始できません。

このサービスには、連続して試行された後開始に失敗したタスクがあります。この時点で、サービススケジューラによって再試行間隔が段階的に増加し始めます。タスクの起動に失敗している理由をトラブルシューティングする必要があります。詳細については、「Amazon ECS サービスのスロットルロジック」を参照してください。

更新されたタスク定義などでサービスが更新された後、サービススケジューラは正常な動作を再開します。

サービス (*service-name*) オペレーションは抑制されています。後で再試行します。

このサービスは、API スロットルの制限により、これ以上のタスクを起動できません。サービススケジューラが追加のタスクを起動できるようになると、再開します。

API レート制限クォータの引き上げをリクエストするには、[AWS サポート Center (センター)] の ページを開き、必要に応じてサインインし、[Create case (ケースを作成する)] を選択します。[Service Limit increase] (サービス制限の緩和) を選択します。フォームに入力して送信します。

サービス (*service-name*) は、サービスデプロイメント構成のため、デプロイメント中にタスクを停止または開始できませんでした。minimumHealthyPercent または maximumPercent の値を更新してから、もう一度試してください。

このサービスは、デプロイメント構成であるため、サービスのデプロイメント中にタスクを停止または開始できません。デプロイ設定は、サービスの作成時に定義される `minimumHealthyPercent` 値と `maximumPercent` 値で構成されます。これらの値は既存のサービスでも更新できます。

`minimumHealthyPercent` は、デプロイ中またはコンテナインスタンスがドレインしているときに、サービスに対して実行する必要があるタスク数の下限を表します。これは、サービスに必要なタスク数に対するパーセンテージです。この値は切り上げられます。例えば、最小正常率が `50` で、必要なタスク数が 4 の場合、スケジューラは 2 つの新しいタスクを開始する前に既存のタスクを 2 つ

停止できます。同様に、最小正常率が 75% で、必要なタスク数が 2 の場合、結果の値も 2 であるため、スケジューラはタスクを停止できません。

maximumPercent は、デプロイ中またはコンテナインスタンスがドレインしているときに、サービスに対して実行する必要があるタスク数の上限を表します。これは、サービスに必要なタスク数に対するパーセンテージです。この値は切り捨てられます。例えば、最大パーセンテージが 200 で、目的のタスク数が 4 の場合、スケジューラは既存のタスクを 4 つ停止する前に 4 つの新しいタスクを開始できます。同様に、最大比率が 125 で、必要なタスク数が 3 の場合、結果の値も 3 であるため、スケジューラはタスクを開始できません。

最小正常率または最大正常率を設定するときは、デプロイメントがトリガーされたときにスケジューラが 1 つ以上のタスクを停止または開始できることを確認する必要があります。

サービス [*service-name(####-#)*] がタスクを配置できませんでした。理由: 同時に実行できるタスク数の上限に達しました

エラーの原因となったリソースに対して、クォータの引き上げをリクエストすることができます。詳細については、「Service Quotas」を参照してください。クォータの引き上げをリクエストするには、「Service Quotas ユーザーガイド」の「クォータ引き上げリクエスト を参照してください。

サービス [*service-name(####-#)*] がタスクを配置できませんでした。理由: 内部エラー。

このエラーが表示される理由として考えられるものは、次のとおりです。

あるサブネットがサポートされていないアベイラビリティーゾーンにあるため、サービスはタスクを開始できません。

サポートされた Fargate リージョンおよびアベイラビリティーゾーンの情報については、the section called “AWS Fargate リージョン” を参照してください。

サブネットのアベイラビリティーゾーンを確認する方法については、「Amazon VPC ユーザーガイド」の「サブネットの確認」を参照してください。

サービス [*service-name(####-#)*] がタスクを配置できませんでした。理由:リクエストされた CPU 構成が制限を超えています。

エラーの原因となったリソースに対して、クォータの引き上げをリクエストすることができます。詳細については、「Service Quotas」を参照してください。クォータの引き上げをリクエストするには、「Service Quotas ユーザーガイド」の「クォータ引き上げリクエスト を参照してください。

サービス [*service-name(####-#)*] がタスクを配置できませんでした。理由:リクエストされたメモリ構成が制限を超えています。

エラーの原因となったリソースに対して、クォータの引き上げをリクエストすることができます。詳細については、「Service Quotas」を参照してください。クォータの引き上げをリクエストするには、「Service Quotas ユーザーガイド」の「クォータ引き上げリクエスト を参照してください。

サービス [*service-name(####-#)*] がタスクを配置できませんでした。理由: 同時に実行できる vCPU の上限数に達しました

AWS Fargate は、タスク数ベースのクォータから vCPU ベースのクォータに移行しています。

Fargate の vCPU ベースのクォータの引き上げをリクエストできます。詳細については、「Service Quotas」を参照してください。Fargate クォータの引き上げをリクエストするには、「Service Quotas ユーザーガイド」の「クォータ引き上げをリクエストする」を参照してください。

タスクセット (*taskSet-ID*) がスケールインできなかったため、サービス (*service-name*) は定常状態に到達できませんでした。理由: 保護されているタスクの数が、必要なタスク数を超えています

サービスに、必要なタスク数よりも多くの保護タスクがあります。次のいずれかを行うことができます。

- 現在のタスクの保護が期限切れになり、タスクを終了できるようになるまでお待ちください。
- どのタスクを停止できるかを判断し、UpdateTaskProtection API で protectionEnabled オプションを false に設定し、これらのタスクに対する保護を設定解除します。
- サービスの必要なタスク数を増やして、保護されているタスクの数よりも多くします。

サービス (*service-name*) が定常状態に到達できませんでした。理由:キャパシティープロバイダーにコンテナインスタンスが見つかりませんでした。

サービススケジューラは、別のタスクを追加するために利用可能なリソースが見つからなかったときに、このイベントメッセージを送信します。以下に示しているのは、その考えられる原因です。

クラスターに関連付けられたキャパシティプロバイダーがありません

クラスターにキャパシティプロバイダーが関連付けられていることを確認するには、describe-services を使用します。サービスのキャパシティプロバイダー戦略を更新できます。

キャパシティプロバイダーに利用可能なキャパシティがあることを確認します。EC2 の場合は、コンテナインスタンスがタスク定義の要件を満たしていることを確認してください。

クラスターにコンテナインスタンスが見つかなかった

タスクを実行しようとしているクラスターにコンテナインスタンスが登録されていない場合は、このエラーが返されます。コンテナインスタンスをクラスターに追加する必要があります。詳細については、「Amazon ECS Linux コンテナインスタンスの起動」を参照してください。

ポートが足りない

タスクで固定ホストポートマッピングを使用している場合 (タスクでウェブサーバーのホスト上のポート 80 を使用している場合など)、タスクごとに 1 つ以上のコンテナインスタンスが必要です。一度に 1 つのホストポートを使用できるコンテナは 1 つだけです。コンテナインスタンスをクラスターに追加するか、タスクの必要数を減らす必要があります。

登録されたポートが多すぎる

タスク配置に最も近いコンテナインスタンスは、コンテナインスタンスごとに許可される最大予約ホストポート数である 100 を超えることはできません。ホストポートの動的マッピングを使用すると、問題を解決できる場合があります。

ポートは既に使用中です

このタスクのタスク定義は、選択されたコンテナインスタンスで既に実行されているタスクと同じポートをポートマッピングで使用します。サービスイベントメッセージは、次のメッセージの一部としてコンテナインスタンス ID を本来選択していました。

```
The closest matching container-instance is already using a port required by your
 task.
```

メモリが足りない

タスク定義で 1,000 MiB のメモリを指定しており、クラスター内の各コンテナインスタンスのメモリが 1,024 MiB の場合、コンテナインスタンスごとにこのタスクのコピーを 1 つのみ実行できます。タスク定義でメモリを減らしてコンテナインスタンスごとに複数のタスクを起動できるようにするか、クラスターで起動するコンテナインスタンスを増やすことができます。

> **Note**
>
> 特定のインスタンスタイプでタスクにできるだけ多くのメモリを提供してリソースの使用率を最大限に高めるには、「Amazon ECS Linux コンテナインスタンスのメモリを予約する」を参照してください。

十分な数の ENI アタッチメントポイントを利用できない

awsvpc ネットワークモードを使用するタスクには、それぞれ独自の Elastic Network Interface (ENI) が提供されます。この ENI は、ENI をホストするコンテナインスタンスに添付されています。Amazon EC2 インスタンスにアタッチできる ENI の数には制限があり、クラスターには利用可能な ENI 容量があるコンテナインスタンスはありません。

個別のコンテナインスタンスの ENI 制限は、以下の条件によって異なります。

- awsvpcTrunking アカウント設定をオプトインしていない場合、各コンテナインスタンスの ENI 制限は、インスタンスタイプによって異なります。詳細については、Amazon EC2 ユーザーガイドの「各インスタンスタイプのネットワークインターフェイスあたりの IP アドレス」を参照してください。
- awsvpcTrunking アカウント設定をオプトインしているが、オプトイン後にサポートされているインスタンスタイプを使用して新しいコンテナインスタンスを起動していない場合、各コンテナインスタンスの ENI 制限はデフォルト値のままです。詳細については、Amazon EC2 ユーザーガイドの「各インスタンスタイプのネットワークインターフェイスあたりの IP アドレス」を参照してください。
- awsvpcTrunking アカウント設定をオプトインしていて、かつオプトイン後にサポートされているインスタンスタイプを使用して新しいコンテナインスタンスを起動している場合、追加の ENI オプトインを利用できます。詳細については、「Amazon ECS コンテナネットワークインターフェイスの増加でサポートされるインスタンス」を参照してください。

awsvpcTrunking アカウント設定のオプトインの詳細については、「Amazon ECS Linux コンテナインスタンスのネットワークインターフェイスを増やす」を参照してください。

クラスターにコンテナインスタンスを追加することで、利用できるネットワークアダプタの数を増やすことができます。

コンテナインスタンスに必須の属性がない

一部のタスク定義パラメータでは、特定バージョンの Docker リモート API をコンテナインスタンスにインストールする必要があります。ロギングドライバーオプションなどのその他のパラ

メータでは、コンテナインスタンスに ECS_AVAILABLE_LOGGING_DRIVERS エージェント設定変数を使用して、それらのロギングドライバーを登録する必要があります。タスク定義に特定のコンテナインスタンス属性を必要とするパラメータが含まれており、この要件を満たすことができるコンテナインスタンスがない場合、そのタスクは配置できません。

このエラーの一般的な原因は、サービスが awsvpc ネットワークモードを使うタスクを使用中で、EC2 と指定のクラスターに登録されたコンテナインスタンスが、サービス作成時に awsvpcConfiguration で指定された同じサブネット内に存在していないことです。

トラブルシューティングには、AWSSupport-TroubleshootECSContainerInstance ランブックを使用できます。ランブックでは、インスタンスのユーザーデータに正しいクラスター情報が含まれているかどうか、インスタンスプロファイルに必要なアクセス権限が含まれているかどうか、およびネットワーク設定の問題が確認されます。詳細については、「AWS Systems Manager オートメーションランブックリファレンスユーザーガイド」の「AWSSupport-TroubleshootECSContainerInstance」を参照してください。

特定のタスク定義パラメータとエージェント設定変数に必要な属性の詳細については、「Fargate での Amazon ECS タスク定義パラメータ」と「Amazon ECS コンテナエージェントの設定」を参照してください。

サービス [*service-name(####-#)*] がタスクを配置できませんでした。理由: 現在、容量は使用できません。後でもう一度試すか、別のアベイラビリティーゾーンで試してください。

現在、サービスを実行できる容量がありません。

次のいずれかを行うことができます。

- Fargate 容量または EC2 コンテナインスタンスが使用可能になるまでお待ちください。
- サービスを再起動し、追加のサブネットを指定します。

サービス (*service-name*) のデプロイに失敗しました:タスクを開始できませんでした。

サービスのタスクが開始できませんでした。

停止タスクをデバッグする方法については、「Amazon ECS の停止したタスクのエラーメッセージ」を参照してください。

サービス (*service-name*) が、Amazon ECS エージェントの開始を待ってタイムアウトになりました。/var/log/ecs/ecs-agent.log でログを確認してください。"

タスク配置に最も一致するコンテナインスタンス上の Amazon ECS コンテナエージェントが切断されています。コンテナインスタンスに SSH で接続できる場合は、エージェントログを調べることができます。詳細については、「Amazon ECS コンテナエージェントのログ設定パラメータ」を参照してください。エージェントがインスタンスで実行されていることも確認する必要があります。Amazon ECS に最適化された AMI を使用している場合、次のコマンドでエージェントを停止して再開始する試みができます。

- Amazon ECS に最適化された Amazon Linux 2 AMI の場合

```
sudo systemctl restart ecs
```

- Amazon ECS に最適化された Amazon Linux AMI の場合:

```
sudo stop ecs && sudo start ecs
```

TARGET GROUP IS NOT FOUND が原因で、ターゲットグループ (*targetGroup-ARN*) 内のサービス (*service-name*) のタスクセット (*TaskSet-ID*) (task *task-id*) に異常があります。

ターゲットグループが見つからなかったため、サービスのタスクセットはヘルスチェックに合格できません。メッセージには、どの特定のタスクがヘルスチェックに失敗しているかを識別するのに役立つタスク ID が含まれています。サービスを削除して再作成してください。対応する Amazon ECS サービスが既に削除されていない限り、Elastic Load Balancing のターゲットグループを削除しないでください。

TARGET IS NOT FOUND が原因で、ターゲットグループ (*targetGroup-ARN*) 内のサービス (*service-name*) のタスクセット (*TaskSet-ID*) (task *task-id*) に異常があります。

ターゲットが見つからなかったため、サービスのタスクセットはヘルスチェックに合格できません。メッセージには、どの特定のタスクがヘルスチェックに失敗しているかを識別するのに役立つタスク ID が含まれています。

IAM アクセス許可ポリシーが正しく設定されていないか変更されているため、ECS はサービスを維持できなくなりました

IAM アクセス許可ポリシーの設定ミスまたは変更により、サービスはタスクを維持できません。ECS サービスまたはタスクに関連付けられた IAM ロールに必要なアクセス許可がない可能性があります。

この問題を解決するには、IAM ロールに必要なアクセス許可を追加します。IAM アクセス許可ポリシーの管理について詳しくは、「IAM ユーザーガイド」の「IAM ID のアクセス許可の追加および削除」を参照してください。

IAM 信頼関係が正しく設定されていないか変更されているため、ECS はサービスを維持できなくなりました。

設定ミスや IAM 信頼関係の変更により、サービスはタスクを維持できません。ECS サービスまたはタスクに関連付けられた IAM ロールの信頼ポリシーが正しくない可能性があります。

この問題を解決するには、タスク定義で使用されるロールの信頼ポリシーを設定します。信頼ポリシーの作成について詳しくは、「IAM ユーザーガイド」の「Creating a role for a custom use case」を参照してください。

サービス (*service-name*) は、デプロイ *deployment-id* のタスク *number* 件を起動できませんでした。

このイベントメッセージは、デプロイワークフローが一部のタスクを正常に開始したが、容量不足エラーが原因でリクエストされたすべてのタスクを起動できなかった場合、サービススケジューラにより送信されます。これは通常、サーキットブレーカーが有効になっており、デプロイが失敗またはロールバックされる理由が可視化されている場合に発生します。

メッセージには、具体的な障害の原因 (CPU 不足、メモリ不足、その他のリソース制約など) が含まれています。これを確認することで、デプロイの問題を解決するために対処する必要があるリソースを理解できます。

詳細については、「サービス (*service-name*) で、すべての要件を満たしたコンテナインスタンスがないため、タスクを配置できませんでした。」を参照してください。

サービス (*service-name*) は、タスクのプロビジョニング容量の制限を超えたため、クラスターにタスクを配置できませんでした。

このイベントメッセージは、クラスターが同時に PROVISIONING 状態になることができる制限値 (タスク 500 件) に達した場合、サービススケジューラにより送信されます。これはクラスターレベルの制限です。サービス固有の問題ではありません。

通常このイベントは、タスクに事前プロビジョニング済みの容量制限がある状態で多数のタスクを実行するサービスを開始した場合や、複数のサービスを同時に開始したためにタスクが大量に作成される場合に発生します。

この問題を解決するには。

- 既存のタスクのプロビジョニングが完了し、RUNNING 状態に移行するまで待機します。
- プロビジョニングの制限に達しないよう、サービスのスケーリング速度を低下させることを検討してください。
- クラスターのキャパシティプロバイダー設定を確認して、適切なリソースが使用可能になっていることを確認します。

Amazon ECS Service Quotas の詳細については、「Amazon Web Services 全般リファレンス」の「Amazon Elastic Container Service のエンドポイントとクォータ」を参照してください。

Amazon ECS サービス異常イベントメッセージ

以下は、サービス異常イベントメッセージの例です。

EC2: (service *service-name*) (task *task-id*) (instance *instance-id*) (port *port-number*) は (target-group *target-group-name*) で異常があり、その原因は (reason *failure-reason*) です。

このメッセージは、EC2 インスタンスで実行されているタスクがヘルスチェックに失敗したことを示します。詳細については次を参照してください:

- EC2 を使用する Amazon ECS タスクが、Application Load Balancer のヘルスチェックにパスするにはどうすればよいですか?

taskSet: (service ***service-name***, taskSet ***taskSet-id***) (task ***task-id***) (instance ***instance-id***) (port ***port-number***) の EC2 が (target-group ***target-group-name***) で異常であり、その原因は (reason ***failure-reason***) です。

このメッセージは、EC2 インスタンスで実行されているタスクセット内のタスクがヘルスチェックに失敗していることを示します。詳細については次を参照してください:

- EC2 を使用する Amazon ECS タスクが、Application Load Balancer のヘルスチェックにパスするにはどうすればよいですか?

Fargate: (service ***service-name***) (task ***task-id***) (port ***port-number***) が (target-group ***target-group-name***) で異常であり、その原因は (reason ***failure-reason***) です

このメッセージは、Fargate タスクがヘルスチェックに失敗したことを示します。

Fargate タスクでのヘルスチェックの失敗のトラブルシューティングの詳細については、「Fargate 上の Amazon ECS タスクがヘルスチェックにパスできない問題をトラブルシューティングする方法を教えてください。」を参照してください。

taskSet: (service ***service-name***, taskSet ***taskSet-id***) (task ***task-id***) (port ***port-number***) の Fargate が、(target-group ***target-group-name***) で異常であり、その原因は (reason ***failure-reason***) です

このメッセージは、Fargate で実行されているタスクセット内のタスクがヘルスチェックに失敗していることを示します。

Fargate タスクでのヘルスチェックの失敗のトラブルシューティングの詳細については、「Fargate 上の Amazon ECS タスクがヘルスチェックにパスできない問題をトラブルシューティングする方法を教えてください。」を参照してください。

Amazon ECS アベイラビリティーゾーンサービスの再調整サービスイベントメッセージ

以下は、表示される可能性があるサービスイベントメッセージの例です。

サービス (*service-name*) は、*Availability Zone 1* の *number-tasks* タスク、*Availability Zone 2* の *number-tasks* タスク、*Availability Zone 3* の *number-tasks* タスクで AZ が調整されていません。AZ の再調整は進行中です。

サービススケジューラは、タスク数がアベイラビリティーゾーン間で均等に分散されていない場合、service (*service-name*) is not AZ balanced サービスイベントを送信します。実行するアクションはありません。これは情報イベントです。

サービス (*service-name*) は、*Availability Zone 1* の *number-tasks* タスク、*Availability Zone 2* の *number-tasks* タスク、*Availability Zone 3* の *number-tasks* タスクで AZ が調整されています。

サービススケジューラは、アベイラビリティーゾーンのサービスの再調整が完了すると、service (*service-name*) is AZ balanced サービスイベントを送信します。実行するアクションはありません。これは情報イベントです。

service-name は、AZ の再調整のために *Availability Zone* の *number-tasks* タスクを開始しました: *task-ids*。

サービススケジューラは、サービスの再調整のためにアベイラビリティーゾーンのタスクを開始する際、*service-name*/*task-set-name* が *Availability Zone* の *number* 件のタスクを開始したことを知らせるサービスイベントを送信します。実行するアクションはありません。これは情報イベントです。

service-name は、AZ の再調整のために *Availability Zone* の実行中の *number-tasks* タスクを停止しました: *task-ids*。

サービススケジューラは、サービスの再調整のためにアベイラビリティーゾーンにあるタスクを停止する際、*service-name*/*task-set-name* が*##########*の *number* 件のタスクを停止したことを知らせるサービスイベントを送信します。実行するアクションはありません。これは情報イベントです。

サービス (*service-name*) で、すべての要件を満たしたコンテナインスタンスがないため、*Availability Zone* にタスクを配置できません。

サービススケジューラは、すべての要件を満たしたコンテナインスタンスがないため、*service-name* で *Availability Zone* にタスクを配置できないというサービスイベントを送信します。問題に対処するには、アベイラビリティーゾーンでインスタンスを起動します。

サービス (*service-name*) で、*Availability Zone* にタスクを配置できません。

サービススケジューラは、Fargate 起動タイプの使用中に利用可能なキャパシティがない場合、*service-name* で *Availability Zone* にタスクを配置できないというサービスイベントを送信します。

追加のキャパシティを取得するには、エラーメッセージでアベイラビリティーゾーンにサブネットを追加するか、サポートにお問い合わせください。

サービス (*service-name*) で、*task-set-name* が *reason* によりスケールインできなかったため、AZ を再調整できませんでした。

サービススケジューラは、タスクスケールイン保護を使用するときに、*task-set-name* が *reason* によりスケールインできなかったため、*service-name* で AZ を再調整できなかったというサービスイベントを送信します。

次のいずれかを行うことができます。

- 現在のタスクの保護が期限切れになり、タスクを終了できるようになるまでお待ちください。
- どのタスクを停止できるかを判断し、UpdateTaskProtection API で protectionEnabled オプションを false に設定し、これらのタスクに対する保護を設定解除します。
- サービスの必要なタスク数を増やして、保護されているタスクの数よりも多くします。

サービス (*service-name*) が AZ の再調整を停止しました。

サービススケジューラは、アベイラビリティーゾーンの再調整オペレーションが停止すると、*service-name* が AZ の再調整を停止したというサービスイベントを送信します。これは情報イベントです。Amazon ECS は、詳細情報を提供する追加のイベントを送信します。

Amazon ECS のサービスロードバランサーのトラブルシューティング

Amazon ECS サービスは、Elastic Load Balancing のロードバランサーにタスクを登録することができます。ロードバランサーの設定エラーは、タスクが停止された一般的な原因です。ロードバランサーを使用するサービスによって停止されたタスクが開始された場合は、以下の原因が考えられます。

Amazon ECS サービスにリンクされたロールは存在しません

Amazon ECS サービスにリンクされたロールを使用すると、Amazon ECS サービスが Elastic Load Balancing ロードバランサーにコンテナインスタンスを登録できます。サービスにリンクされたロールはアカウントに作成される必要があります。詳細については、「Amazon ECS のサービスリンクロールの使用」を参照してください。

コンテナインスタンスのセキュリティグループ

コンテナがコンテナインスタンスのポート 80 にマッピングされている場合、コンテナインスタンスのセキュリティグループでは、ロードバランサーのヘルスチェックに合格するように、ポート 80 上の受信トラフィックを許可する必要があります。

一部のアベイラビリティゾーンで、Elastic Load Balancing ロードバランサーが設定されていません

リージョン内のすべてのアベイラビリティーゾーンを使用するように、または少なくとも、コンテナインスタンスが存在するすべてのアベイラビリティーゾーンを使用するように、ロードバランサーが設定されている必要があります。サービスでロードバランサーを使用し、ロードバランサーを使用するように設定されていないアベイラビリティーゾーンにあるコンテナインスタンスでタスクが開始されると、そのタスクはヘルスチェックに合格できません。これにより、タスクは強制終了されます。

Elastic Load Balancing ロードバランサー ヘルスチェックが正しく設定されていません

ロードバランサーのヘルスチェックパラメータが過度に制限されているか、存在しないリソースを指している可能性があります。コンテナインスタンスが異常であると判断されると、そのコンテナインスタンスはロードバランサーから削除されます。以下のパラメータがサービスロードバランサーに対して正しく設定されていることを確認してください。

ping ポート

ロードバランサーのヘルスチェックの [Ping Port (ping ポート)] 値は、ロードバランサーが正常であるかどうかを判断するために確認するコンテナインスタンス上のポートです。このポートの設定が正しくないと、多くの場合、ロードバランサーからコンテナインスタンスが登録解除されます。このポートは、ヘルスチェックで使用しているサービスのタスク定義内のコンテナで `hostPort` 値を使用するように設定する必要があります。

ping パス

これはロードバランサーのヘルスチェックの一部です。これは、アプリケーションが正常な場合に、成功のステータスコード (200 など) を返すことができる、アプリケーション上のエンドポイントです。この値はよく `index.html` に設定されることがありますが、サービスがそのリクエストに応答しない場合、ヘルスチェックは失敗します。コンテナに `index.html`

ファイルがない場合は、これを / に設定して、コンテナインスタンスのベース URL をターゲットにすることができます。

応答タイムアウト

これは、コンテナがヘルスチェック ping に対する応答を返す必要のある時間です。この値が応答に必要な時間よりも短い場合、ヘルスチェックは失敗します。

ヘルスチェック間隔

これは、ヘルスチェック ping 間の時間です。ヘルスチェックの間隔が短くなるほど、コンテナインスタンスが [Unhealthy Threshold (非正常のしきい値)] に達するのが速くなります。

非正常のしきい値

これは、コンテナインスタンスが異常と見なされるまでに、ヘルスチェックが失敗できる回数です。異常と見なされるまでのしきい値が 2、ヘルスチェックの間隔が 30 秒の場合、コンテナインスタンスが異常と見なされるまでに、タスクはヘルスチェック ping に 60 秒間応答します。異常と見なされるまでのしきい値を増やすか、ヘルスチェックの間隔を長くすると、タスクが ping に応答する時間が長くなります。

servicename サービスを更新できません: タスク定義でロードバランサーのコンテナ名またはポートが変更されました

サービスでロードバランサーを使用している場合は、AWS CLI または SDK を使用してロードバランサーの設定を変更できます。設定の変更方法については、「Amazon Elastic Containers Service API リファレンス」の UpdateService を参照してください。サービスのタスク定義を更新する場合は、ロードバランサー設定で指定されたコンテナ名とコンテナポートはタスク定義のままにしておく必要があります。

同時に実行できるタスク数の上限に達しました。

新しいアカウントの場合、クォータがサービスクォータよりも低くなることがあります。Service Quotas コンソールでは、アカウントのサービスクォータを表示できます。クォータの引き上げをリクエストするには、Service Quotas ユーザーガイド の「クォータ引き上げリクエスト」を参照してください。

Amazon ECS のサービス自動スケーリングのトラブルシューティング

Amazon ECS デプロイの進行中、Application Auto Scaling はスケールインプロセスをオフにします。このプロセスは、デプロイの完了後に再開されます。ただし、スケールアウトプロセスは、中断

しない限り、デプロイ中に引き続き発生します。詳細については、「Application Auto Scaling のスケーリングの中断と再開」を参照してください。

コンソールの Amazon ECS イベントキャプチャ

Amazon ECS コンソールには、サービスアクションやタスク状態の変更など、Amazon ECS が生成したイベントを EventBridge 経由で Amazon CloudWatch Logs に保存するイベントキャプチャ機能が用意されています。この機能には、モニタリングとトラブルシューティングのためのフィルタリング機能を備えたクエリインターフェイスが含まれています。

イベントは、サービスのデプロイ、サービス、タスク、インスタンスの動作に関する詳細情報を提供します。この情報を使用して、タスクまたはサービスのデプロイの失敗をトラブルシューティングできます。

イベントキャプチャを有効にすると、選択した保持期間中に Amazon ECS が生成するすべてのイベントにアクセスできるようになります。これにより、フィルタリングされていない 100 件を超える過去のイベントにアクセスでき、また、停止したタスクは 1 時間以上表示可能になります。

仕組み

イベントキャプチャは EventBridge を使用して、事前定義された Amazon CloudWatch Logs ロググループにイベントを保存します。Amazon ECS コンソールは、事前に構築されたクエリとフィルタリングオプションを提供し、イベントを関連付けて直感的な形式でタスクライフサイクルを提供します。

次のタイプのイベントをクエリして取得できます。

- サービスアクションイベント – プロビジョニングまたはリソース割り当ての問題を特定するのに役立ちます
- タスクライフサイクルイベント – タスクまたはコンテナが起動または実行停止に失敗する理由を特定するのに役立ちます

Amazon ECS コンソールを使用すると、ワンクリックでイベントキャプチャを設定し、一般的に使用されるクエリやフィルタリングを利用できます。クエリ言語の習得は不要で、複数のコンソール間を移動する必要もありません。

イベントタイプ

イベントキャプチャは、Amazon ECS で生成されたすべてのイベントを次のカテゴリに保存します。

タスク状態変更イベント

コンテナの停止およびその他の終了イベント。トラブルシューティングやタスクのライフサイクルタイムラインのモニタリングに使用できます。

サービスアクション

定常状態に達する、タスク配置の失敗、リソースの制約などのイベント。

サービスデプロイの状態の変更

サービスデプロイの状態をモニタリングするために、サーキットブレーカーとロールバック設定によってトリガーされるイベント (進行中のデプロイ、完了したデプロイ、失敗したデプロイなど)。

コンテナインスタンス状態の変更

EC2 および Amazon ECS マネージドインスタンスのワークロードの場合、イベントには接続と切断の状態が表示されます。

ロググループ設定

イベントキャプチャを有効にすると、Amazon ECS は次のリソースを自動的に作成します。

- `/aws/events/ecs/containerinsights/${clusterName}/performance` という名前の Amazon CloudWatch Logs ロググループ
- `aws.ecs` ソースからすべてのイベントを取り込みロググループに転送する EventBridge ルール

ロググループの保持期間は 1 日から 10 年の間で指定できます。デフォルトの保持期間は 7 日間です。

考慮事項

イベントキャプチャを使用する場合は、以下の点を考慮してください。

- イベントキャプチャは、簡素化の目的ですべてのイベントを保存します。Amazon ECS コンソールで、特定のイベントのみをキャプチャするルールを設定することはできません。

- Amazon ECS コンソールには、事前定義済みのクエリ条件が用意されています。高度なクエリの場合は、Amazon CloudWatch Logs Insights を使用してロググループに直接クエリを実行します。
- ライブテール機能は Amazon ECS コンソールでは使用できません。ライブテールには Amazon CloudWatch Logs を直接使用します。
- イベントキャプチャを無効にすると、EventBridge ルールは削除されます。
- イベントキャプチャでは、EventBridge のデータ取り込み、Amazon CloudWatch Logs ストレージ、クエリの実行に追加のコストが発生します。

  EventBridge の料金の詳細については、「EventBridge の料金」を参照してください。

  CloudWatch の料金の詳細については、「CloudWatch の料金」をご覧ください。

イベントベースのトラブルシューティング

Amazon ECS で生成されたイベントを活用して、一般的なトラブルシューティングの疑問に対応します。

タスク障害分析

STOPPED タスクの状態変更イベント、停止コード、コンテナ終了コードを確認して、タスクが起動に失敗した理由や、実行中に失敗した理由を特定できます。

配置の失敗およびリソース制約情報に関するサービスアクションイベントを確認して、リソースの制約が原因でタスクが配置に失敗した理由を特定できます。

一般的なタスク障害シナリオ

最も一般的に発生する異常なタスク障害は、次の問題に関連しています。

- CI/CD サービスのデプロイの失敗
- 自動スケーリングの失敗
- タスクの再調整の失敗
- out–of–memory (OOM) エラーなど、異常なコンテナの終了

異常なタスク障害は、`EssentialContainerExited` または `TaskFailedToStart` の停止コードを含む STOPPED タスク状態変更イベントを生成します。これらの停止コードでフィルタリングして、コンテナの実行と停止の動作を調べることができます。

既存の Amazon ECS クラスターのイベントキャプチャを有効にする

既存の Amazon ECS クラスターのイベントキャプチャを有効にして、EventBridge を通じて Amazon ECS で生成されたイベントを Amazon CloudWatch Logs に保存できます。この機能は、タスクの障害、サービスのデプロイ、その他のクラスターアクティビティのモニタリングおよびトラブルシューティングに役立ちます。

イベントキャプチャを有効にすると、Amazon ECS は次のリソースを作成します。

- `/aws/events/ecs/containerinsights/${clusterName}/performance` という名前の Amazon CloudWatch Logs ロググループ
- `aws.ecs` ソースからすべてのイベントをキャプチャする EventBridge ルール

クラスタービューに [履歴] タブが表示され、タスクライフサイクルイベントとサービスアクションにクエリを実行できます。イベントキャプチャは直ちに開始され、指定した保持期間に従って Amazon ECS が生成したすべてのイベントを保存します。

前提条件

- 既存の Amazon ECS クラスター
- クラスター設定を変更し、Amazon CloudWatch Logs リソースを作成するための適切な IAM アクセス権限

コンソールを使用してイベントキャプチャを有効にする

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。
3. イベントキャプチャを有効にするクラスターを選択します。

   クラスター詳細のページが表示されます。
4. [設定] を選択します。
5. [ECS イベント] セクションで、[イベントキャプチャをオンにする] を選択します。

   [イベントキャプチャをオンにする] のダイアログボックスが表示されます。
6. [イベントを失効] で、Amazon CloudWatch Logs ロググループの保持期間を選択します。デフォルトは 7 日間です。
7. [オンにする] を選択します。

Amazon ECS サービスおよびタスクの状態変更イベントを表示する

Amazon ECS コンソールには、サービスアクションやタスク状態の変更など、Amazon ECS が生成したイベントを EventBridge 経由で Amazon CloudWatch Logs に保存するイベントキャプチャ機能が用意されています。この機能には、モニタリングとトラブルシューティングを拡張するフィルタリング機能を備えたクエリインターフェイスが含まれています。

イベントは、サービスのデプロイ、サービス、タスク、インスタンスの動作に関する詳細情報を提供します。この情報を使用して、タスクまたはサービスのデプロイの失敗をトラブルシューティングできます。

次のいずれかの基準を使用して、イベントをフィルタリングできます。

- デプロイ ID (これは [サービス詳細] ページでのみ使用できます)
- 開始時間
- 終了時間
- サービス名 ([クラスターの詳細] ページ、[サービス詳細] ページでのみ使用できます。デフォルトで現在のサービスになります)
- タスク ID
- タスクの最終状態
- タスク定義ファミリー
- タスク定義リビジョン

クラスターレベルでのイベントを表示する

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
2. [クラスター] を選択します。

   クラスターリストページが表示されます。
3. クラスターを選択します。

   クラスター詳細のページが表示されます。
4. [履歴] で、表示するイベントを決定します。

   a. サービスアクションイベントを表示するには、[サービスアクションイベント] を選択します。

b. タスク状態の変更イベントを表示するには、[タスク状態の変更イベント] を選択します。

c. (オプション) [クエリ基準] に、表示するイベントのフィルターを入力します。

5. [Run query] (クエリの実行) を選択します。

   イベントがリストに表示されます。

6. イベントの詳細をすべて表示するには、イベントを選択します。

サービスレベルで表示する

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
2. [クラスター] ページで、クラスターを選択します。
3. [クラスターの詳細] ページの [サービス] セクションで、サービスを選択します。

   [サービスの詳細] ページが表示されます。

4. [履歴] で、表示するイベントを決定します。

   a. サービスアクションイベントを表示するには、[サービスアクションイベント] を選択します。

   b. タスク状態の変更イベントを表示するには、[タスク状態の変更イベント] を選択します。

   c. (オプション) [クエリ基準] に、表示するイベントのフィルターを入力します。

5. [Run query] (クエリの実行) を選択します。

   イベントがリストに表示されます。

6. イベントの詳細をすべて表示するには、イベントを選択します。

Amazon ECS タスク定義の無効な CPU またはメモリエラーをトラブルシューティングする

Amazon ECS API または AWS CLI を使用してタスク定義を登録する場合、無効なcpuまたはmemory値を指定すると、以下のエラーが返されます。

```
An error occurred (ClientException) when calling the RegisterTaskDefinition operation:
 Invalid 'cpu' setting for task.
```

> **Note**
>
> Terraform の使用時に、以下のエラーが返される可能性があります。
>
> ```
> Error: ClientException: No Fargate configuration exists for given values.
> ```

この問題を解決するには、タスク定義でタスクの CPU とメモリにサポートされている値を指定する必要があります。cpu 値はタスク定義で、CPU ユニットまたは vCPU で表すことができます。タスク定義が登録されると、CPU ユニットを示す整数に変換されます。memory 値はタスク定義で、MiB または GB で表すことができます。タスク定義が登録されると、MiB を示す整数に変換されます。

requiresCompatibilities パラメータに FARGATE を指定しているタスク定義については (EC2 も指定されている場合も)、次の表のいずれかの値を使用する必要があります。これらの値は、CPU とメモリのパラメータでサポートされる値の範囲を決定します。

Fargate でホストされるタスクの場合、次の表に有効な CPU とメモリの組み合わせを示します。JSON ファイルのメモリ値は MiB 単位で指定されます。この値に 1024 を掛けると、GB 値を MiB に変換できます。例えば、1 GB = 1024 MiB です。

CPU の値	メモリの値	AWS Fargate でサポートされるオペレーティングシステム
256 (.25 vCPU)	512 MiB、1 GB、2 GB	Linux
512 (.5 vCPU)	1 GB、2 GB、3 GB、4 GB	Linux
1,024 (1 vCPU)	2 GB、3 GB、4 GB、5 GB、6 GB、7 GB、8 GB	Linux、Windows
2,048 (2 vCPU)	4 GB ~ 16 GB (1 GB のインクリメント)	Linux、Windows
4,096 (4 vCPU)	8 GB ~ 30 GB (1 GB のインクリメント)	Linux、Windows
8192 (8 vCPU)	16 GB ~ 60 GB (4 GB のインクリメント)	Linux

CPU の値	メモリの値	AWS Fargate でサポートされるオペレーティングシステム
Note このオプションには Linux プラットフォーム 1.4.0 以降が必要です。		
16384 (16 vCPU) Note このオプションには Linux プラットフォーム 1.4.0 以降が必要です。	32 GB ~ 120 GB (8 GB のインクリメント)	Linux
32768 (32 vCPU) Note このオプションには Linux プラットフォーム 1.4.0 以降が必要です。	60 GB、120 GB、244 GB	Linux

Amazon EC2 でホストされているタスクでサポートされるタスク CPU の値は、0.25 vCPU ~ 192 vCPU です。

CPU 制御メカニズムは、EC2 と Fargate で異なります。

- Amazon EC2 でホストされているタスクの場合: Amazon ECS は、CPU 期間と CPU クォータを使用して、タスクサイズの CPU ハード制限を制御します。タスク定義で vCPU を指定すると、Amazon ECS は `cgroup` に適用される CPU 期間と CPU クォータの設定に値を変換します。

- Fargate でホストされているタスクの場合: Amazon ECS は CPU 共有を使用して CPU 割り当てを制御します。CPU クォータと期間値は、Fargate タスクの CPU 制限には使用されません。

Amazon EC2 タスクの場合、CPU クォータは特定の CPU 期間中に cgroup に付与される CPU 時間を制御します。どちらの設定も、マイクロ秒単位で表されます。CPU クォータが CPU 期間と等しい場合、cgroup は 1 つの vCPU で最大 100% (または複数の vCPU の場合は合計で 100% になる割合) まで実行できます。CPU クォータは最大 1,000,000 マイクロ秒で、CPU 期間は最小 1 ミリ秒です。これらの値を使用して CPU 数の制限を設定できます。CPU クォータを変更せずに CPU 期間を変更する場合、有効になる制限はタスク定義で指定した制限とは異なります。

100 ミリ秒の期間では、0.125 ~ 10 の vCPU を使用できます。

> **Note**
>
> タスクレベル CPU およびメモリのパラメータは Windows コンテナでは無視されます。

Amazon ECS コンテナエージェントログの表示

Amazon ECS によってログはコンテナインスタンスの /var/log/ecs フォルダに保存されます。Amazon ECS コンテナエージェントから得られるログと、コンテナインスタンスのエージェントの状態 (スタート/停止) を制御する ecs-init サービスから得られるログがあります。これらのログファイルは、コンテナインスタンスに SSH で接続することにより表示できます。

> **Note**
>
> コンテナインスタンスのログをすべて収集する方法がわからない場合は、Amazon ECS ログコレクターを使用できます。詳細については、「Amazon ECS ログコレクターを使用したコンテナログの収集」を参照してください。

Linux オペレーティングシステム

ecs-init プロセスはログを /var/log/ecs/ecs-init.log に保存します。

ecs-init.log ファイルには、コンテナエージェントのライフサイクル管理、設定、ブートストラップに関する情報が含まれています。

ログファイルを表示するには、次のコマンドが使用できます。

```
cat /var/log/ecs/ecs-init.log
```

出力:

```
2018-02-16T18:13:54Z [INFO] pre-start
2018-02-16T18:13:56Z [INFO] start
2018-02-16T18:13:56Z [INFO] No existing agent container to remove.
2018-02-16T18:13:56Z [INFO] Starting Amazon Elastic Container Service Agent
```

Windows オペレーティングシステム

Windows 用の Amazon ECS ログコレクターを使用できます。詳細については、Github の「Amazon ECS Logs Collector For Windows」を参照してください。

1. インスタンスに接続します。
2. PowerShell を開き、管理者権限で次のコマンドを実行します。このコマンドは、スクリプトをダウンロードし、ログを収集します。

   ```
   Invoke-WebRequest -OutFile ecs-logs-collector.ps1 https://
   raw.githubusercontent.com/awslabs/aws-ecs-logs-collector-for-windows/master/ecs-
   logs-collector.ps1
   .\ecs-logs-collector.ps1
   ```

Amazon ECS エージェントおよび Docker デーモンのデバッグログを有効にできます。このオプションを使用すると、スクリプトはデバッグモードを有効にする前にログを収集できます。スクリプトは Docker デーモンと Amazon ECS エージェントを再起動し、インスタンスで実行されているすべてのコンテナを終了します。次のコマンドを実行する前に、コンテナインスタンスをドレインし、重要なタスクを他のコンテナインスタンスに移動します。

次のコマンドを実行して、ログ記録をオンにします。

```
.\ecs-logs-collector.ps1 -RunMode debug
```

Amazon ECS ログコレクターを使用したコンテナログの収集

> **Note**
>
> Amazon ECS マネージドインスタンスで Amazon ECS ログコレクターを使用することはできません。

コンテナインスタンスのさまざまなログをすべて収集する方法がわからない場合は、Amazon ECS ログコレクターを使用できます。Linux 用と Windows 用のいずれも GitHub で入手できます。スクリプトは一般的なオペレーティングシステムログおよび Docker と Amazon ECS コンテナエージェントログを収集します。これらは、AWS サポート ケースのトラブルシューティングに役立ちます。次に、収集された情報が、診断目的で簡単に共有することができる 1 つのファイルに圧縮およびアーカイブされます。また、Docker デーモン、および Amazon Linux バリアントで Amazon ECS コンテナエージェント (Amazon ECS 最適化 AMI など) に対してデバッグモードを有効にすることもできます。

> **Note**
>
> Amazon Linux Amazon ECS 最適化 AMI のバージョン 20250909 以降では、Amazon ECS ログコレクターが `/opt/amazon/ecs/ecs-logs-collector.sh` にあらかじめインストールされているため、GitHub からダウンロードしなくても使用をすぐに開始できます。詳細については、ECS 最適化 AMI ドキュメントの「ECS Logs Collector」を参照してください。

現在、Amazon ECS ログコレクターでは以下のオペレーティングシステムがサポートしています:

- Amazon Linux
- Red Hat Enterprise Linux
- Ubuntu
- Windows サーバー

Linux 用 Amazon ECS ログコレクターを実行するには (ECS 最適化 AMI)

1. コンテナインスタンスに接続します。
2. スクリプトを実行してログを収集し、アーカイブを作成します。

> **Note**
>
> Docker デーモンと Amazon ECS コンテナエージェントに対してデバッグモードを有効にするには、次のコマンドに --mode=enable-debug オプションを追加します。これにより、Docker デーモンが再起動され、インスタンスで実行されているすべてのコンテナが強制終了されます。デバッグモードを有効にする前に、コンテナインスタンスをドレインし、重要なタスクを他のコンテナインスタンスに移動することを検討してください。詳細については、「Amazon ECS コンテナインスタンスをドレインする」を参照してください。

```
[ec2-user ~]$ sudo /opt/amazon/ecs/ecs-logs-collector.sh
```

スクリプトを実行した後、スクリプトによって作成された collect フォルダに収集されたログを調べることができます。collect.tgz ファイルはすべてのログの圧縮アーカイブであり、AWS サポートと共有することで診断に役立ちます。

LinuxにAmazon ECS ログコレクターをダウンロードして実行するには

1. コンテナインスタンスに接続します。
2. Amazon ECS ログ コレクタースクリプトをダウンロードします。

```
curl -O https://raw.githubusercontent.com/awslabs/ecs-logs-collector/master/ecs-logs-collector.sh
```

3. スクリプトを実行してログを収集し、アーカイブを作成します。

```
$ sudo bash ./ecs-logs-collector.sh
```

WindowsでAmazon ECS ログコレクターをダウンロードして実行するには

1. コンテナインスタンスに接続します。詳細については、「Amazon EC2 ユーザーガイド」の「RDP を使用した Windows インスタンスへの接続」を参照してください。
2. PowerShell を使用して、Amazon ECSログコレクターのスクリプトをダウンロードします。

```
Invoke-WebRequest -OutFile ecs-logs-collector.ps1 https://
raw.githubusercontent.com/awslabs/aws-ecs-logs-collector-for-windows/master/ecs-
logs-collector.ps1
```

3. スクリプトを実行してログを収集し、アーカイブを作成します。

> **Note**
>
> Docker デーモンと Amazon ECS コンテナエージェントに対してデバッグモードを有効にするには、次のコマンドに -RunMode debug オプションを追加します。これにより、Docker デーモンが再起動され、インスタンスで実行されているすべてのコンテナが強制終了されます。デバッグモードを有効にする前に、コンテナインスタンスをドレインし、重要なタスクを他のコンテナインスタンスに移動することを検討してください。詳細については、「Amazon ECS コンテナインスタンスをドレインする」を参照してください。

```
.\ecs-logs-collector.ps1
```

スクリプトを実行した後、スクリプトによって作成された collect フォルダに収集されたログを調べることができます。collect.tgz ファイルはすべてのログの圧縮アーカイブであり、診断に役立つように AWS サポートと共有できます。

エージェントのイントロスペクションを使用して Amazon ECS 診断の詳細を取得する

Amazon ECS エージェントのイントロスペクション API は、Amazon ECS エージェントとコンテナインスタンスの全体的な状態に関する情報を提供します。

エージェントのイントロスペクション API を使用して、タスク内のコンテナの Docker ID を取得できます。コンテナインスタンスに SSH で接続することにより、エージェントイントロスペクション API を使用できます。

> **Important**
>
> イントロスペクション API に到達するには、コンテナインスタンスにAmazon ECS にアクセスできる IAM ロールが必要です。詳細については、「Amazon ECS コンテナインスタンスの IAM ロール」を参照してください。

次の例では、2 つのタスクを示しています。1 つは現在実行中のタスク、もう 1 つは停止されたタスクです。

> **Note**
>
> 次のコマンドは、読みやすくするために python -mjson.tool によりパイプされています。

```
curl http://localhost:51678/v1/tasks | python -mjson.tool
```

出力:

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1095  100  1095    0     0   117k      0 --:--:-- --:--:-- --:--:--  133k
{
    "Tasks": [
        {
            "Arn": "arn:aws:ecs:us-west-2:aws_account_id:task/090eff9b-1ce3-4db6-848a-
a8d14064fd24",
            "Containers": [
                {
                    "DockerId":
 "189a8ff4b5f04affe40e5160a5ffadca395136eb5faf4950c57963c06f82c76d",
                    "DockerName": "ecs-console-sample-app-static-6-simple-
app-86caf9bcabe3e9c61600",
                    "Name": "simple-app"
                },
                {
                    "DockerId":
 "f7f1f8a7a245c5da83aa92729bd28c6bcb004d1f6a35409e4207e1d34030e966",
                    "DockerName": "ecs-console-sample-app-static-6-busybox-
ce83ce978a87a890ab01",
```

```
                    "Name": "busybox"
                }
            ],
            "Family": "console-sample-app-static",
            "KnownStatus": "STOPPED",
            "Version": "6"
        },
        {
            "Arn": "arn:aws:ecs:us-west-2:aws_account_id:task/1810e302-eaea-4da9-
a638-097bea534740",
            "Containers": [
                {
                    "DockerId":
 "dc7240fe892ab233dbbcee5044d95e1456c120dba9a6b56ec513da45c38e3aeb",
                    "DockerName": "ecs-console-sample-app-static-6-simple-app-
f0e5859699a7aecfb101",
                    "Name": "simple-app"
                },
                {
                    "DockerId":
 "096d685fb85a1ff3e021c8254672ab8497e3c13986b9cf005cbae9460b7b901e",
                    "DockerName": "ecs-console-sample-app-static-6-
busybox-92e4b8d0ecd0cce69a01",
                    "Name": "busybox"
                }
            ],
            "DesiredStatus": "RUNNING",
            "Family": "console-sample-app-static",
            "KnownStatus": "RUNNING",
            "Version": "6"
        }
    ]
}
```

上記の例では、停止されたタスク (*090eff9b-1ce3-4db6-848a-a8d14064fd24*) には 2 つのコンテナがあります。docker inspect ***container-ID*** を使用して、各コンテナの詳細情報を表示できます。詳細については、「Amazon ECS コンテナの詳細分析」を参照してください。

Amazon ECS の Docker 診断

Docker には、コンテナやタスクに関する問題のトラブルシューティングに役立つ診断ツールがいくつか用意されています。使用できるすべての Docker コマンドラインユーティリティの詳細について

は、Docker ドキュメントの Docker CLI リファレンスを参照してください。コンテナインスタンスに SSH で接続することにより、Docker コマンドラインユーティリティにアクセスできます。

Docker コンテナからレポートされる終了コードからも診断情報を得られます (例えば、終了コード 137 は、コンテナが SIGKILL 信号を受信したことを意味します)。詳細については、Docker ドキュメントの「終了ステータス」を参照してください。

Amazon ECS の Docker コンテナを一覧表示する

コンテナインスタンスの docker ps コマンドを使用して、実行中のコンテナを一覧表示できます。次の例では、Amazon ECS コンテナエージェントのみが実行されています。詳細については、Docker ドキュメントの「docker ps」を参照してください。

```
docker ps
```

出力:

```
CONTAINER ID        IMAGE                            COMMAND             CREATED
      STATUS              PORTS                          NAMES
cee0d6986de0        amazon/amazon-ecs-agent:latest   "/agent"            22 hours ago
      Up 22 hours         127.0.0.1:51678->51678/tcp   ecs-agent
```

docker ps -a コマンドを使用して、すべてのコンテナ (停止されたコンテナまたは強制終了されたコンテナも含む) を表示できます。この情報は、予期せず停止されたコンテナを一覧表示するために役立ちます。以下の例では、コンテナ f7f1f8a7a245 は 9 秒前に終了したため、-a フラグのない docker ps 出力には表示されません。

```
docker ps -a
```

出力:

```
CONTAINER ID        IMAGE                                              COMMAND
 CREATED             STATUS                          PORTS                          NAMES
db4d48e411b1        amazon/ecs-emptyvolume-base:autogenerated   "not-applicable"
  19 seconds ago                                                                     ecs-
console-sample-app-static-6-internalecs-emptyvolume-source-c09288a6b0cba8a53700
f7f1f8a7a245        busybox:buildroot-2014.02                   "\"sh -c '/bin/sh -c
  22 hours ago        Exited (137) 9 seconds ago                                     ecs-
console-sample-app-static-6-busybox-ce83ce978a87a890ab01
```

```
189a8ff4b5f0        httpd:2                                              "httpd-foreground"
  22 hours ago        Exited (137) 40 seconds ago                                      ecs-
console-sample-app-static-6-simple-app-86caf9bcabe3e9c61600
0c7dca9321e3        amazon/ecs-emptyvolume-base:autogenerated   "not-applicable"
  22 hours ago                                                                       ecs-
console-sample-app-static-6-internalecs-emptyvolume-source-90fefaa68498a8a80700
cee0d6986de0        amazon/amazon-ecs-agent:latest              "/agent"
  22 hours ago        Up 22 hours                    127.0.0.1:51678->51678/tcp   ecs-
agent
```

Amazon ECS で Docker ログを表示する

STDOUT コマンドを使用して、コンテナの STDERR および docker logs ストリームを表示できます。この例では、ログは *dc7240fe892a* コンテナのものが表示され、見やすくするために head コマンドによりパイプされています。詳細については、Docker ドキュメントの「Docker ログ」を参照してください。

> **Note**
>
> デフォルトの `json` ログドライバーを使用している場合、Docker ログはコンテナインスタンスのみで利用できます。`awslogs` ログドライバーを使用するようにタスクを設定している場合には、コンテナログは CloudWatch Logs で使用できます。詳細については、「Amazon ECS ログを CloudWatch に送信する」を参照してください。

```
docker logs dc7240fe892a | head
```

出力:

```
AH00558: httpd: Could not reliably determine the server's fully qualified domain name,
 using 172.17.0.11. Set the 'ServerName' directive globally to suppress this message
AH00558: httpd: Could not reliably determine the server's fully qualified domain name,
 using 172.17.0.11. Set the 'ServerName' directive globally to suppress this message
[Thu Apr 23 19:48:36.956682 2015] [mpm_event:notice] [pid 1:tid 140327115417472]
 AH00489: Apache/2.4.12 (Unix) configured -- resuming normal operations
[Thu Apr 23 19:48:36.956827 2015] [core:notice] [pid 1:tid 140327115417472] AH00094:
 Command line: 'httpd -D FOREGROUND'
10.0.1.86 - - [23/Apr/2015:19:48:59 +0000] "GET / HTTP/1.1" 200 348
10.0.0.154 - - [23/Apr/2015:19:48:59 +0000] "GET / HTTP/1.1" 200 348
10.0.1.86 - - [23/Apr/2015:19:49:28 +0000] "GET / HTTP/1.1" 200 348
```

```
10.0.0.154 - - [23/Apr/2015:19:49:29 +0000] "GET / HTTP/1.1" 200 348
10.0.1.86 - - [23/Apr/2015:19:49:50 +0000] "-" 408 -
10.0.0.154 - - [23/Apr/2015:19:49:50 +0000] "-" 408 -
10.0.1.86 - - [23/Apr/2015:19:49:58 +0000] "GET / HTTP/1.1" 200 348
10.0.0.154 - - [23/Apr/2015:19:49:59 +0000] "GET / HTTP/1.1" 200 348
10.0.1.86 - - [23/Apr/2015:19:50:28 +0000] "GET / HTTP/1.1" 200 348
10.0.0.154 - - [23/Apr/2015:19:50:29 +0000] "GET / HTTP/1.1" 200 348
time="2015-04-23T20:11:20Z" level="fatal" msg="write /dev/stdout: broken pipe"
```

Amazon ECS で Docker コンテナを検査する

コンテナの Docker ID がある場合は、docker inspect コマンドを使用してコンテナを検査できます。コンテナを検査すると、コンテナが起動された環境の最も詳細なビューを得られます。詳細については、Docker ドキュメントの「Docker の検査」を参照してください。

```
docker inspect dc7240fe892a
```

出力:

```
[{
    "AppArmorProfile": "",
    "Args": [],
    "Config": {
        "AttachStderr": false,
        "AttachStdin": false,
        "AttachStdout": false,
        "Cmd": [
            "httpd-foreground"
        ],
        "CpuShares": 10,
        "Cpuset": "",
        "Domainname": "",
        "Entrypoint": null,
        "Env": [
            "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/
local/apache2/bin",
            "HTTPD_PREFIX=/usr/local/apache2",
            "HTTPD_VERSION=2.4.12",
            "HTTPD_BZ2_URL=https://www.apache.org/dist/httpd/httpd-2.4.12.tar.bz2"
        ],
        "ExposedPorts": {
            "80/tcp": {}
```

```
        },
        "Hostname": "dc7240fe892a",
...
```

Amazon ECS の Docker デーモンからの詳細な出力の設定

Docker コンテナまたはイメージに問題がある場合は、Docker デーモンに対してデバッグモードをオフにすることができます。デバッグを使用すると、デーモンからより詳細な出力が得られます。これを使用して、Amazon ECR などのコンテナレジストリから送信されるエラーメッセージを取得できます。

> **Important**
>
> この手順は、Amazon ECS 最適化 Amazon Linux AMI 用に書かれています。他のオペレーティングシステムについては、Docker ドキュメントの「デバッグの有効化」と「systemd による Docker の制御と設定」を参照してください。

Amazon ECS に最適化された Amazon Linux AMI で Docker デーモンのデバッグモードを使用するには

1. コンテナインスタンスに接続します。
2. Docker options ファイルを vi などのテキストエディタで開きます。Amazon ECS 最適化 Amazon Linux AMI の場合、Docker options ファイルは`/etc/sysconfig/docker`にあります。
3. Docker options ステートメントを見つけ、引用符の中の文字列に `-D` オプションを追加します。

   > **Note**
   >
   > Docker options ステートメントが `#` で始まっている場合、その文字を削除してそのステートメントをコメント解除し、オプションを有効にします。

   Amazon ECS 最適化 Amazon Linux AMI では、Docker options ステートメントを `OPTIONS` と呼びます。例:

   ```
   # Additional startup options for the Docker daemon, for example:
   ```

```
# OPTIONS="--ip-forward=true --iptables=true"
# By default we limit the number of open files per container
OPTIONS="-D --default-ulimit nofile=1024:4096"
```

4. ファイルを保存し、テキストエディタを終了します。
5. Docker デーモンを再び開始します。

   ```
   sudo service docker restart
   ```

   出力は次のとおりです。

   ```
   Stopping docker:                                       [  OK  ]
   Starting docker: .                                  [  OK  ]
   ```

6. Amazon ECS エージェントを再び開始します。

   ```
   sudo service ecs restart
   ```

これで、Docker ログにはより詳細な出力が表示されます。

```
time="2015-12-30T21:48:21.907640838Z" level=debug msg="Unexpected response from
 server: \"{\\\"errors\\\":[{\\\"code\\\":\\\"DENIED\\\",\\\"message\\\":\\\"User:
 arn:aws:sts::1111:assumed-role/ecrReadOnly/i-abcdefg is not authorized to perform:
 ecr:InitiateLayerUpload on resource: arn:aws:ecr:us-east-1:1111:repository/nginx_test
\\\"}]}\\n\" http.Header{\"Connection\":[]string{\"keep-alive\"}, \"Content-Type\":
[]string{\"application/json; charset=utf-8\"}, \"Date\":[]string{\"Wed, 30 Dec 2015
 21:48:21 GMT\"}, \"Docker-Distribution-Api-Version\":[]string{\"registry/2.0\"},
 \"Content-Length\":[]string{\"235\"}}"
```

Amazon ECS の Docker **API error (500): devmapper** のトラブルシューティング

以下の Docker エラーは、コンテナインスタンスのシンプールストレージがいっぱいで、Docker デーモンが新しいコンテナを作成できないことを示しています。

```
CannotCreateContainerError: API error (500): devmapper: Thin Pool has 4350 free data
 blocks which is less than minimum required 4454 free data blocks. Create more free
 space in thin pool or use dm.min_free_space option to change behavior
```

デフォルトでは、バージョン 2015.09.d以降の Amazon ECS 最適化 Amazon Linux AMIs は、/dev/xvdaに添付されファイルシステムのルートとしてマウントされたオペレーティングシステム用の8 GiB ボリュームで起動します。また、Docker がイメージとメタデータの保存に使用する /dev/xvdczに添付された22 GiB のボリュームが追加されています。このストレージ領域がいっぱいになると、Docker デーモンは新しいコンテナを作成できません。

コンテナインスタンスにストレージを追加する最も簡単な方法は、既存のインスタンスを終了し、サイズを増加したデータストレージボリュームで新しいインスタンスを起動することです。ただし、この方法を実行できない場合は、「Amazon ECS に最適化された Linux AMI」の手順に従って、Docker が使用するボリュームグループにストレージを追加し、その論理ボリュームを拡張できます。

コンテナインスタンスストレージがあまりにも速くいっぱいになる場合は、以下のいくつかの対処方法があります。

- thin poll情報を表示するには、コンテナインスタンスで次のコマンドを実行します。

  ```
  docker info
  ```

- (Amazon ECS コンテナエージェント 1.8.0 以降) 停止または終了したコンテナがコンテナインスタンスに残る時間を短縮できます。ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION エージェント設定変数で、タスクが停止されてから Docker コンテナが削除されるまでの時間を設定します (デフォルトでは、この値は 3 時間です)。これにより Docker コンテナのデータは削除されます。この値が低すぎると、ログを削除するまでは停止したコンテナを検査したり、ログを表示したりすることができない場合があります。詳細については、「Amazon ECS コンテナエージェントの設定」を参照してください。
- 実行されていないコンテナと未使用のイメージをコンテナインスタンスから削除できます。次のコマンドの例を使用して、停止したコンテナと未使用のイメージを手動で削除することができます。削除されたコンテナを後から検査することはできませんので、削除されたイメージは、新しいコンテナをスタートする前に、再度プルする必要があります。

  実行中ではないコンテナを削除するには、コンテナインスタンスで以下のコマンドを実行します。

  ```
  docker rm $(docker ps -aq)
  ```

  使用しないイメージを削除するには、コンテナインスタンスで以下のコマンドを実行します。

  ```
  docker rmi $(docker images -q)
  ```

- コンテナ内で使用されていないデータブロックは削除できます。以下のコマンドを使用して、実行中のコンテナで fstrim を実行し、コンテナファイルシステムによって使用されていないデータブロックを破棄できます。

```
sudo sh -c "docker ps -q | xargs docker inspect --format='{{ .State.Pid }}' | xargs -IZ fstrim /proc/Z/root/"
```

Amazon ECS Exec に関する問題のトラブルシューティング

ECS Exec の使用時にエラーが発生する理由を診断するトラブルシューティングに関する注意事項を以下に示します。

Execチェッカーを使用して検証する

ECS Exec チェッカー スクリプトを使用すると、Amazon ECS クラスターとタスクが ECS Exec 機能を使用するための前提条件を満たしていることを確認および検証できます。ECS Exec チェッカースクリプトは、AWS CLI 環境およびクラスターを両方認証し、お客様に代わってさまざまな API を呼び出すことで、タスクが ECS Exec に対応できるようにします。このツールには、AWS CLI の最新バージョンとjqが利用可能であることが必要です。詳細については、GitHub で「ECS Exec チェッカー」を参照してください。

`execute-command` 呼び出し時のエラー

`The execute command failed` エラーが発生した場合は、以下の原因が考えられます。

- タスクに必要なアクセス許可がありません。タスクの起動に使用されるタスク定義にタスク IAM のロールが定義されていること、およびロールに必要なアクセス許可があることを確認します。詳細については、「ECS Exec のアクセス許可」を参照してください。
- SSM エージェントがインストールされていないか、実行されていません。
- Amazon ECS 用のインターフェイス Amazon VPC エンドポイントはありますが、システムマネージャーセッションマネージャー用のエンドポイントはありません。

Amazon ECS Anywhere に関する問題のトラブルシューティング

Amazon ECS Anywhere は、オンプレミスサーバーや仮想マシン (VM) などの外部インスタンスを Amazon ECS クラスターに登録するためのサポートを提供します。以下は、発生する可能性のある一般的な問題と、一般的なトラブルシューティングの推奨事項です。

トピック

- 外部インスタンス登録の問題
- 外部インスタンスネットワークの問題
- 外部インスタンスでのタスクの実行に関する問題

外部インスタンス登録の問題

Amazon ECS クラスターに外部インスタンスを登録する場合、次の要件を満たす必要があります:

- アクティベーション ID およびアクティベーションコードで構成される、AWS Systems Manager のアクティベーションを取得する必要があります。Systems Manager マネージドインスタンスとして、外部 インスタンスを登録するために使用します。Systems Manager の有効化が要求されたら、登録制限と有効期限を指定します。登録制限は、アクティベーションを使用して登録できるインスタンスの最大数を指定します。登録制限のデフォルト値は 1 instance です。有効期限は、アクティベーションが期限切れになる日付です。デフォルト値は 24 時間です。外部インスタンスの登録に使用している Systems Manager のアクティベーションが有効でない場合は、新しいものをリクエストします。詳細については、「Amazon ECS クラスターに外部インスタンスを登録する」を参照してください。
- IAM ポリシーは、外部インスタンスが AWS API 操作との通信に必要なアクセス許可を提供するために使用されます。このマネージド ポリシーが正しく作成されず、必要なアクセス権限が含まれていない場合、外部インスタンスの登録は失敗します。詳細については、「Amazon ECS Anywhere IAM ロール」を参照してください。
- Amazon ECS には、Docker、Amazon ECS コンテナエージェント、および Systems Manager Agent を外部インスタンスにインストールするインストールスクリプトが用意されています。インストールスクリプトが失敗した場合、エラーが発生しなくても、同じインスタンスでスクリプトを再実行できない可能性があります。このような場合は、クリーンアッププロセスに従ってAWSリソースをインスタンスから削除して、再度インストールスクリプトを実行できます。詳細については、「Amazon ECS 外部インスタンスの登録を解除する」を参照してください。

> **Note**
>
> インストールスクリプトが Systems Manager のアクティベーションを正常に要求し、使用した場合、インストールスクリプトを 2 回実行すると Systems Manager のアクティベーションが再び使用されることに注意してください。これにより、順番にアクティベー

ションの登録制限に達する可能性があります。この制限に達した場合、新しいアクティベーションを作成する必要があります。

- GPU ワークロードの外部インスタンスでインストールスクリプトを実行するときに、NVIDIA ドライバが検出されない、または正しく設定されていない場合、エラーが発生します。インストールスクリプトは `nvidia-smi` コマンドを実行して、NVIDIA ドライバの存在を確認します。

外部インスタンスネットワークの問題

変更内容を伝えるには、外部インスタンスはAWSにネットワーク接続が必要です。外部インスタンスがAWSへのネットワーク接続が切断された場合、マニュアルで停止しない限り、インスタンスで実行されているタスクは、引き続き実行されます。AWS への接続後が復元されると、外部インスタンスの Amazon ECS コンテナエージェントと Systems Manager Agent によって使用される AWS 認証情報は自動的に更新されます。外部インスタンスとAWSの間の通信に使用されるAWSドメインの詳細については、「ネットワーク」を参照してください。

外部インスタンスでのタスクの実行に関する問題

タスクまたはコンテナが外部インスタンスで実行されない場合、最もよくある原因はネットワークまたはアクセス許可に関連しています。コンテナがAmazon ECRからイメージを引き出している場合や、コンテナのログをCloudWatch Logsに送信するように設定されている場合は、タスク定義で有効なタスク実行IAMロールを指定する必要があります。有効なタスク実行 IAM ロールがない場合、コンテナは起動しません。ネットワーク関連の問題の詳細については、「外部インスタンスネットワークの問題」を参照してください。

Important

Amazon ECS には、Amazon ECS ログ収集ツールが用意されています。これを使用して、トラブルシューティングの目的で外部インスタンスからログを収集することができます。詳細については、「Amazon ECS ログコレクターを使用したコンテナログの収集」を参照してください。

Fargate での Java クラス読み込みに関する問題のトラブルシューティング

Fargate で実行されている Java アプリケーションでは、特にアプリケーションが非決定論的クラスの読み込み動作に依存している場合、プラットフォームの更新後にクラス読み込みの問題が発生する可能性があります。これは、依存関係インジェクションエラー、Spring Boot 障害、または以前のデプロイで存在しなかったその他のランタイム例外として現れる可能性があります。

症状

次の症状が発生する場合があります。

- Spring Boot 依存関係インジェクションエラー
- ClassNotFoundException または NoClassDefFoundError 例外
- 以前は Fargate で動作していたアプリケーションが断続的に失敗するようになった
- 同じコンテナイメージが Amazon EC2 では動作するが、Fargate では失敗する
- 同一のコンテナイメージを使用したデプロイ間で動作に一貫性がない

原因

これらの問題は通常、次の原因で発生します。

- 非決定論的クラスの読み込み: クラスが JAR ファイルからロードされる順序に依存する Java アプリケーションは、基盤となるプラットフォームでファイルへのアクセス方法やキャッシュ方法が変更されると失敗する可能性があります。
- プラットフォームの更新: Fargate プラットフォームバージョンの更新により、基盤となるファイルシステムの動作が変更され、クラスの検出と読み込みの順序に影響を及ぼす可能性があります。
- JAR ファイルの順序付けにおける依存関係: 明示的な依存関係管理を行わずに、特定の JAR 読み込みっ順序に暗黙的に依存するアプリケーション。

解決策

Fargate での Java クラス読み込みの問題を解決するには、決定論的クラスの読み込みプラクティスを実装します。

即時修正

即時の回避策が必要な場合は次の操作を実行します。

1. JAR 読み込み順序を適用する: アプリケーションのクラスパス設定で JAR ファイルを読み込む順序を明示的に指定します。
2. 明示的な依存関係管理を使用する: 推移的な依存関係に依存するのではなく、すべての依存関係がビルド設定 (Maven、Gradle など) で明示的に宣言されていることを確認します。

長期的なベストプラクティス

以下のプラクティスを実装すると、今後のクラス読み込みの問題を防ぐことができます。

1. クラスの読み込みを決定論的にする
   - ビルドファイルで明示的な依存関係宣言を使用します
   - クラスパスのスキャン順序に依存しないようにします
   - 依存関係管理ツールを使用してバージョン競合を解決します
   - `-verbose:class` などの Java 仮想マシン (JVM) オプションを使用して、JVM で読み込まれたクラスに関する情報を取得します。
2. Spring Boot アプリケーション
   - 明示的なベースパッケージで `@ComponentScan` を使用します
   - 明示的に Bean を設定して自動設定の競合を回避します
   - `@Depends0n` 注釈を使用して Bean の初期化順序を制御します
3. ビルド設定
   - Maven または Gradle の依存関係管理セクションを使用します
   - 競合の原因となる推移的な依存関係を除外します
   - Maven Enforcer Plugin などのツールを使用して依存関係の問題を検出します
4. テスト
   - さまざまな JVM 実装でアプリケーションをテストします
   - さまざまなデプロイ環境をシミュレートする統合テストを実行します
   - ツールを使用して開発中のクラスパスの競合を分析します

防止

今後のデプロイで Java クラス読み込みの問題を防ぐには、以下を実行します。

- 決定論的クラスの読み込みプラクティスに従う: クラスパスからクラスを読み込む順序に依存しないようにアプリケーションを設計します。
- 明示的な依存関係管理を使用する: ビルド設定で、必要なすべての依存関係とそのバージョンを常に明示的に宣言します。
- 環境間でテストする: さまざまな環境やプラットフォームバージョンでアプリケーションを定期的にテストして、潜在的な問題を早期に特定します。
- プラットフォームの更新をモニタリングする: Fargate プラットフォームの更新に関する最新情報を入手し、本番環境のワークロードに影響を与える前に新しいプラットフォームバージョンでアプリケーションをテストします。

AWS Fargate スロットリングのクォータ

AWS Fargate は、リージョンごとに各 AWS アカウントのトークンバケットアルゴリズムを使用して、Amazon ECS タスクと Amazon EKS ポッドの起動レートをクォータ (以前は制限と呼ばれていました) に制限します。このアルゴリズムでは、アカウントには、特定の数のトークンを保持するバケットがあります。バケット内のトークンの数は、特定の秒におけるレートクォータを表します。各カスタマーアカウントには、タスクとポッドのトークンバケットがあり、カスタマーアカウントが起動したタスクとポッドの数に応じて枯渇していきます。このトークンバケットは、定期的により多くのリクエストを行うことができるバケットの最大値と、必要に応じて一定数のリクエストを維持できるリフィルレートを持ちます。

例えば、Fargate カスタマーアカウントのタスクとポッドトークンのバケットサイズは 100 トークンで、リフィルレートは毎秒 20 トークンです。したがって、カスタマーアカウントごとに最大 100 の Amazon ECS タスクと Amazon EKS ポッドをすぐに起動できます。持続的な起動レートは、1 秒あたり 20 の Amazon ECS タスクと Amazon EKS ポッドです。

アクション	バケット最大容量 (またはバーストレート)	バケット補充率 (または持続率)
オンデマンドの Amazon ECS タスクおよび Amazon EKS ポッドの Fargate リソースレートクォータ[1]	100	20

アクション	バケット最大容量 (またはバーストレート)	バケット補充率 (または持続率)
スポット Amazon ECS タスクの Fargate リソースレートクォータ	100	20

[1]Amazon EKS ポッドのみを起動するアカウントのバーストレートは 20 で、Amazon EKS プラットフォームバージョンで呼び出されるプラットフォームバージョンを使用する場合、1 秒あたりの持続的なポッド起動レートは 20 です。

Fargate での **RunTask** API のスロットリング

さらに、Fargate は Amazon ECS RunTask API を使用したタスクの起動時に、別のクォータを使用してリクエストレートを制限します。Fargate は、リージョンごとに、各 AWS アカウントの Amazon ECS RunTask API リクエストを制限します。各リクエストは、バケットから 1 つのトークンが削除されます。これは、サービスのパフォーマンスを向上し、Fargate のすべてのお客様に平等にご利用いただくために行います。API コールは、呼び出し元が Amazon Elastic Container Service コンソール、コマンドラインツール、サードパーティーアプリケーションのどれであるかに関係なく、リクエストクォータの対象となります。Amazon ECS RunTask API への呼び出しのレートクォータは毎秒 20 コール (バーストおよび持続的) です。ただし、この API を呼び出すたびに、最大 10 のタスクを起動できます。つまり、この API に 10 回の呼び出しを行い、各呼び出しで 10 のタスクを起動するように要求することで、1 秒で 100 のタスクを起動できます。同様に、この API に 20 回の呼び出しを行い、各呼び出しで 5 つのタスクを起動するように要求することもできます。Amazon ECS RunTask API の API スロットリングの詳細については、Amazon ECS API リファレンスの「API リクエストのスロットリング」を参照してください。

実際には、タスクやポッドの起動レートは、ダウンロードおよび解凍するコンテナイメージ、ヘルスチェック、タスクやポッドをロードバランサーに登録するなどの統合を有効にする他の考慮事項にも依存します。お客様が有効にした機能により、以前のクォータに比べるとタスクとポッドの起動レートにばらつきが見られます。

Fargate でのレートクォータの調整

AWS アカウントの Fargate レートスロットリングクォータの増加をリクエストできます。詳細については、「サービスクォータ ユーザーガイド」の「Requesting a quota increase」を参照してください。

Amazon ECS マネージドインスタンスのトラブルシューティング

次の手順を使用して、一般的な問題、診断手法、解決手順など、Amazon ECS マネージドインスタンスのトラブルシューティングを行います。

前提条件

Amazon ECS マネージドインスタンスのトラブルシューティングを行う前に、次の要件を満たしていることを確認してください。

- AWS CLI が適切なアクセス許可でインストールおよび設定されている

  詳細については、「AWS Command Line Interfaceユーザーガイド」の「AWS Command Line Interfaceの最新バージョンのインストールまたは更新」を参照してください。
- Amazon ECS マネージドインスタンスキャパシティプロバイダーによるクラスターへのアクセス
  詳細については、「the section called “Amazon ECS マネージドインスタンスのクラスターを作成する”」を参照してください。

一般的なトラブルシューティングのシナリオ

Amazon ECS マネージドインスタンスコンテナエージェントログの表示

これらの Amazon ECS ログファイルは、インスタンスで実行されている特権コンテナに接続することで、Amazon ECS マネージドインスタンスで表示できます。

診断手順

Amazon ECS タスクとして権限と Linux 機能を備えたデバッグコンテナをデプロイします。

次の環境変数を設定します。

user-input を独自の値に置き換えます。

```
export ECS_CLUSTER_NAME="your-cluster-name"
export AWS_REGION="your-region"
export ACCOUNT_ID="your-account-id"
```

node-debugger.json という CLI JSON ファイルを使用してタスク定義を作成します。

```
cat << EOF > node-debugger.json
```

```
{
  "family": "node-debugger",
  "taskRoleArn": "arn:aws:iam::${ACCOUNT_ID}:role/ecsTaskExecutionRole",
  "executionRoleArn": "arn:aws:iam::${ACCOUNT_ID}:role/ecsTaskExecutionRole",
  "cpu": "256",
  "memory": "1024",
  "networkMode": "host",
  "pidMode": "host",
  "requiresCompatibilities": ["MANAGED_INSTANCES", "EC2"],
  "containerDefinitions": [
    {
      "name": "node-debugger",
      "image": "public.ecr.aws/amazonlinux/amazonlinux:2023",
      "essential": true,
      "privileged": true,
      "command": ["sleep", "infinity"],
      "healthCheck": {
          "command": ["CMD-SHELL", "echo debugger || exit 1"],
          "interval": 30,
          "retries": 3,
          "timeout": 5
      },
      "linuxParameters": {
        "initProcessEnabled": true
      },
      "mountPoints": [
        {
          "sourceVolume": "host-root",
          "containerPath": "/host",
          "readOnly": false
        }
      ],
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "/aws/ecs/node-debugger",
          "awslogs-create-group": "true",
          "awslogs-region": "${AWS_REGION}",
          "awslogs-stream-prefix": "ecs"
        }
      }
    }
  ],
  "volumes": [
```

```
    {
      "name": "host-root",
      "host": {
        "sourcePath": "/"
      }
    }
  ]
}
EOF
```

タスクを登録して実行します。以下のコマンドを実行します。

```
aws ecs register-task-definition --cli-input-json file://node-debugger.json

TASK_ARN=$(aws ecs run-task \
  --cluster $ECS_CLUSTER_NAME \
  --task-definition node-debugger \
  --enable-execute-command \
  --capacity-provider-strategy capacityProvider=managed-instances-default,weight=1 \
  --query 'tasks[0].taskArn' --output text)

# Wait for task to be running
aws ecs wait tasks-running --cluster $ECS_CLUSTER_NAME --tasks $TASK_ARN
```

コンテナに接続します。以下のコマンドを実行してください。

```
aws ecs execute-command \
  --cluster $ECS_CLUSTER_NAME \
  --task $TASK_ARN \
  --container node-debugger \
  --interactive \
  --command "/bin/sh"
```

Amazon ECS エージェントログを確認します。

コンテナのインタラクティブセッションで、次のコマンドを実行します。

```
# Install required tools
yum install -y util-linux-core

# View ECS agent logs
nsenter -t 1 -m -p cat /var/log/ecs/ecs-agent.log | tail -50
```

```
# Check agent registration
nsenter -t 1 -m -p grep "Registered container instance" /var/log/ecs/ecs-agent.log

Example Output:

{"level":"info","time":"2025-10-16T12:39:37.665","msg":"Registered container instance
 with cluster!"}

# Verify capabilities
nsenter -t 1 -m -p grep "Response contained expected value for attribute" /var/log/ecs/
ecs-agent.log
```

エージェントメトリクスを確認します。

次のコマンドを実行して、ログを表示します。

```
# View metrics logs
nsenter -t 1 -m -p cat /var/log/ecs/metrics.log | tail -20
```

タスク配置の問題

以下は、タスク配置の問題で見られる症状です。

- タスクが保留中状態でスタックする
- Amazon ECS マネージドインスタンスでタスク開始に失敗する
- リソース不足のエラーが発生する

診断手順

次のコマンドを実行して、タスク配置の問題を診断し、クラスター容量、コンテナインスタンス、システムサービスに関する情報を収集します。

```
# Check cluster capacity
aws ecs describe-clusters --clusters cluster-name --include STATISTICS

# Check cluster capacity providers
aws ecs describe-clusters --clusters cluster-name --include STATISTICS --query
 'clusters[].capacityProviders'

# List container instances
aws ecs list-container-instances --cluster cluster-name
```

```
# Check container instance details
aws ecs describe-container-instances --cluster cluster-name --container-
instances container-instance-arn

# Check container instance remaining resources CPU/Mem
aws ecs describe-container-instances --cluster $ECS_CLUSTER_NAME --container-
instances container-instance-arn --query 'containerInstances[].remainingResources'

# Check container instance Security Group
aws ecs describe-container-instances --cluster $ECS_CLUSTER_NAME --container-
instances container-instance-arn --query 'containerInstances[].ec2InstanceId' --output
 text
aws ec2 describe-instances --instance-ids instance-id --query
 'Reservations[0].Instances[0].SecurityGroups'
aws ec2 describe-security-groups --group-ids security-group-id
```

システムサービスのモニタリング:

```
# Check Containerd status
nsenter -t 1 -m -p systemctl status containerd.service

# Check Amazon ECS container agent status
nsenter -t 1 -m -p systemctl status ecs
```

解決策

タスク配置の問題を解決するには、以下の手順に従って、適切な設定と容量であることを確認してください。

- タスクリソースの要件と使用可能な容量を検証する
- 配置の制約と戦略を確認する
- Amazon ECS マネージドインスタンスのキャパシティプロバイダーが設定されていることを確認する
- タスクとコンテナインスタンスのセキュリティグループに、Amazon ECS エージェント管理エンドポイントのトラフィックを許可するアウトバウンドルールがあることを確認する

ネットワーキングの問題

ネットワークの問題では、次のような症状が見られます。

- タスクが外部サービスに到達できない
- DNS 解決に問題がある

診断手順

ネットワーク接続のテスト:

デバッグコンテナで次のコマンドを実行します。

> **Note**
>
> キャパシティプロバイダーまたは Amazon ECS タスクにアタッチされたセキュリティグループがトラフィックを許可していることを確認します。

```
# Install DNS Utility
yum install bind-utils -y

# Test DNS resolution
nslookup amazon.com

# Test external connectivity
curl -I https://amazon.com
```

リソースの制約

ネットワークの問題では、次のような症状が見られます。

- タスクがメモリ制限により強制終了される
- CPU スロットリングが発生する
- ディスク容量に問題がある

診断手順

コマンドを実行して、リソースとコンテナの制限をモニタリングします。

リソースのモニタリング:

```
# Check memory usage
```

```
nsenter -t 1 -m -p free -h

# Check disk usage
nsenter -t 1 -m -p lsblk

# Check disk usage
nsenter -t 1 -m -p df -h
```

コンテナの制限:

```
# Check OOM kills
nsenter -t 1 -m -p dmesg | grep -i "killed process"
```

コンテナインスタンスエージェントが切断される問題

コンテナインスタンスエージェントの切断の問題には、次のような症状が見られます。

- Amazon ECS コンソールで、コンテナインスタンスが切断されたと表示される
- 特定のインスタンスにタスクを配置できない
- ログでエージェント登録が失敗している

診断手順

ECS Exec がアクセス可能なホストで既存の権限タスクが実行中の場合は、次のコマンドを実行してエージェント接続の問題を診断します。

```
# check service status
nsenter -t 1 -m -p systemctl status ecs
nsenter -t 1 -m -p systemctl status containerd

# restart stopped services
nsenter -t 1 -m -p systemctl restart ecs
nsenter -t 1 -m -p systemctl restart containerd
```

それ以外の場合は、Amazon ECS マネージドインスタンスを強制的に登録解除します。次のコマンドを実行してください:

```
# list ECS Managed Instance container
aws ecs list-container-instances --cluster managed-instances-cluster --query
 'containerInstanceArns' --output text
```

```
# deregister the specific container instance
aws ecs deregister-container-instance \
    --cluster $ECS_CLUSTER_NAME \
    --container-instance container-instance-arn \
    --force
```

解決策

エージェントの切断の問題を解決するには、次の手順に従います。

- コンテナインスタンスの IAM ロールのアクセス許可を検証する
- セキュリティグループルールが、ECS エンドポイントへのアウトバウンド HTTPS トラフィックを許可していることを確認する
- AWS サービスへのネットワーク接続を確認する
- 必要に応じて、`nsenter -t 1 -m -p systemctl restart ecs` で ECS エージェントサービスを再起動する
- /etc/ecs/ecs.config の ECS_CLUSTER 設定がクラスター名と一致することを確認する

Amazon ECS マネージドインスタンスのログ分析

システムログ

次のコマンドを使用して、システムログを確認し、マネージドインスタンスの潜在的な問題を特定します。

```
# Check system messages
nsenter -t 1 -m -p journalctl --no-pager -n 50

# Check kernel logs
nsenter -t 1 -m -p dmesg | tail -20

# Check for disk space errors
nsenter -t 1 -m -p journalctl --no-pager | grep -i "no space\|disk full\|enospc"
```

EC2 AWS CLI を使用して Amazon ECS マネージドインスタンスからコンソール出力を取得する

Amazon EC2 インスタンス ID を使用してコンソール出力を取得します。

user-input を独自の値に置き換えます。

```
aws ec2 get-console-output --instance-id instance-id --latest --output text
```

クリーンアップ

以下を実行して、デバッグタスクを停止し、タスク定義を登録解除します。

```
# Stop debug task
aws ecs stop-task --cluster $ECS_CLUSTER_NAME --task $TASK_ARN

# Deregister task definition (optional)
aws ecs deregister-task-definition --task-definition node-debugger
```

その他のリソース

Amazon ECS マネージドインスタンスのトラブルシューティングの詳細については、次のリソースを参照してください。

- the section called “Amazon ECS マネージドインスタンスエラー”
- トラブルシューティング
- the section called “コンテナエージェントの設定”
- the section called “ECS Exec を使用して Amazon ECS コンテナをモニタリングする”

Amazon ECS マネージドインスタンスのトラブルシューティング

Amazon ECS マネージドインスタンスでタスクを起動する場合、Amazon ECS は、まず既存の容量へのタスク配置を試行し、配置できないタスクに対する追加の容量を要求します。インスタンスのプロビジョニングが失敗した場合、Amazon EC2 リクエスト ID がタスク失敗メッセージに含まれています。このリクエスト ID を使用して、CloudTrail で失敗したリクエストの詳細を検索し、トラブルシューティングに活用することができます。

> **Note**
>
> AmazonECSInstanceRolePolicyForManagedInstances 管理ポリシーを使用する代わりに、アクセスの最小特権を適用し、インスタンスプロファイルに独自のアクセス許可を指定する場合は、次のアクセス権限を追加して、Amazon ECS マネージドインスタンスのタスク関連の問題のトラブルシューティングに役立てることができます。

- ecs:StartTelemetrySession
- ecs:PutSystemLogEvents

タスク定義に Amazon ECS マネージドインスタンスとの互換性がありません

一般的な原因

このエラーは、Amazon ECS マネージドインスタンスでサポートされていないパラメータや設定がタスク定義に含まれている場合に発生します。一般的には、サポートされていないネットワークモード、タスクロール、リソース要件などが互換性のない原因になります。

解決策

1. タスク定義で、MANAGED_INSTANCES に設定した requiresCompatibilities を使用していることを確認します。
2. タスク定義で awsvpc ネットワークモードを使用していることを確認します。
3. CPU とメモリの値が、Amazon ECS マネージドインスタンスのサポート範囲内にあることを確認します。
4. 特定の互換性問題の詳細については、エラーメッセージの細部を確認してください。

キャパシティプロバイダーがクラスターに関連付けられていません

一般的な原因

このエラーは、キャパシティプロバイダー戦略で指定されたキャパシティプロバイダーがクラスターに関連付けられていないか、当該のプロバイダーが存在しない場合に発生します。

解決策

1. キャパシティプロバイダーがアカウントとリージョンに存在することを確認します。
2. Amazon ECS コンソールまたは CLI を使用して、キャパシティプロバイダーをクラスターに関連付けます。
3. 使用する前に、キャパシティプロバイダーが ACTIVE 状態であることを確認します。

インフラストラクチャロールのアクセス許可エラー

一般的な原因

このエラーは、Amazon ECS インフラストラクチャロールに、ユーザーに代わって Amazon EC2 オペレーションを実行するために必要なアクセス許可がない場合、または信頼関係の問題によりロールを引き受けることができない場合に発生します。

解決策

1. インフラストラクチャロールに Amazon ECS との適切な信頼関係があることを確認します。
2. ロールに必要な Amazon EC2 アクセス許可 (`ec2:RunInstances`、`ec2:DescribeInstances`、`iam:PassRole` など) があることを確認します。
3. 特定のアクセス許可の詳細については、CloudTrail のエンコードされた認可失敗メッセージを確認してください。
4. ロールポリシーを更新して、不足しているアクセス許可 (エラーメッセージで確認する) を含めます。

VcpuLimitExceeded エラー

一般的な原因

このエラーは、現在のリージョンでインスタンスタイプファミリーの vCPU サービスクォータに達した場合に発生します。Amazon ECS マネージドインスタンスは、容量が利用可能になるまで追加のインスタンスを起動できません。

解決策

1. AWS サポートセンターを通じて、影響を受けるインスタンスタイプファミリー用のサービスクォータの引き上げをリクエストします。
2. 異なる vCPU クォータカテゴリに分類されるインスタンスタイプを別に使用することを検討してください。
3. 未使用の Amazon EC2 インスタンスを終了して vCPU 容量を解放します。
4. vCPU 要件が低いインスタンスタイプを使用する場合は、キャパシティプロバイダーの設定を確認します。

InsufficientCapacity と関連するキャパシティエラー

一般的な原因

これらのエラーは、インスタンスリクエストに対応する十分な容量が AWS にない場合に発生します。この場合、リクエストされたアベイラビリティーゾーンのインスタンス容量、アドレス容量、ボリューム容量などが不足してる可能性があります。

解決策

1. キャパシティプロバイダーで複数のサブネットを設定して、別のアベイラビリティーゾーンでインスタンスを起動してみてください。
2. 使用可能な容量が多い可能性のある別のインスタンスタイプを使用することを検討してください。
3. キャパシティの可用性は頻繁に変化しますので、オペレーションを待機して再試行します。
4. 永続的な容量が必要な場合は、リザーブドインスタンスまたは Savings Plans の使用を検討してください。

UnauthorizedOperation エラー

一般的な原因

このエラーは、Amazon ECS サービスに Amazon EC2 オペレーションを実行した場合や、IAM ロールを渡すために必要なアクセス許可がない場合に発生します。一般的なシナリオとして、インスタンスプロファイルの `ec2:RunInstances` アクセス許可または `iam:PassRole` アクセス許可が不足していることが挙げられます。

解決策

1. Amazon ECS インフラストラクチャロールに、Amazon EC2 インスタンスの起動に必要なアクセス許可があることを確認します。
2. インフラストラクチャロールに、Amazon ECS マネージドインスタンスで使用されるインスタンスプロファイルに対する `iam:PassRole` アクセス許可があることを確認します。
3. 特定のアクセス許可の詳細については、CloudTrail のエンコードされた認可失敗メッセージを確認してください。
4. ロールポリシーを更新して、不足しているアクセス許可 (エラーメッセージで確認する) を含めます。

キャパシティの待機中にタスクがタイムアウトしました

一般的な原因

このエラーは、インスタンスの起動とクラスターへの登録に想定以上の時間を要する場合に発生します。これは、Amazon EC2 の容量の制約、インスタンスの起動の失敗、ネットワーク接続の問題が原因で発生する可能性があります。

解決策

1. 進行中の問題がないか、リージョンの Amazon EC2 サービスの健全性を確認します。
2. サブネットに十分な IP アドレスがあることを確認します。
3. セキュリティグループで Amazon ECS エージェント通信に必要なトラフィックを許可していることを確認します。
4. キャパシティの可用性を向上させるには、複数のアベイラビリティーゾーンを使用することを検討してください。
5. キャパシティの制約は一時的である場合が多いため、タスク起動オペレーションを再試行します。

ネットワーク設定エラー

一般的な原因

これらのエラーは、タスクのネットワーク要件と、キャパシティプロバイダーのネットワーク設定の間に不一致がある場合に発生します (VPC の不一致やネットワーク設定の不足など)。

解決策

1. キャパシティプロバイダーが正しい VPC とサブネットで設定されていることを確認します。
2. セキュリティグループとサブネットが同じ VPC に属していることを確認します。
3. タスク定義のネットワーク設定にキャパシティプロバイダーとの互換性があることを確認します。
4. 正しいネットワーク設定でキャパシティプロバイダー設定を更新します。

インスタンスがスタックしているためキャパシティプロバイダーを削除できません

一般的な原因

これらのエラーは、Amazon ECS マネージドインスタンスが ACTIVE または DRAINING の状態でスタックしているが、インスタンスに実行中のタスクがない場合に発生します。

解決策

キャパシティプロバイダーの削除を続行できるようにするには、次のコマンドを使用して、スタックしているインスタンスを強制的に登録解除します。

```
aws ecs deregister-container-instance \
    --cluster arn:aws:ecs:us-east-1:111122223333:cluster/MyCluster \
    --container-instance arn:aws:ecs:us-east-1:111122223333:container-instance/
a1b2c3d4-5678-90ab-cdef-11111EXAMPLE \
    --force
```

Amazon ECS Amazon ECS マネージドインスタンスエラーのトラブルシューティング

Amazon ECS マネージドインスタンスのエラーメッセージと、エラーを修正するために実行できるアクションの一部を以下に示します。

クラスター名のない Amazon ECS マネージドインスタンスプロバイダーはサポートされません

このエラーは、Amazon ECS マネージドインスタンスプロバイダーのパラメータを使用して、クラスターフィールドなしでキャパシティプロバイダーを作成しようとすると発生します。

この問題を解決するには、キャパシティプロバイダーの作成時に有効なクラスター名を指定します。

Amazon ECS マネージドインスタンスキャパシティプロバイダーの作成には Amazon ECS マネージドインスタンスプロバイダーの詳細が必要です

このエラーは、Amazon ECS マネージドインスタンスプロバイダーで、実際のオブジェクトを指定せずにキャパシティプロバイダーを作成しようとした場合に表示されます。

この問題を修正するには、有効な Amazon ECS マネージドインスタンスプロバイダーの詳細を指定して、もう一度試してください。

Amazon ECS マネージドインスタンスキャパシティプロバイダーはインスタンスプロファイルを指定する必要があります

このエラーは、Amazon ECS マネージドインスタンスプロバイダーで、Ec2InstanceProfile を指定せずにキャパシティプロバイダーを作成しようとした場合に表示されます。

これを解決するには、Amazon ECS マネージドインスタンスキャパシティプロバイダーの有効な EC2 インスタンスプロファイルを指定します。詳細については、「the section called “Amazon ECS マネージドインスタンスのインスタンスプロファイル”」を参照してください。

Amazon ECS マネージドインスタンスキャパシティプロバイダーは InfrastructureRole を指定する必要があります

このメッセージは、Amazon ECS マネージドインスタンスプロバイダーで、InfrastructureRole を指定せずにキャパシティプロバイダーを作成しようとした場合に表示されます。

これを修正するには、Amazon ECS マネージドインスタンスキャパシティプロバイダーの有効なインフラストラクチャロールを指定します。詳細については、「the section called “インフラストラクチャ IAM ロール”」を参照してください。

Amazon ECS マネージドインスタンスキャパシティプロバイダーはネットワーク設定を指定する必要があります

このエラーは、Amazon ECS マネージドインスタンスプロバイダーで、ネットワーク設定を指定せずにキャパシティプロバイダーを作成しようとした場合に発生します。

これを解決するには、Amazon ECS マネージドインスタンスキャパシティプロバイダーに、空のサブネットがない有効なネットワーク設定を指定します。詳細については、「the section called “Amazon ECS マネージドインスタンスのタスクネットワーキング”」を参照してください。

Amazon ECS マネージドインスタンスキャパシティプロバイダーに指定されたインスタンス要件を満たすインスタンスタイプがありません

これは、Amazon ECS マネージドインスタンスプロバイダーでキャパシティプロバイダーを作成しようとしたが、インスタンス要件を満たす EC2 インスタンスタイプがない場合に発生します。

これに対処するには、使用可能な EC2 インスタンスタイプに合わせてインスタンス要件を確認し、調整します。

指定されたインフラストラクチャロール ARN が無効です

このエラーは、InfrastructureRole が特定の ARN 形式に従っていない場合に発生します。

必要な形式は `arn:partition:iam::account-id:role/role-name` です。有効なロール ARN を指定して、もう一度試してください。詳細については、「the section called “インフラストラクチャ IAM ロール”」を参照してください。

指定されたインスタンスプロファイルロール ARN が無効です

このエラーは、インスタンスプロファイルが特定の ARN 形式に従っていない場合に発生します。

必要な形式は `arn:partition:iam::account-id:instance-profile/profile-name` です。有効なロール ARN を指定して、もう一度試してください。詳細については、「the section called “Amazon ECS マネージドインスタンスのインスタンスプロファイル”」を参照してください。

指定されたセキュリティグループ ID が無効です

このエラーは、セキュリティグループ ID が特定の形式に従っていない場合に発生します。

想定される形式は、`sg-xxxxxxxx` または `sg-xxxxxxxxxxxxxxxxx` (「sg–」以降は文字 (小文字) と数字を含む 8 文字または 17 文字) です。セキュリティグループ ID を指定して、もう一度試してください。

指定されたサブネット ID が無効です

このエラーは、サブネット ID が特定の形式に従っていない場合に発生します。

想定される形式は、`subnet-xxxxxxxx` または `subnet-xxxxxxxxxxxxxxxxx` (「subnet–」以降は文字 (小文字) と数字を含む 8 文字または 17 文字) です。サブネット ID を指定して、もう一度試してください。

Amazon ECS マネージドインスタンスキャパシティプロバイダーはサブネットが空ではないネットワーク設定を指定する必要があります

このエラーは、ネットワーク設定のサブネットが空の状態のまま Amazon ECS マネージドインスタンスプロバイダーでキャパシティプロバイダーを作成しようとすると発生します。

これを解決するには、Amazon ECS マネージドインスタンスキャパシティプロバイダーに、空のサブネットがない有効なネットワーク設定を指定します。

Amazon ECS マネージドインスタンスキャパシティプロバイダーのタグ数は最大許容値を超えることはできません

このエラーは、タグの最大許容数 (45) を超えるキャパシティプロバイダーを作成しようとすると発生します。

これを解決するには、タグの数を 45 以下に減らして、もう一度試してください。

propagateTags の値が無効です

このエラーは、無効な propagateTags 値を持つキャパシティプロバイダーを作成しようとすると発生します。

値は有効な propagateTags 値のいずれかである必要があります。有効な値を指定して、もう一度試してください。

指定されたキャパシティプロバイダーがクラススコープに既に存在します

このエラーは、クラスター名なしで Amazon ECS マネージドインスタンスキャパシティプロバイダーを作成、更新、削除しようとした際に、既存のクラススコープにキャパシティプロバイダーが存在する場合に発生します。

キャパシティプロバイダーの設定を変更または削除するには、クラスターパラメータを使用して、キャパシティプロバイダーを更新または削除します。

指定されたキャパシティプロバイダーがアカウントまたは別のクラスターに既に存在します

このエラーは、クラスターフィールドを使用して Amazon ECS マネージドインスタンスキャパシティプロバイダーを作成、更新、削除しようとした際に、既存のアカウントスコープのキャパシティプロバイダーか、別のクラスター用の既存のクラススコープのキャパシティプロバイダーが存在する場合に発生します。

これを解決するには、別のキャパシティプロバイダー名を使用するか、既存のキャパシティプロバイダーを操作します。

キャパシティプロバイダーのアクティブなクラスター名がリクエストのクラスター名と一致しません

このエラーは、キャパシティプロバイダーのアクティブなクラスター名が、リクエストで指定されたクラスター名と一致しない場合に発生します。

リクエストのクラスター名が、関連付けられたキャパシティプロバイダーのクラスターと一致していることを確認します。

キャパシティプロバイダーが、Amazon ECS マネージドインスタンスのキャパシティプロバイダーではありません

このエラーは、Amazon ECS マネージドインスタンスのキャパシティプロバイダーが必要な場面で、それ以外のキャパシティプロバイダーを使おうとすると発生します。

リクエストで、有効な Amazon ECS マネージドインスタンスキャパシティプロバイダーを使用していることを確認します。

CapacityProvider 名は空白にできません

このエラーは、キャパシティプロバイダー名を指定せずにキャパシティプロバイダーを作成しようとすると発生します。

有効なキャパシティプロバイダー名を指定して、もう一度試してください。

キャパシティプロバイダーの更新には名前が必要です

このエラーは、キャパシティプロバイダー名を指定せずにキャパシティプロバイダーを更新しようとすると発生します。

有効な名前を使用してもう一度試してください。

タグは null にできません。また、null 要素を含むこともできません

このエラーは、キャパシティプロバイダーに null 値をタグ付けしようとすると発生します。

すべてのタグ値が正しく指定され、null ではないことを確認します。

Amazon ECS での API エラーの原因

Amazon ECS API、コンソール、または AWS CLI でトリガーした API アクションが `failures` エラーメッセージを表示して終了した場合、以下が原因のトラブルシューティングに役立つことがあります。失敗すると、その理由と、その失敗に関連付けられたリソースの Amazon リソースネーム (ARN) が返されます。

多くのリソースはリージョン固有であるため、コンソールを使用するときは、リソースに正しいリージョンを設定してください。AWS CLI コマンドを使用するときは、AWS CLI コマンドが `--region` *`region`* パラメータで正しいリージョンに送信されるようにします。

Failureデータ型の構造の詳細については、「Amazon Elastic Container サービス API リファレンス」の[Failure(失敗)]を参照してください。

次の内容は、API コマンド実行時に受信する場合がある障害メッセージの例です。

API アクション	[失敗] の理由または [停止] の理由	原因
DescribeClusters	MISSING	指定されたクラスターは見つかりませんでした。クラスター名が正しく入力されているか確認します。
DescribeInstances	MISSING	指定されたコンテナインスタンスは見つかりませんでした。コンテナインスタンスが登録されているクラスターを指定したこと、およびコンテナインスタンスの ARN または ID の両方が正しいことを確認します。
DescribeServices	MISSING	指定されたサービスは見つかりませんでした。正しいクラスターまたはリージョンが指定されていること、サービス ARN またはサービス名が有効であることを確認します。
DescribeTasks	MISSING	指定されたタスクは見つかりませんでした。正しいクラスターまたはリージョンが指定されており、タスク ARN または ID の両方が有効であることを確認します。
DescribeTasks	TaskFailedToStart: RESOURCE:*	RESOURCE:CPU　エラーの場合、タスクによって要求され

API アクション	[失敗] の理由または [停止] の理由	原因
		た CPU の数がコンテナインスタンスで利用できません。これは通常、タスク定義の CPU ユニット要件が、キャパシティプロバイダーにマッピングされた Auto Scaling グループで定義される Amazon EC2 インスタンスの CPU サイズよりも大きい場合に発生します。キャパシティプロバイダーの設定を確認する必要があります。 RESOURCE:MEMORY エラーの場合、タスクによって要求されたメモリの量がコンテナインスタンスで利用できません。これは通常、タスク定義のメモリ量要件が、キャパシティプロバイダーにマッピングされた Auto Scaling グループで定義される Amazon EC2 インスタンスでサポートされているメモリよりも大きい場合に発生します。キャパシティプロバイダーの設定を確認する必要があります。

API アクション	[失敗] の理由または [停止] の理由	原因
	`TaskFailedToStart: AGENT`	タスクを起動しようとしたコンテナインスタンスに、現在接続されていないエージェントがあります。タスク配置の待ち時間が長くならないように、リクエストは拒否されました。 切断されたエージェントをトラブルシューティングする方法については、「切断された Amazon ECS エージェントをトラブルシューティングするにはどうすればよいですか?」を参照してください。
	`TaskFailedToStart: MemberOf placement constraint unsatisfied`	タスク定義で定義された配置制約を満たすコンテナインスタンスがありません。

API アクション	[失敗] の理由または [停止] の理由	原因
	TaskFailedToStart: ATTRIBUTE	タスク定義に、コンテナインスタンスで使用できない特定のコンテナインスタンス属性が必要なパラメータが含まれています。例えば、awsvpc ネットワークモードを使用するタスクであるものの、指定したサブネット内に、ecs.capability.task-eni 属性を持つインスタンスがありません。特定のタスク定義パラメータとエージェント設定変数に必要な属性の詳細については、「Fargate での Amazon ECS タスク定義パラメータ」と「Amazon ECS コンテナエージェントの設定」を参照してください。
	TaskFailedToStart: NO ACTIVE INSTANCES	キャパシティプロバイダーにアクティブなインスタンスがありません。Auto Scaling グループの管理方法については、「Amazon EC2 Auto Scaling ユーザーガイド」の「Auto Scaling グループ」を参照してください。

API アクション	[失敗] の理由または [停止] の理由	原因
	TaskFailedToStart: EMPTY CAPACITY PROVIDER	クラスターにはインスタンスはありません。これは、キャパシティープロバイダーが空であるか、キャパシティープロバイダーのインスタンスがクラスターに登録されていないことが原因と考えられます。Auto Scaling グループの管理方法については、「Amazon EC2 Auto Scaling ユーザーガイド」の「Auto Scaling グループ」を参照してください。
GetTaskProtection	MISSING	指定されたタスクは見つかりませんでした。クラスター名または ARN、タスク ARN または ID が有効であることを確認します。
	TASK_NOT_VALID	指定されたタスクは、Amazon ECS サービスの一部ではありません。保護できるのは Amazon ECS サービスの管理対象タスクだけです。タスク ARN または ID を確認して、もう一度お試しください。

API アクション	[失敗] の理由または [停止] の理由	原因
RunTask-または-StartTask	RESOURCE:*	タスクでリクエストされたリソースまたは複数のリソースは、クラスターのコンテナインスタンスで使用できません。リソースが CPU、メモリ、ポート、または Elastic Network Interface の場合は、クラスターへのその他のコンテナインスタンスを追加する必要がある場合があります。 RESOURCE:ENI エラーの場合、awsvpc ネットワークモードを使用するタスクで必要な Elastic Network Interface アタッチメントポイントが、クラスターで利用できません。Amazon EC2 インスタンスには、アタッチできるネットワークインターフェイスの数には制限があり、プライマリネットワークインターフェイスも 1 つ分としてカウントされます。各インスタンスタイプでサポートされるネットワークインターフェイスの数の詳細については、「Amazon EC2 ユーザーガイド」の「各インスタンスタイプのネットワークインターフェイスあたりの IP アドレス数」を参照してください。

API アクション	[失敗] の理由または [停止] の理由	原因
		RESOURCE:GPU エラーの場合、タスクでリクエストされた GPU 数が利用できないため、クラスターへ GPU 対応のコンテナインスタンスの追加が必要になる場合があります。詳細については、「GPU ワークロード向けの Amazon ECS タスク定義」を参照してください。
	AGENT	タスクを起動しようとしたコンテナインスタンスに、現在接続されていないエージェントがあります。タスク配置の待ち時間が長くならないように、リクエストは拒否されました。 切断されたエージェントをトラブルシューティングする方法については、「切断された Amazon ECS エージェントをトラブルシューティングするにはどうすればよいですか?」を参照してください。
	LOCATION	タスクを起動しようとしたコンテナインスタンスが、awsVpcConfiguration で指定したサブネットと異なるアベイラビリティーゾーンにあります。

API アクション	[失敗] の理由または [停止] の理由	原因
	ATTRIBUTE	タスク定義に、コンテナインスタンスで使用できない特定のコンテナインスタンス属性が必要なパラメータが含まれています。例えば、awsvpc ネットワークモードを使用するタスクであるものの、指定したサブネット内に、ecs.capability.task-eni 属性を持つインスタンスがありません。特定のタスク定義パラメータとエージェント設定変数に必要な属性の詳細については、「Fargate での Amazon ECS タスク定義パラメータ」と「Amazon ECS コンテナエージェントの設定」を参照してください。
StartTask	MISSING	タスクを起動しようとした対象のコンテナインスタンスが見つかりません。間違ったクラスターまたはリージョンが指定されていたり、コンテナインスタンスの ARN または ID が正しく記述されていなかったりしないかを確認してください。
	INACTIVE	タスクを起動しようとしたコンテナインスタンスは、Amazon ECS ですでに登録解除されており、使用できません。

API アクション	[失敗] の理由または [停止] の理由	原因
StopServiceDeployment	ECS deployment failed	不正アカウントが StopServiceDeployment API を実行しました。
TagResource	InvalidParameterException	タグ付けするサービスの ARN は短い形式です。長い形式に移行する必要があります。移行方法についての詳細は、「Amazon ECS の短いサービス ARN を長い ARN に移行する」を参照してください。
UpdateTaskProtection	DEPLOYMENT_BLOCKED	1 つ以上の保護されたタスクが原因で、サービスのデプロイが安定した状態に移行できず、タスク保護を設定できません。既存のタスクに対するタスク保護の設定を解除するか、タスク保護の有効期限が切れるまでお待ちください。
	MISSING	指定されたタスクは見つかりませんでした。クラスター名または ARN、タスク ARN または ID が有効であることを確認します。
	TASK_NOT_VALID	指定されたタスクは、Amazon ECS サービスの一部ではありません。保護できるのは Amazon ECS サービスの管理対象タスクだけです。タスク ARN または ID を確認して、もう一度お試しください。

> **Note**
>
> ここで説明した障害シナリオの他に、API 操作が例外によってエラーとなり、エラー応答が発生することもあります。このような例外のリストについては、「一般的なエラー」を参照してください。

Amazon Q Developer のトラブルシューティング

Amazon ECS コンソールで Amazon Q Developer を使用すると、Amazon ECS リソースに関する問題の診断と解決に役立ちます。コンテナ、タスク、サービス、デプロイ、タスク定義に関する特定のエラーとステータスメッセージについては、コンソールに [Amazon Q Developer を使用して検査する] オプションが表示されます。このオプションを選択すると、Amazon Q Developer が問題をコンテキストで分析し、考えられる原因と修復手順を提案します。

必要なアクセス許可

- クラスター、サービス、タスク、タスク定義など、トラブルシューティング対象の Amazon ECS リソースを表示するアクセス許可。
- コンソールで Amazon Q Developer を使用するアクセス許可。
- (推奨) 次のような関連するログとメトリクスを表示するアクセス許可。
  - CloudWatch Logs
  - CloudWatch

手順

1. https://console.aws.amazon.com/ecs/v2 でコンソールを開きます。
2. トラブルシューティングするリソースを決定します。

リソース	Steps
タスク	1. [クラスター] ページで、クラスターを選択します。 2. [タスク] タブを選択します。 3. 調査するタスクを選択します。

リソース	Steps
コンテナ	1. [クラスター] ページで、クラスターを選択します。 2. [タスク] タブを選択し、タスクを選択します。 3. [コンテナ] セクションで、調査するコンテナを選択します。
サービスとデプロイ	1. [クラスター] ページで、クラスターを選択します。 2. [サービス] タブを選択します。 3. サービスを選択し、[デプロイ] セクションを確認します。
タスク定義	1. ナビゲーションペインで、タスク定義 を選択します。 2. タスク定義ファミリーを選択し、調査するリビジョンを選択します。

3. [リソース詳細] ページで、問題を説明する状態または健全性の理由を見つけます。
4. 状態の理由をクリックして、ポップ画面を開きます。
5. 可能な場合は、[Amazon Q Developer を使用して検査する] を選択します。
6. Amazon Q Developer が提供する説明と、推奨される修復手順を確認します。環境に応じて、設定または運用上の変更を適用します。

考慮事項

Amazon ECS で Amazon Q Developer を使用する場合は、次の点を考慮してください。

- ボタンの可用性 – [Amazon Q Developer を使用して検査する] ボタンは、問題発生の可能性があるリソースに対してのみ表示されます。このオプションは、正常なリソースでは使用できません。
- 読み取り専用オペレーション – Amazon Q Developer 統合では、読み取りオペレーションのみを実行します。変更アクションや書き込みアクションは実行しません。

- クロスリージョン処理 – Amazon Q Developer は、AI を活用した分析を提供するために、AWS リージョン間でデータを処理することがあります。詳細については、「Amazon Q Developer でのクロスリージョン処理」を参照してください。
- コンソール専用 – この統合はコンソールでのみ使用できます。AWS CLI、AWS API、Infrastructure as Code ツールでは使用できません。

詳細情報

Amazon Q Developer の操作方法の詳細については、「AWS に関する Amazon Q Developer とのチャット」を参照してください。

Amazon Elastic Container Service のセキュリティ

AWS ではクラウドセキュリティが最優先事項です。セキュリティを最も重視する組織の要件を満たすために構築された 「AWS」 のデータセンターとネットワークアーキテクチャは、お客様に大きく貢献します。

セキュリティは、「AWS」 と顧客の間の責任共有です。責任共有モデルでは、この責任がクラウドのセキュリティおよびクラウド内のセキュリティとして説明されています。

- クラウドのセキュリティ - 「AWS」は、「AWS」クラウドで「AWS」のサービスを実行するインフラストラクチャを保護する責任を負います。また、「AWS」は、使用するサービスを安全に提供します。「AWS」 コンプライアンスプログラムの一環として、サードパーティーの監査が定期的にセキュリティの有効性をテストおよび検証しています。Amazon Elastic Container Service, に適用されるコンプライアンスプログラムについては、コンプライアンスプログラムによる対象範囲内のAWSサービス」を参照してください。
- クラウド内のセキュリティ - ユーザーの責任は、使用する AWS のサービスに応じて異なります。また、ユーザーは、データの機密性、会社の要件、適用される法律や規制など、その他の要因についても責任を負います。

このドキュメントは、Amazon ECS 使用時における責任共有モデルの適用法を理解するのに役立ちます。以下のトピックでは、セキュリティとコンプライアンスの目的を満たすように Amazon ECS を設定する方法について説明します。Amazon ECS リソースのモニタリングやセキュリティ確保に役立つ他の AWS サービスの使用方法についても説明します。

トピック

- Amazon Elastic コンテナサービスのアイデンティティ とアクセス管理
- Amazon Elastic Container Service でのログとモニタリング
- Amazon Elastic Container Service でのコンプライアンス検証
- AWS Fargate 連邦情報処理標準 (FIPS-140)
- Amazon Elastic Container Service におけるインフラストラクチャセキュリティ
- AWSAmazon ECS の 責任共有モデル。
- Amazon ECS マネージドインスタンスの責任共有モデル
- Amazon ECS のネットワークセキュリティのベストプラクティス
- Amazon ECS タスクおよびコンテナのセキュリティのベストプラクティス

Amazon Elastic コンテナサービスのアイデンティティ とアクセス管理

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御するために役立つ AWS のサービス です。IAM 管理者は、誰を認証 (サインイン) し、誰に Amazon ECS リソースの使用を承認する (アクセス許可を付与する) かを制御します。IAM は、追加費用なしで使用できる AWS のサービス です。

トピック

- 対象者
- アイデンティティによる認証
- ポリシーを使用したアクセス権の管理
- IAM を使用するAmazon Elastic Container Service
- Amazon Elastic Container Service のアイデンティティベースのポリシーの例
- AWSAmazon Elastic Container Service に関する 管理ポリシー
- Amazon ECS のサービスリンクロールの使用
- Amazon ECS の IAM ロール
- Amazon ECS コンソールに必要なアクセス許可
- Amazon ECS ブルー/グリーンデプロイの Lambda 関数に必要となるアクセス許可
- Amazon ECS のサービス自動スケーリングに必要な IAM アクセス許可
- リソース作成時にタグ付けするための許可を付与する
- Amazon Elastic Container Service のアイデンティティとアクセスのトラブルシューティング
- Amazon ECS の IAM ベストプラクティス

対象者

AWS Identity and Access Management (IAM) の使用方法は、ロールによって異なります。

- サービスユーザー - 機能にアクセスできない場合は、管理者にアクセス許可をリクエストします (「Amazon Elastic Container Service のアイデンティティとアクセスのトラブルシューティング」を参照)。

- サービス管理者 - ユーザーアクセスを決定し、アクセス許可リクエストを送信します (「IAM を使用するAmazon Elastic Container Service」を参照)
- IAM 管理者 - アクセスを管理するためのポリシーを作成します (「Amazon Elastic Container Service のアイデンティティベースのポリシーの例」を参照)

アイデンティティによる認証

認証とは、アイデンティティ認証情報を使用して AWS にサインインする方法です。ユーザーは、IAM ユーザー の AWS アカウントのルートユーザー として、または IAM ロールを引き受けることによって、認証される 必要があります。

AWS IAM アイデンティティセンター (IAM アイデンティティセンター)、シングルサインオン認証、Google/Facebook 認証情報などの ID ソースからの認証情報を使用して、フェデレーテッドアイデンティティとしてサインインできます。サインインの詳細については、「AWS サインイン ユーザーガイド」の「AWS アカウントにサインインする方法」を参照してください。

プログラムによるアクセスの場合、AWS はリクエストに暗号で署名するための SDK と CLI を提供します。詳細については、「IAM ユーザーガイド」の「API リクエストに対する AWS 署名バージョン 4」を参照してください。

AWS アカウント のルートユーザー

AWS アカウントを作成すると、すべての AWS のサービスとリソースに対する完全なアクセス権を持つ AWS アカウント ルートユーザーと呼ばれる 1 つのサインイン ID を使用して開始します。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザー認証情報を必要とするタスクについては、「IAM ユーザーガイド」の「ルートユーザー認証情報が必要なタスク」を参照してください。

フェデレーテッドアイデンティティ

ベストプラクティスでは、人間のユーザーが一時的な認証情報を使用して AWS のサービス にアクセスする際、アイデンティティプロバイダーとのフェデレーションを使用することが求められます。

フェデレーテッドアイデンティティは、エンタープライズディレクトリ、ウェブ ID プロバイダー、Directory Service のユーザーであり、ID ソースからの認証情報を使用して AWS のサービスにアクセスするユーザーです。フェデレーテッドアイデンティティは、一時的な認証情報を提供するロールを引き受けます。

アクセスを一元管理する場合は、AWS IAM アイデンティティセンター をお勧めします。詳細については、「AWS IAM アイデンティティセンター ユーザーガイド」の「IAM アイデンティティセンターとは」を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、特定の個人やアプリケーションに対する特定のアクセス許可を持つアイデンティティです。長期認証情報を持つ IAM ユーザーの代わりに一時的な認証情報を使用することをお勧めします。詳細は「IAM ユーザーガイド」の「人間のユーザーが一時的な認証情報を使用して AWS にアクセスするには ID プロバイダーとのフェデレーションの使用が必要です」を参照してください。

IAM グループは、IAM ユーザーの集合を指定し、大量のユーザーに対するアクセス許可の管理を容易にします。詳細については、「IAM ユーザーガイド」の「IAM ユーザーに関するユースケース」を参照してください。

IAM ロール

IAM ロールは、特定のアクセス許可を持つアイデンティであり、一時的な認証情報を提供します。ユーザーから IAM ロール (コンソール) に切り替える、または AWS CLI や AWS API オペレーションを呼び出すことで、ロールを引き受けることができます。詳細については、「IAM ユーザーガイド」の「ロールを引き受けるための各種方法」を参照してください。

IAM ロールは、フェデレーションユーザーアクセス、一時的な IAM ユーザーのアクセス許可、クロスアカウントアクセス、クロスサービスアクセス、および Amazon EC2 で実行するアプリケーションに役立ちます。詳細については、IAM ユーザーガイド の IAM でのクロスアカウントリソースアクセス を参照してください。

ポリシーを使用したアクセス権の管理

AWS でアクセスを制御するには、ポリシーを作成して AWS ID またはリソースにアタッチします。ポリシーは、アイデンティティやリソースとの関連付けに伴うアクセス許可を定義します。AWS は、プリンシパルがリクエストを行うときに、これらのポリシーを評価します。大半のポリシーは JSON ドキュメントとして AWS に保存されます。JSON ポリシードキュメントの詳細については、「IAM ユーザーガイド」の「JSON ポリシー概要」を参照してください。

管理者は、ポリシーを使用して、どのプリンシパルがどのリソースに対して、どのような条件でアクションを実行できるかを定義することで、誰が何にアクセスできるかを指定します。

デフォルトでは、ユーザーやロールにアクセス許可はありません。IAM 管理者は IAM ポリシーを作成してロールに追加し、このロールをユーザーが引き受けられるようにします。IAM ポリシーは、オペレーションの実行方法を問わず、アクセス許可を定義します。

アイデンティティベースのポリシー

アイデンティティベースのポリシーは、アイデンティティ (ユーザー、グループ、またはロール) にアタッチできる JSON アクセス許可ポリシードキュメントです。これらのポリシーは、アイデンティティがどのリソースに対してどのような条件下でどのようなアクションを実行できるかを制御します。アイデンティティベースポリシーの作成方法については、IAM ユーザーガイド の カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する を参照してください。

アイデンティティベースのポリシーは、インラインポリシー (単一の ID に直接埋め込む) または管理ポリシー (複数の ID にアタッチされたスタンドアロンポリシー) にすることができます。管理ポリシーとインラインポリシーのいずれかを選択する方法については、「IAM ユーザーガイド」の「管理ポリシーとインラインポリシーのいずれかを選択する」を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。例としては、IAM ロール信頼ポリシーや Amazon S3 バケットポリシーなどがあります。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーで IAM の AWS マネージドポリシーを使用することはできません。

その他のポリシータイプ

AWS は、より一般的なポリシータイプで付与された最大数のアクセス許可を設定できる、追加のポリシータイプをサポートしています。

- アクセス許可の境界 – アイデンティティベースのポリシーで IAM エンティティに付与することのできるアクセス許可の数の上限を設定します。詳細については、「IAM ユーザーガイド」の「IAM エンティティのアクセス許可境界」を参照してください。
- サービスコントロールポリシー (SCP) - AWS Organizations 内の組織または組織単位の最大のアクセス許可を指定します。詳細については、「AWS Organizations ユーザーガイド」の「サービスコントロールポリシー」を参照してください。

- リソースコントロールポリシー (RCP) – は、アカウント内のリソースで利用できる最大数のアクセス許可を定義します。詳細については、「AWS Organizations ユーザーガイド」の「リソースコントロールポリシー (RCP)」を参照してください。
- セッションポリシー – ロールまたはフェデレーションユーザーの一時セッションを作成する際にパラメータとして渡される高度なポリシーです。詳細については、「IAM ユーザーガイド」の「セッションポリシー」を参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成されるアクセス許可を理解するのがさらに難しくなります。複数のポリシータイプが関連するとき、リクエストを許可するかどうかを AWS が決定する方法の詳細については、IAM ユーザーガイド の ポリシーの評価ロジック を参照してください。

IAM を使用するAmazon Elastic Container Service

IAM を使用して Amazon ECS へのアクセスを管理する前に、Amazon ECS で使用できる IAM 機能について理解しておく必要があります。

IAM 機能	Amazon ECS サポート
アイデンティティベースのポリシー	あり
リソースベースのポリシー	なし
ポリシーアクション	あり
ポリシーリソース	部分的
ポリシー条件キー	あり
ACL	なし
ABAC (ポリシー内のタグ)	あり
一時的な認証情報	あり
転送アクセスセッション (FAS)	あり

IAM 機能	Amazon ECS サポート
サービスロール	あり
サービスリンクロール	はい

大部分の IAM 機能が Amazon ECS および、その他のサービスでどのように機能するかに関するおおまかな説明については、AWS IAM ユーザーガイドの「IAM と連携する AWS のサービス」を参照してください。

Amazon ECS のアイデンティティベースの ポリシー

ID ベースのポリシーのサポート: あり

アイデンティティベースポリシーは、IAM ユーザー、ユーザーグループ、ロールなど、アイデンティティにアタッチできる JSON 許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースポリシーの作成方法については、「IAM ユーザーガイド」の「カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する」を参照してください。

IAM アイデンティティベースのポリシーでは、許可または拒否するアクションとリソース、およびアクションを許可または拒否する条件を指定できます。JSON ポリシーで使用できるすべての要素について学ぶには、「IAM ユーザーガイド」の「IAM JSON ポリシーの要素のリファレンス」を参照してください。

Amazon ECS の ID ベースのポリシー例

Amazon ECS でのアイデンティティベースのポリシーの例は、「Amazon Elastic Container Service のアイデンティティベースのポリシーの例」でご確認ください。

Amazon ECS 内のリソースベースのポリシー

リソースベースのポリシーのサポート: なし

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシー や Amazon S3 バケットポリシー があげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスをコントロールできます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポ

リシーによって定義されます。リソースベースのポリシーで、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーションユーザー、または AWS のサービス を含めることができます。

クロスアカウントアクセスを有効にするには、全体のアカウント、または別のアカウントの IAM エンティティを、リソースベースのポリシーのプリンシパルとして指定します。詳細については、IAM ユーザーガイドのIAM でのクロスアカウントリソースアクセスを参照してください。

Amazon ECS のポリシーアクション

ポリシーアクションのサポート: あり

管理者は AWS JSON ポリシーを使用して、だれが何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのようなリソースにどのような条件でアクションを実行できるかということです。

JSON ポリシーの `Action` 要素にはポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。このアクションは関連付けられたオペレーションを実行するためのアクセス許可を付与するポリシーで使用されます。

Amazon ECS アクションのリストを確認するには、「Service Authorization Reference」の「Actions defined by Amazon Elastic Container Service」を参照してください。

Amazon ECS のポリシーアクションは、アクションの前に以下のプレフィックスを使用します。

```
ecs
```

単一のステートメントで複数のアクションを指定するには、アクションをカンマで区切ります。

```
"Action": [
      "ecs:action1",
      "ecs:action2"
         ]
```

ワイルドカード (*) を使用して複数アクションを指定できます。例えば、`Describe` という単語で始まるすべてのアクションを指定するには、次のアクションを含めます:

```
"Action": "ecs:Describe*"
```

Amazon ECS でのアイデンティティベースのポリシーの例は、「Amazon Elastic Container Service のアイデンティティベースのポリシーの例」でご確認ください。

Amazon ECS のポリシーリソース

ポリシーリソースのサポート: 一部

管理者は AWS JSON ポリシーを使用して、だれが何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

`Resource` JSON ポリシー要素はアクションが適用されるオブジェクトを指定します。ベストプラクティスとして、Amazon リソースネーム (ARN) を使用してリソースを指定します。リソースレベルのアクセス許可をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

```
"Resource": "*"
```

Amazon ECS リソースタイプとその ARN のリストを確認するには、「Service Authorization Reference」の「Resources defined by Amazon Elastic Container Service」を参照してください。各リソースの ARN を指定できるアクションについては、「Amazon Elastic Container Service で定義されるアクション」を参照してください。

複数のリソースをサポートする Amazon ECS API アクションもあります。例えば、`DescribeClusters` API アクションを呼び出すときに複数のクラスターを参照できます。複数リソースを単一ステートメントで指定するには、ARN をカンマで区切ります。

```
"Resource": [
      "EXAMPLE-RESOURCE-1",
      "EXAMPLE-RESOURCE-2"
```

例えば、Amazon ECS クラスターリソースの ARN は次のようになります。

```
arn:${Partition}:ecs:${Region}:${Account}:cluster/${clusterName}
```

次の ARN を使用して、ステートメントで `my-cluster-1` および `my-cluster-2` クラスタを指定します。

```
"Resource": [
```

```
        "arn:aws:ecs:us-east-1:123456789012:cluster/my-cluster-1",
        "arn:aws:ecs:us-east-1:123456789012:cluster/my-cluster-2"
```

特定のアカウントに属するすべてのクラスターを指定するには、ワイルドカード (*) を使用します。

```
"Resource": "arn:aws:ecs:us-east-1:123456789012:cluster/*"
```

タスク定義では、最新のリビジョンまたは特定のリビジョンを指定できます。

タスク定義のすべてのリビジョンを指定するには、ワイルドカード (*) を使用します。

```
"Resource:arn:${Partition}:ecs:${Region}:${Account}:task-definition/
${TaskDefinitionFamilyName}:*"
```

特定のタスク定義リビジョンを指定する場合は、${TaskDefinitionRevisionNumber} を使用します。

```
"Resource:arn:${Partition}:ecs:${Region}:${Account}:task-definition/
${TaskDefinitionFamilyName}:${TaskDefinitionRevisionNumber}"
```

Amazon ECS でのアイデンティティベースのポリシーの例は、「Amazon Elastic Container Service のアイデンティティベースのポリシーの例」でご確認ください。

Amazon ECS のポリシー条件キー

サービス固有のポリシー条件キーのサポート: あり

管理者は AWS JSON ポリシーを使用して、だれが何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

Condition 要素は、定義された基準に基づいてステートメントが実行される時期を指定します。イコールや未満などの条件演算子を使用して条件式を作成して、ポリシーの条件とリクエスト内の値を一致させることができます。すべての AWS グローバル条件キーを確認するには、IAM ユーザーガイドのAWS グローバル条件コンテキストキーを参照してください。

Amazon ECS は、以下のサービス固有の条件キーをサポートしており、IAM ポリシーのきめ細かいフィルタリングの提供に使用することができます。

条件キー	説明	評価の種類
aws:RequestTag/${TagKey}	コンテキストキーは "aws:RequestTag/ *tag-key*":"*tag-value* " という形式です。ここで *tag-key* および *tag-value* はタグキーと値のペアです。 タグキーと値のペアが AWS リクエストに含まれていることを確認します。例えば、リクエストに「"Dept"」タグキーが含まれ、「"Accounting" 」という値が含まれているかどうかを確認できます。	String
aws:ResourceTag/${TagKey}	コンテキストキーは "aws:ResourceTag/ *tag-key*":"*tag-value* " という形式です。ここで *tag-key* および *tag-value* はタグキーと値のペアです。 リソース (ユーザーまたはロール) を識別するためにアタッチされたタグが、指定されたキーの名前および値と一致するかどうかをチェックします。	String
aws:TagKeys	このコンテキストキーは "aws:TagKeys":" *tag-key*" という形式であり、ここで *tag-key* は値 (["Dept","Cost-Center"] など) のないタグキーのリストです。 AWS リクエストに存在するタグキーをチェックします。	String
ecs:ResourceTag/${TagKey}	コンテキストキーは "ecs:ResourceTag/ *tag-key*":"*tag-value* " という形式です。ここで *tag-key* および *tag-value* はタグキーと値のペアです。 リソース (ユーザーまたはロール) を識別するためにアタッチされたタグが、指定されたキーの名前および値と一致するかどうかをチェックします。	String
ecs:account-setting	コンテキストキーは "ecs:account-setting":" *account-setting* " という形式で、*account-setting* はアカウント設定の名前です。	String

条件キー	説明	評価の種類
ecs:auto-assign-public-ip	コンテキストキーは"ecs:auto-assign-public-ip":" *value*" 形式であり、*value-* は「true」または「false」です。	String
ecs:capacity-provider	コンテキストキーは "ecs:capacity-provider":" *capacity-provider-arn* " という形式で、*capacity-provider-arn* はキャパシティープロバイダーの ARN です。	ARN、Null
ecs:cluster	コンテキストキーは "ecs:cluster":" *cluster-arn* " という形式です。ここで *cluster-arn* は、Amazon ECS クラスターの ARN です。	ARN、Null
ecs:compute-compatability	コンテキストキーは "ecs:compute-compatability":" *compatability-type* " という形式で、 *compatability-type* は「FARGATE」、「EC2」、または「EXTERNAL」です。	String
ecs:container-instances	コンテキストキーは "ecs:container-instances":" *container-instance-arns* " という形式です。ここで *container-instance-arns* は、1 つ以上のコンテナインスタンス ARN です。	ARN、Null
ecs:container-name	コンテキストキーは "ecs:container-name":" *container-name* " 形式で、*container-instan-* は、タスク定義で定義されている Amazon ECS コンテナの名前です。	String
ecs:enable-execute-command	コンテキストキーは"ecs:enable-execute-command":" *value*" 形式であり、*value-* は「true」または「false」です。	String
ecs:enable-service-connect	コンテキストキーは "ecs:enable-service-connect":" *value*" 形式であり、*value* は "true" または "false" です。	String

条件キー	説明	評価の種類
ecs:enable-ebs-volumes	コンテキストキーは "ecs:enable-ebs-volumes":" *value*" 形式であり、*value* は "true" または "false" です。	String
ecs:enable-managed-tags	コンテキストキーは "ecs:enable-managed-tags":" *value*" 形式であり、*value* は "true" または "false" です。	String
ecs:enable-vpc-lattice	コンテキストキーは "ecs:vpc-lattice":" *value*" 形式であり、*value* は "true" または "false" です。	String
ecs:fargate-ephemeral-storage-kms-key	コンテキストキーは "ecs:fargate-ephemeral-storage-kms-key":" *key*" という形式で、*##*は AWS KMS キーの ID です。	String
ecs:namespace	コンテキストキーは "ecs:namespace":" *namespace-arn* " 形式であり、*namespace-arn* は AWS Cloud Map 名前空間の ARN です。	ARN、Null
ecs:propagate-tags	コンテキストキーは "ecs:propagate-tags":" *value*" という形式で、*#*は "TASK_DEFINITION" 、"SERVICE" 、または "NONE" です。	String
ecs:service	コンテキストキーは "ecs:service":" *service-arn* " という形式です。ここで *service-arn* は、Amazon ECS サービスの ARN です。	ARN、Null
ecs:task-definition	コンテキストキーは "ecs:task-definition":" *task-definition-arn* " という形式です。ここで *task-definition-arn* は、Amazon ECS タスク定義の ARN です。	ARN、Null
ecs:subnet	コンテキストキーは "ecs:subnet":" *subnets*" という形式で、*#####*は awsvpc モードを使用するタスクまたはサービスのサブネットです。	String

条件キー	説明	評価の種類
ecs:task	コンテキストキーは "ecs:task":" *task-arn*" という形式で、*task-arn* は、Amazon ECS サービスの ARN です。	ARN、Null
ecs:task-cpu	コンテキストキーは "ecs:task-cpu":" *task-cpu*" という形式で、*task-cpu* はタスク CPU で、1024 = 1 vCPU の整数です。	整数
ecs:task-memory	コンテキストキーは "ecs:task-memory":" *task-memory* " という形式で、*task-memory* は MiB のタスクメモリです。	整数

Amazon ECS 条件キーのリストを確認するには、「Service Authorization Reference」の「Condition keys for Amazon Elastic Container Service」を参照してください。条件キーを使用できるアクションとリソースについては、「Amazon Elastic Container Service で定義するアクション」を参照してください。

Amazon ECS でのアイデンティティベースのポリシーの例は、「Amazon Elastic Container Service のアイデンティティベースのポリシーの例」でご確認ください。

Amazon ECS アクセスコントロールリスト (ACL)

ACL のサポート: なし

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするためのアクセス許可を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon ECS での属性ベースのアクセスコントロール (ABAC)

> **Important**
>
> Amazon ECS は、すべての Amazon ECS リソースに対して属性ベースのアクセスコントロールをサポートしています。属性を使用してアクションの範囲を設定できるかどうかを判断するには、「サービス許可リファレンス」の「Amazon Elastic Container Service で定義されるアクション」の表を参照してください。まず、リソース列にリソースがあることを確認

します。次に、[条件キー]列を使用して、アクションとリソースの組み合わせのキーを確認します。

ABAC (ポリシー内のタグ) のサポート: あり

属性ベースのアクセス制御 (ABAC) は、タグと呼ばれる属性に基づいてアクセス許可を定義する認可戦略です。IAM エンティティと AWS リソースにタグを付けることで、プリンシパルのタグがリソースタグと一致するときに操作を許可する ABAC ポリシーを設計できます。

タグに基づいてアクセスを管理するには、`aws:ResourceTag/key-name`、`aws:RequestTag/key-name`、または `aws:TagKeys` の条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。

サービスがすべてのリソースタイプに対して 3 つの条件キーすべてをサポートする場合、そのサービスの値はありです。サービスが一部のリソースタイプに対してのみ 3 つの条件キーのすべてをサポートする場合、値は「部分的」になります。

ABAC の詳細については、「IAM ユーザーガイド」の「ABAC 認可でアクセス許可を定義する」を参照してください。ABAC をセットアップする手順を説明するチュートリアルについては、「IAM ユーザーガイド」の「属性ベースのアクセスコントロール (ABAC) を使用する」を参照してください。

Amazon ECS リソースのタグ付けの詳細については、「Amazon ECS リソースにタグ付けする」を参照してください。

リソースのタグに基づいてリソースへのアクセスを制限するためのアイデンティティベースポリシーの例を表示するには、「タグに基づき、Amazon ECS サービスを記述する」を参照してください。

Amazon ECS での一時的な認証情報の使用

一時的な認証情報のサポート: あり

一時的な認証情報は、AWSリソースへの短期的なアクセスを提供し、フェデレーションの使用時またはロールの切り替え時に自動的に作成されます。AWS では、長期的なアクセスキーを使用する代わりに、一時的な認証情報を動的に生成することをお勧めします。詳細については、「IAM ユーザーガイド」の「IAM の一時的な認証情報」および「AWS のサービス と IAM との連携」を参照してください。

Amazon ECS のフォワードアクセスセッション

転送アクセスセッション (FAS) のサポート: あり

転送アクセスセッション (FAS) は、AWS のサービス を呼び出すプリンシパルのアクセス許可を AWS のサービス のリクエストと合わせて使用し、ダウンストリームのサービスに対してリクエストを行います。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。

Amazon ECS のサービスロール

サービスロールのサポート: あり

サービスロールとは、サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、IAM ユーザーガイドの AWS のサービス に許可を委任するロールを作成するを参照してください。

> **Warning**
>
> サービスロールの許可を変更すると、Amazon ECS の機能が破損する可能性があります。Amazon ECS が指示する場合以外は、サービスロールを編集しないでください。

Amazon ECS のサービスにリンクされたロール

サービスリンクロールのサポート: あり

サービスにリンクされたロールは、AWS のサービス にリンクされているサービスロールの一種です。サービスがロールを引き受け、ユーザーに代わってアクションを実行できるようになります。サービスにリンクされたロールは、AWS アカウント に表示され、サービスによって所有されます。IAM 管理者は、サービスリンクロールのアクセス許可を表示できますが、編集することはできません。

Amazon ECS でのサービスにリンクされたロールの作成または管理の詳細については、「Amazon ECS のサービスリンクロールの使用」を参照してください。

Amazon Elastic Container Service のアイデンティティベースのポリシーの例

デフォルトでは、ユーザーおよびロールにはAmazon ECS リソースを作成または変更する許可はありません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。

これらのサンプルの JSON ポリシードキュメントを使用して IAM アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「IAM ポリシーを作成する (コンソール)」を参照してください。

Amazon ECS が定義するアクションとリソースタイプ (リソースタイプごとの ARN の形式を含む) の詳細については、「Service Authorization Reference」の「Actions, resources, and condition keys for Amazon Elastic Container Service」を参照してください。

トピック

- Amazon ECS ポリシーのベストプラクティス
- 自分のアクセス許可の表示を Amazon ECS ユーザーに許可する
- Amazon ECS クラスターの例
- Amazon ECS コンテナインスタンスの例
- Amazon ECS タスク定義の例
- Amazon ECS タスク実行の例
- Amazon ECS タスク開始の例
- Amazon ECS のタスクの例を一覧表示して説明する
- Amazon ECS サービス作成の例
- タグに基づき、Amazon ECS サービスを記述する
- Amazon ECS Service Connect 名前空間の上書き拒否の例

Amazon ECS ポリシーのベストプラクティス

ID ベースのポリシーは、ユーザーのアカウント内で誰かが Amazon ECS リソースを作成、アクセス、または削除できるどうかを決定します。これらのアクションでは、AWS アカウント に費用が発生する場合があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS マネージドポリシーの使用を開始し、最小特権のアクセス許可に移行する – ユーザーとワークロードへのアクセス許可の付与を開始するには、多くの一般的なユースケースのためにアクセス許可を付与する AWS マネージドポリシーを使用します。これらは AWS アカウントで使用できます。ユースケースに固有の AWS カスタマー管理ポリシーを定義して、アクセス許可を絞り込むことをお勧めします。詳細については、IAM ユーザーガイド の AWS マネージドポリシー または ジョブ機能の AWS マネージドポリシー を参照してください。
- 最小特権を適用する – IAM ポリシーでアクセス許可を設定する場合は、タスクの実行に必要な許可のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権アクセス許可とも呼ばれています。IAM を使用して許可を適用する方法の詳細については、IAM ユーザーガイド の IAM でのポリシーとアクセス許可 を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する – ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。たとえば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。また、CloudFormation などの特定の AWS のサービス を介して使用する場合、条件を使用してサービスアクションへのアクセスを許可することもできます。詳細については、IAM ユーザーガイド の IAM JSON ポリシー要素:条件 を参照してください。
- IAM アクセスアナライザー を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する – IAM アクセスアナライザー は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、IAM ユーザーガイド の IAM Access Analyzer でポリシーを検証する を参照してください。
- 多要素認証 (MFA) を要求する – AWS アカウントで IAM ユーザーまたはルートユーザーを要求するシナリオがある場合は、セキュリティを強化するために MFA をオンにします。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、IAM ユーザーガイド の MFA を使用した安全な API アクセス を参照してください。

IAM でのベストプラクティスの詳細については、IAM ユーザーガイド の IAM でのセキュリティのベストプラクティス を参照してください。

自分のアクセス許可の表示を Amazon ECS ユーザーに許可する

この例では、ユーザーアイデンティティにアタッチされたインラインおよびマネージドポリシーの表示を IAM ユーザーに許可するポリシーの作成方法を示します。このポリシーには、コンソールで、

または AWS CLI か AWS API を使用してプログラム的に、このアクションを完了するアクセス許可が含まれています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon ECS クラスターの例

次の IAM ポリシーでは、クラスターを作成し、記載したアクセス権限を付与します。`CreateCluster` と `ListClusters` のアクションはリソースを受け入れないため、すべてのリソースでリソース定義は * に設定されます。

JSON

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:CreateCluster",
                "ecs:ListClusters"
            ],
            "Resource": "*"
        }
    ]
}
```

JSON

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:DescribeClusters",
                "ecs:DeleteCluster"
            ],
            "Resource": ["arn:aws:ecs:us-east-1:123456789012:cluster/cluster-
name"]
        }
    ]
}
```

次の IAM ポリシーでは、特定のクラスターに記述、および削除するアクセス権限を付与します。DescribeClusters と DeleteCluster のアクションはリソースとしてクラスター ARN を使用します。

JSON

```
{
```

```
    "Statement": [
        {
            "Action": [
                "ecs:Describe*",
                "ecs:List*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "ecs:DeleteCluster",
                "ecs:DeregisterContainerInstance",
                "ecs:ListContainerInstances",
                "ecs:RegisterContainerInstance",
                "ecs:SubmitContainerStateChange",
                "ecs:SubmitTaskStateChange"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:ecs:us-east-1:123456789012:cluster/default"
        },
        {
            "Action": [
                "ecs:DescribeContainerInstances",
                "ecs:DescribeTasks",
                "ecs:ListTasks",
                "ecs:UpdateContainerAgent",
                "ecs:StartTask",
                "ecs:StopTask",
                "ecs:RunTask"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Condition": {
                "ArnEquals": {"ecs:cluster": "arn:aws:ecs:us-
east-1:123456789012:cluster/default"}
            }
        }
    ]
}
```

次の IAM ポリシーは、ユーザーまたはグループが特定のクラスターでのオペレーションの実行のみを許可するユーザーまたはグループにアタッチできます。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "ecs:Describe*",
                "ecs:List*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "ecs:DeleteCluster",
                "ecs:DeregisterContainerInstance",
                "ecs:ListContainerInstances",
                "ecs:RegisterContainerInstance",
                "ecs:SubmitContainerStateChange",
                "ecs:SubmitTaskStateChange"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:ecs:us-east-1:111122223333:cluster/default"
        },
        {
            "Action": [
                "ecs:DescribeContainerInstances",
                "ecs:DescribeTasks",
                "ecs:ListTasks",
                "ecs:UpdateContainerAgent",
                "ecs:StartTask",
                "ecs:StopTask",
                "ecs:RunTask"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Condition": {
                "ArnEquals": {"ecs:cluster": "arn:aws:ecs:us-
east-1:111122223333:cluster/default"}
```

```
            }
        }
    ]
}
```

Amazon ECS コンテナインスタンスの例

コンテナインスタンス登録は、Amazon ECS エージェントによって処理されますが、ユーザーがクラスターからインスタンスを手動で登録解除を許可する場合があります。コンテナインスタンスを間違ったクラスターに登録、または実行中のタスクがあるインスタンスを誤って終了してしまうかもしれません。

次に示す IAM ポリシーでは、ユーザーが指定したクラスターでコンテナインスタンスをリストし、登録を解除することができます。

JSON

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:DeregisterContainerInstance",
                "ecs:ListContainerInstances"
            ],
            "Resource": "arn:aws:ecs:us-east-1:123456789012:cluster/cluster_name"
        }
    ]
}
```

Amazon ECS タスク定義の例

タスク定義 IAM ポリシーは、リソースレベルのアクセス権限をサポートしていませんが、次の IAM ポリシーでは、ユーザーがタスク定義を登録、一覧表示、および記述することができます。

コンソールを使用する場合は、`CloudFormation: CreateStack` を `Action` として追加する必要があります。

JSON

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:RegisterTaskDefinition",
                "ecs:ListTaskDefinitions",
                "ecs:DescribeTaskDefinition"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon ECS タスク実行の例

RunTask のリソースは、タスク定義です。ユーザーがタスク定義を実行できるクラスターを制限するには、Condition ブロックで指定します。この方法には、適切なアクセス権を許可するためにリソースでタスク定義とクラスターの両方をリストする必要がないという利点があります。いずれか、または両方を適用できます。

次の IAM ポリシーでは、特定のクラスターで特定のタスク定義の変更を実行するアクセス権限を付与します。

JSON

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["ecs:RunTask"],
            "Condition": {
                "ArnEquals": {"ecs:cluster": "arn:aws:ecs:us-
east-1:123456789012:cluster/cluster_name"}
            },
            "Resource": ["arn:aws:ecs:us-east-1:123456789012:task-definition/
task_family:*"]
        }
```

```
    ]
}
```

Amazon ECS タスク開始の例

StartTask のリソースは、タスク定義です。ユーザーがタスク定義を開始できるクラスターとコンテナインスタンスを制限するには、Condition ブロックでそれらを指定します。この方法には、適切なアクセス権を許可するためにリソースでタスク定義とクラスターの両方をリストする必要がないという利点があります。いずれか、または両方を適用できます。

次の IAM ポリシーでは、特定のクラスターおよび特定のコンテナインスタンスで特定のタスク定義の変更を開始するアクセス許可を付与します。

> **Note**
>
> この例では、StartTask または別の AWS CLI SDK で AWS API を呼び出すときに、Resource マッピングが一致するようにタスク定義リビジョンを指定する必要があります。

JSON

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["ecs:StartTask"],
            "Condition": {
                "ArnEquals": {
                    "ecs:cluster": "arn:aws:ecs:us-
east-1:123456789012:cluster/cluster_name",
                    "ecs:container-instances": ["arn:aws:ecs:us-
east-1:123456789012:container-instance/cluster_name/container_instance_UUID"]
                }
            },
            "Resource": ["arn:aws:ecs:us-east-1:123456789012:task-
definition/task_family:*"]
        }
    ]
```

```
}
```

Amazon ECS のタスクの例を一覧表示して説明する

次の IAM ポリシーでは、ユーザーが指定したクラスターに指定タスクを記述することができます。

JSON

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["ecs:DescribeTasks"],
            "Condition": {
                "ArnEquals": {"ecs:cluster": "arn:aws:ecs:us-
east-1:123456789012:cluster/cluster_name"}
            },
            "Resource": ["arn:aws:ecs:us-
east-1:123456789012:task/cluster_name/task_UUID"]
        }
    ]
}
```

Amazon ECS サービス作成の例

次の IAM; ポリシーでは、ユーザーが AWS マネジメントコンソール で Amazon ECS サービスを作成することができます。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "application-autoscaling:Describe*",
                "application-autoscaling:PutScalingPolicy",
                "application-autoscaling:RegisterScalableTarget",
```

```
                "cloudwatch:DescribeAlarms",
                "cloudwatch:PutMetricAlarm",
                "ecs:List*",
                "ecs:Describe*",
                "ecs:CreateService",
                "elasticloadbalancing:Describe*",
                "iam:GetPolicy",
                "iam:GetPolicyVersion",
                "iam:GetRole",
                "iam:ListAttachedRolePolicies",
                "iam:ListRoles",
                "iam:ListGroups",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

タグに基づき、Amazon ECS サービスを記述する

アイデンティティベースのポリシーの条件を使用して、タグに基づいて Amazon ECS リソースへのアクセスをコントロールできます。この例では、サービスを表示できるポリシーを作成する方法を示します。ただし、アクセス許可は、サービスタグ `Owner` にそのユーザーのユーザー名の値がある場合のみ、付与されます。このポリシーでは、このアクションをコンソールで実行するために必要なアクセス許可も付与します。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DescribeServices",
            "Effect": "Allow",
            "Action": "ecs:DescribeServices",
            "Resource": "*"
        },
        {
            "Sid": "ViewServiceIfOwner",
            "Effect": "Allow",
```

```
            "Action": "ecs:DescribeServices",
            "Resource": "arn:aws:ecs:*:*:service/*",
            "Condition": {
                "StringEquals": {"ecs:ResourceTag/Owner": "${aws:username}"}
            }
        }
    ]
}
```

このポリシーをアカウントの IAM ユーザーにアタッチできます。richard-roe という名前のユーザーが Amazon ECS サービスを表示する場合は、サービスに Owner=richard-roe または owner=richard-roe とタグ付けする必要があります。それ以外の場合、アクセスは拒否されます。条件キー名では大文字と小文字が区別されないため、条件タグキー Owner は Owner と owner の両方に一致します。詳細については、「IAM ユーザーガイド」の「IAM JSON ポリシー要素：条件」を参照してください。

Amazon ECS Service Connect 名前空間の上書き拒否の例

以下の IAM ポリシーは、サービス設定のデフォルトの Service Connect 名前空間をユーザーが上書きすることを拒否します。デフォルトの名前空間はクラスターに設定されます。ただし、サービス設定内ではそれを上書きできます。一貫性を保つため、新しいサービスはすべて同じ名前空間を使用するように設定することを検討してください。サービスに特定の名前空間を使用するように要求するには、以下のコンテキストキーを使用します。以下の例で、<region>、<aws_account_id>、<cluster_name>、<namespace_id> を独自のものに置き換えます。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:CreateService",
                "ecs:UpdateService"
            ],
            "Condition": {
                "ArnEquals": {
```

```
                    "ecs:cluster": "arn:aws:ecs:us-
east-1:123456789012:cluster/cluster_name",
                    "ecs:namespace": "arn:aws:servicediscovery:us-
east-1:123456789012:namespace/namespace_id"
                }
            },
            "Resource": "*"
        }
    ]
}
```

AWSAmazon Elastic Container Service に関する 管理ポリシー

ユーザー、グループ、ロールにアクセス許可を追加するには自分でポリシーを作成するよりも、AWS マネージドポリシーを使用する方が簡単です。チームに必要な権限のみを提供する IAM カスタマーマネージドポリシーを作成する には時間と専門知識が必要です。すぐに使用を開始するために、AWS マネージドポリシーを使用できます。これらのポリシーは一般的なユースケースを対象範囲に含めており、AWS アカウントで利用できます。AWS マネージドポリシーの詳細については「IAM ユーザーガイド」の「AWS マネージドポリシー」を参照してください。

AWS のサービスはAWS マネージドポリシーを維持および更新します。AWS マネージドポリシーの許可を変更することはできません。サービスでは新しい機能を利用できるようにするために、AWS マネージドポリシーに権限が追加されることがあります。この種類の更新はポリシーがアタッチされている、すべてのアイデンティティ (ユーザー、グループおよびロール) に影響を与えます。新しい機能が立ち上げられた場合や、新しいオペレーションが使用可能になった場合に、各サービスが AWS マネージドポリシーを更新する可能性が最も高くなります。サービスは AWS マネージドポリシーから権限を削除しないため、ポリシーの更新によって既存の権限が破棄されることはありません。

さらに、AWS は複数のサービスにまたがるジョブ機能の特徴に対するマネージドポリシーもサポートしています。例えば、ReadOnlyAccess AWS マネージドポリシーではすべての AWS のサービスおよびリソースへの読み取り専用アクセスを許可します。サービスが新しい機能を起動する場合、AWS は新たなオペレーションとリソース用に、読み取り専用の許可を追加します。ジョブ機能のポリシーの一覧および詳細については、「IAM ユーザーガイド」の「AWS のジョブ機能のマネージドポリシー」を参照してください。

Amazon ECS および Amazon ECR では、ユーザー、グループ、ロール、Amazon EC2 インスタンス、および Amazon ECS タスクにアタッチして、リソースや API オペレーションで異なる制御レベルを使用できる複数の管理ポリシーと信頼関係を提供しています。これらのポリシーを直接適用することも、独自のポリシーを作成する開始点として使用することもできます。Amazon ECR 管理ポリシーの詳細については、「Amazon ECR 管理ポリシー」を参照してください。

AmazonECS_FullAccess

`AmazonECS_FullAccess` ポリシーを IAM IDにアタッチできます。このポリシーは、Amazon ECS リソースへの管理アクセスを許可し、IAM ID(ユーザー、グループ、ロールなど)にAWSサービスは、Amazon ECS のすべての機能を使用するために統合されています。このポリシーを使用すると、AWS マネジメントコンソール で利用可能な Amazon ECS のすべての機能にアクセスできます。これらの機能は、。

このポリシーのアクセス許可を確認するには、「AWS マネージドポリシーリファレンス」の「AmazonECS_FullAccess」を参照してください。

AmazonECSInfrastructureRolePolicyForVolumes

`AmazonECSInfrastructureRolePolicyForVolumes` マネージドポリシーを IAM エンティティにアタッチできます。

ポリシーは、Amazon ECS がユーザーに変わって AWS API コールを行うのに必要なアクセス許可を付与します。このポリシーは、Amazon ECS のタスクとサービスを起動するときにボリューム設定で指定する IAM ロールにアタッチできます。このロールにより、Amazon ECS はタスクにアタッチされたボリュームを管理できます。詳細については、「Amazon ECS インフラストラクチャの IAM ロール」を参照してください。

このポリシーのアクセス許可を確認するには、「AWS マネージドポリシーリファレンス」の「AmazonECSInfrastructureRolePolicyForVolumes」を参照してください。

AmazonEC2ContainerServiceforEC2Role

`AmazonEC2ContainerServiceforEC2Role` ポリシーを IAM IDにアタッチできます。このポリシーは、ユーザーに代わって AWS に呼び出すことを Amazon ECS コンテナインスタンスに許可する管理権限を付与します。詳細については、「Amazon ECS コンテナインスタンスの IAM ロール」を参照してください。

Amazon ECS は、Amazon EC2 インスタンスまたは外部インスタンスに対して、ユーザーに代わってアクションを実行することを Amazon ECS に許可するサービスロールにこのポリシーをアタッチします。

このポリシーのアクセス許可を確認するには、「AWS マネージドポリシーリファレンス」の「AmazonEC2ContainerServiceforEC2Role」を参照してください。

考慮事項

AmazonEC2ContainerServiceforEC2Role が管理する IAM ポリシーを使用するときは、次の推奨事項と考慮事項を検討する必要があります。

- 最小権限を付与する標準のセキュリティアドバイスに従って、AmazonEC2ContainerServiceforEC2Role 管理ポリシーを変更して、特定のニーズに合わせることができます。管理ポリシーで付与されたアクセス許可のいずれかがユースケースに必要でない場合、カスタムポリシーを作成し、必要なアクセス許可のみを追加します。例えば、UpdateContainerInstancesState アクセス許可は、スポットインスタンスのドレインに提供されます。その権限がユースケースに必要ない場合、カスタムポリシーを使用して除外します。
- コンテナインスタンスで実行しているコンテナは、インスタンスのメタデータを通じてコンテナインスタンスのロールに提供されているすべての権限にアクセスできます。コンテナインスタンスのロールのアクセス許可は、以下に提供されるマネージド型 AmazonEC2ContainerServiceforEC2Role ポリシーのアクセス許可のミニマリストに制限することをお勧めします。タスクのコンテナでリストされていない追加のアクセス許可が必要な場合は、独自の IAM ロールを使用してタスクを提供することをお勧めします。詳細については、「Amazon ECS タスクの IAM ロール」を参照してください。

  コンテナを防ぐには、docker0 ブリッジからコンテナインスタンスロールに指定されたアクセス許可にアクセスしないようにします。これは、次の iptables コマンドをコンテナインスタンスで実行することで Amazon ECS タスクの IAM ロール が提供するアクセス許可を許可して、コンテナは、有効なこのルールでインスタンスメタデータをクエリできません。このコマンドはデフォルトの Docker のブリッジ設定を前提としており、host ネットワークモードを使用してコンテナでは動作しません。詳細については、「ネットワークモード」を参照してください。

  ```
  sudo yum install -y iptables-services; sudo iptables --insert DOCKER USER 1 --in-interface docker+ --destination 169.254.169.254/32 --jump DROP
  ```

  再起動後も有効にするには、コンテナインスタンスでこの iptables ルールを保存する必要があります。Amazon ECS 最適化 AMI の場合は、次のコマンドを使用します。他のオペレーティングシステムについては、その OS のドキュメントを参照してください。

  - Amazon ECS に最適化された Amazon Linux 2 AMI の場合:

```
sudo iptables-save | sudo tee /etc/sysconfig/iptables && sudo systemctl enable --
now iptables
```

- Amazon ECS に最適化された Amazon Linux AMI の場合:

```
sudo service iptables save
```

AmazonEC2ContainerServiceEventsRole

`AmazonEC2ContainerServiceEventsRole` ポリシーを IAM IDにアタッチできます。このポリシーは、Amazon EventBridge(旧 CloudWatch Events)がユーザーに代わってタスクを実行できるようにするアクセス権限を付与します。このポリシーは、スケジュールされたタスクの作成時に指定された IAM ロールにアタッチできます。詳細については、「Amazon ECS EventBridge IAM ロール」を参照してください。

このポリシーのアクセス許可を確認するには、「AWS マネージドポリシーリファレンス」の「AmazonEC2ContainerServiceEventsRole」を参照してください。

AmazonECSTaskExecutionRolePolicy

`AmazonECSTaskExecutionRolePolicy` 管理 IAM ポリシーは、Amazon ECS コンテナエージェントおよび AWS Fargate コンテナエージェントに必要なアクセス権限を付与し、ユーザーに代わって AWS API コールを作成します。このポリシーは、タスク実行 IAM ロールに追加できます。詳細については、「Amazon ECS タスク実行IAM ロール」を参照してください。

このポリシーのアクセス許可を確認するには、「AWS マネージドポリシーリファレンス」の「AmazonECSTaskExecutionRolePolicy」を参照してください。

AmazonECSServiceRolePolicy

`AmazonECSServiceRolePolicy` マネージド IAM ポリシーにより、Amazon Elastic Container Service でクラスターを管理できるようになります。このポリシーは、AWSServiceRoleForECS サービスにリンクされたロールに追加できます。

このポリシーのアクセス許可を確認するには、「AWS マネージドポリシーリファレンス」の「AmazonECSServiceRolePolicy」を参照してください。

AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecur

IAM エンティティに AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecurity ポリシーをアタッチできます。このポリシーは、お客様に代わって Amazon ECS Service Connect TLS 機能を管理するために必要な AWS Private Certificate Authority、Secrets Manager、およびその他の AWS サービスへの管理アクセスを提供します。

このポリシーのアクセス許可を確認するには、「AWS マネージドポリシーリファレンス」の「AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecurity」を参照してください。

AWSApplicationAutoscalingECSServicePolicy

IAM エンティティに AWSApplicationAutoscalingECSServicePolicy をアタッチすることはできません。このポリシーは、ユーザーに代わって Application Auto Scaling がアクションを実行することを許可する、サービスにリンクされたロールにアタッチされます。詳細については、「Application Auto Scaling のサービスにリンクされたロール」を参照してください。

このポリシーのアクセス許可を確認するには、「AWS マネージドポリシーリファレンス」の「AWSApplicationAutoscalingECSServicePolicy」を参照してください。

AWSCodeDeployRoleForECS

IAM エンティティに AWSCodeDeployRoleForECS をアタッチすることはできません。このポリシーは、ユーザーに代わって CodeDeploy がアクションを実行することを許可する、サービスにリンクされたロールにアタッチされます。詳細については、AWS CodeDeploy ユーザーガイドの「CodeDeploy のサービスロールを作成する」を参照してください。

このポリシーのアクセス許可を確認するには、「AWS マネージドポリシーリファレンス」の「AWSCodeDeployRoleForECS」を参照してください。

AWSCodeDeployRoleForECSLimited

IAM エンティティに AWSCodeDeployRoleForECSLimited をアタッチすることはできません。このポリシーは、ユーザーに代わって CodeDeploy がアクションを実行することを許可する、サービスにリンクされたロールにアタッチされます。詳細については、AWS CodeDeploy ユーザーガイドの「CodeDeploy のサービスロールを作成する」を参照してください。

このポリシーのアクセス許可を確認するには、「AWS マネージドポリシーリファレンス」の「AWSCodeDeployRoleForECSLimited」を参照してください。

AmazonECSInfrastructureRolePolicyForLoadBalancers

IAM エンティティに AmazonECSInfrastructureRolePolicyForLoadBalancers ポリシーをアタッチできます。このポリシーは、Amazon ECS がユーザーに代わって Elastic Load Balancing リソースを管理できるようにするアクセス許可を付与します。このポリシーには以下が含まれます。

- リスナー、ルール、ターゲットグループ、ターゲットヘルスを記述するための読み取り専用アクセス許可
- ターゲットグループでターゲットを登録および登録解除するためのアクセス許可
- Application Load Balancer と Network Load Balancer のリスナーを変更するためのアクセス許可
- Application Load Balancer のルールを変更するためのアクセス許可

これらのアクセス許可により、Amazon ECS はサービスの作成時または更新時にロードバランサーの設定を自動的に管理できるようになり、コンテナへのトラフィックの適切なルーティングが確保されます。

このポリシーのアクセス許可を確認するには、「AWS マネージドポリシーリファレンス」の「AmazonECSInfrastructureRolePolicyForLoadBalancers」を参照してください。

AmazonECSInfrastructureRolePolicyForManagedInstances

IAM エンティティに AmazonECSInfrastructureRolePolicyForManagedInstances ポリシーをアタッチできます。このポリシーは、ユーザーに代わって ECS マネージドインスタンスの Amazon EC2 リソースを作成および更新するために Amazon ECS が必要とするアクセス許可を付与します。このポリシーには以下が含まれます。

- マネージドインスタンスの Amazon EC2 起動テンプレートを作成および管理するアクセス許可
- CreateFleet と RunInstances を使用して Amazon EC2 インスタンスをプロビジョニングするアクセス許可
- ECS が作成する Amazon EC2 リソースでタグを作成および管理するためのアクセス許可
- マネージドインスタンスの Amazon EC2 インスタンスに IAM ロールを渡すアクセス許可
- Amazon EC2 スポットインスタンス用のサービスにリンクされたロールを作成するアクセス許可
- インスタンス、インスタンスタイプ、起動テンプレート、ネットワークインターフェイス、アベイラビリティーゾーン、セキュリティグループ、サブネット、VPC、EC2 イメージ、およびキャパシティ予約を含めた Amazon EC2 リソースを記述するための読み取り専用許可

- Amazon ResourceGroups リソースを一覧表示するための読み取り専用許可 (タグ付けされたリソースを取得し、Amazon CloudFormation スタックリソースを一覧表示するための基礎的な許可が必要です)

これらのアクセス許可により、Amazon ECS は ECS マネージドインスタンスの Amazon EC2 インスタンスを自動的にプロビジョニングおよび管理し、基盤となるコンピューティングリソースの適切な設定とライフサイクル管理を確保できます。

このポリシーのアクセス許可を確認するには、「AWS マネージドポリシーリファレンス」の「AmazonECSInfrastructureRolePolicyForManagedInstances」を参照してください。

AmazonECSInfrastructureRolePolicyForVpcLattice

IAM エンティティに AmazonECSInfrastructureRolePolicyForVpcLattice ポリシーをアタッチできます。このポリシーは、ユーザーに代わって Amazon ECS ワークロードの VPC Lattice 機能を管理するために必要な他の AWS サービスリソースへのアクセスを提供します。

このポリシーのアクセス許可を確認するには、「AWS マネージドポリシーリファレンス」の「AmazonECSInfrastructureRolePolicyForVpcLattice」を参照してください。

ユーザーに代わって Amazon ECS ワークロードの VPC Lattice 機能を管理するために必要な他の AWS サービスリソースへのアクセスを提供します。

AmazonECSInfrastructureRoleforExpressGatewayServices

IAM エンティティに AmazonECSInfrastructureRoleforExpressGatewayServices ポリシーをアタッチできます。このポリシーは、Amazon ECS がユーザーに代わって Express サービスを使用してウェブアプリケーションを作成および更新するために必要なアクセス許可を付与します。このポリシーには以下が含まれます。

- Amazon ECS Application Auto Scaling のサービスにリンクされたロールを作成するアクセス許可
- これには Application Load Blancer、リスナー、ルール、ターゲットグループの作成、更新、削除のアクセス許可が含まれます。
- ECS マネージドリソースの VPC セキュリティグループを作成、変更、削除するアクセス許可
- ACM 経由で SSL/TLS 証明書のリクエスト、管理、削除を実行するアクセス許可
- Amazon ECS サービスの Application Auto Scaling ポリシーとターゲットを設定するアクセス許可
- 自動スケーリングトリガー用 CloudWatch アラームを作成および管理するためのアクセス許可

- ロードバランサー、VPC リソース、証明書、自動スケーリング設定、CloudWatch アラームを記述するための読み取り専用アクセス許可

これらのアクセス許可により、Amazon ECS は Express サービスウェブアプリケーションに必要なインフラストラクチャコンポーネント (ロードバランシング、セキュリティグループ、SSL 証明書、自動スケーリング設定など) のプロビジョニングおよび管理を自動的に実行できます。

このポリシーのアクセス許可を確認するには、「AWS マネージドポリシーリファレンス」の「AmazonECSInfrastructureRoleforExpressGatewayServices」を参照してください。

AmazonECSComputeServiceRolePolicy

AmazonECSComputeServiceRole サービスにリンクされたロールが AmazonECSComputeServiceRolePolicy ポリシーにアタッチされます。詳細については、「ロールを使用して Amazon ECS マネージドインスタンスを管理する」を参照してください。

このポリシーには Amazon ECS に次のタスクを完了させるための権限が含まれています。

- Amazon ECS が起動テンプレートを記述および削除する。
- Amazon ECS が起動テンプレートのバージョンを記述および削除する。
- Amazon ECS がインスタンスを終了する。
- Amazon ECS では、次のインスタンスデータパラメータを記述できます。
  - インスタンス
  - インスタンスネットワークインターフェイス: Amazon ECS は、EC2 インスタンスのライフサイクルを管理するインターフェイスを記述できます。
  - インスタンスイベントウィンドウ: Amazon ECS は、インスタンスのパッチ適用を目的としたワークフロー中断の可否を決定するためのイベントウィンドウ情報を記述できます。
  - インスタンスステータス: Amazon ECS は、インスタンスの状態をモニタリングするためにインスタンスステータスを記述できます。

このポリシーのアクセス許可を確認するには、「AWS マネージドポリシーリファレンス」の「AmazonECSComputeServiceRolePolicy」を参照してください。

AmazonECSInstanceRolePolicyForManagedInstances

この AmazonECSInstanceRolePolicyForManagedInstances ポリシーは、Amazon ECS マネージドインスタンスを Amazon ECS クラスターに登録し、Amazon ECS サービスと通信するためのアクセス許可を提供します。

このポリシーには Amazon ECS マネージドインスタンスに次のタスクを完了させるための権限が含まれます。

- Amazon ECS クラスターの登録および登録解除を実行する。
- コンテナインスタンスの状態の変更を送信する。
- タスクの状態の変更を送信する。
- Amazon ECS エージェントのポーリングエンドポイントを検出する。

このポリシーのアクセス許可を確認するには、「AWS マネージドポリシーリファレンス」の「AmazonECSInstanceRolePolicyForManagedInstances」を参照してください。

Amazon ECS での AWS 管理ポリシーに関する更新

Amazon ECS 向けの AWS 管理ポリシーについて、このサービスがこれらの変更の追跡を開始して以降行われた更新の詳細情報を示します。このページの変更に関する自動通知を入手するには、Amazon ECS ドキュメントの履歴ページから、RSS フィードにサブスクライブしてください。

変更	説明	日付
AmazonECS_FullAccess にアクセス許可を追加します。	AmazonECS_FullAccess マネージド IAM ポリシーが更新され、cloudformation:TagResource および cloudformation:UntagResource アクセス許可が含まれるようになりました。これらのアクセス許可により、スタックの作成時または更新時に CloudFormation スタックにタグを追加できます。	2026 年 9 月 2 日

変更	説明	日付
AmazonECSServiceRolePolicy に許可を追加	AmazonECSServiceRolePolicy マネージド IAM ポリシーが更新され、autoscaling:TerminateInstanceInAutoScalingGroup アクセス許可が含まれるようになりました。このアクセス許可により、Amazon ECS は Amazon ECS クラスターへの登録に失敗したカスタマー ASG の Amazon EC2 インスタンスを終了できます。	2026 年 4 月 24 日

変更	説明	日付
AmazonECSInfrastructureRolePolicyForManagedInstances ポリシーを更新	Amazon ECS マネージドインスタンスでキャパシティ予約をサポートするため、以下の許可で AmazonECSInfrastructureRolePolicyForManagedInstances ポリシーが更新されました。 • Amazon ECS がキャパシティ予約のグループを取得できるようにする resource-groups:ListGroupResources 許可が追加されました。このコマンドを実行するには、cloudformation:ListStackResources 許可、cloudformation:DescribeStacks 許可、および tag:GetResources 許可も必要です。 • Amazon ECS が Amazon ECS マネージドインスタンスとの統合に必要なキャパシティ予約の詳細情報を収集できるようにする ec2:DescribeCapacityReservations 許可が追加されました。 • Amazon ECS が対象の Amazon EC2 キャパシ	2026 年 2 月 24 日

変更	説明	日付
	ティ予約をプロビジョニングするための Amazon ResourceGroups グループリソースを含めるように `ec2:RunInstances` 許可が変更されました。	
AmazonECSServiceRolePolicy に許可を追加	`AmazonECSServiceRolePolicy` マネージド IAM ポリシーが更新され、`ssmmessages:OpenDataChannel` アクセス許可が含まれるようになりました。この許可は、Amazon ECS が ECS Exec セッションのデータチャネルを開くことを可能にします。	2026 年 1 月 20 日

変更	説明	日付
AmazonECSInfrastructureRolePolicyForManagedInstances ポリシーを更新	AmazonECSInfrastructureRolePolicyForManagedInstances ポリシーが更新され、CreateFleet 許可が変更されました。以下の理由により、サブネット、セキュリティグループ、EC2 イメージに対するリソースベースの条件が削除されました。 • サブネットとセキュリティグループは Amazon ECS が作成または管理するものではないため、必須の AmazonECSManaged タグが含まれていない。 • Amazon ECS が EC2 イメージを所有および管理していても、タグがお客様に表示されないため、リソースベースのポリシーが適用されない。 この変更によって、期待される ECS 管理タグがないリソースでもポリシーが正しく機能するようになります。	2025 年 12 月 15 日

変更	説明	日付
AmazonECSServiceRolePolicy に許可を追加	AmazonECSServiceRolePolicy マネージド IAM ポリシーを更新し、新しい Amazon EC2 アクセス許可を追加しました。このアクセス許可で、Amazon ECS はイベントウィンドウに関連付けられたサービスおよびクラスターの Amazon EC2 イベントウィンドウを取得できます。	2025 年 11 月 20 日
AmazonECSServiceRolePolicy に許可を追加	AmazonECSServiceRolePolicy マネージド IAM ポリシーを更新し、新しい Amazon EC2 アクセス許可を追加しました。このアクセス許可で、Amazon ECS はタスク ENI のプロビジョニングおよびプロビジョニング解除を実行できます。	2025 年 11 月 14 日

変更	説明	日付
AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecurity ポリシーを更新	AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecurity マネージド IAM ポリシーを更新し、secretsmanager:DescribeSecret アクセス許可を独自のポリシーステートメントとして分離しました。このアクセス許可は、引き続き Amazon ECS で作成したシークレットにのみ Amazon ECS アクセスの範囲を設定します。また、範囲設定にはリソースタグの代わりに ARN パターンマッチングを使用します。これにより、Amazon ECS は、シークレットが削除された場合などを含めて、ライフサイクル全体でシークレットの状態をモニタリングできます。	2025 年 11 月 13 日
新しい the section called “AmazonECSInfrastructureRoleforExpressGatewayServices” の追加	Express サービスを使用してウェブアプリケーションを作成および管理するための Amazon ECS アクセス許可を提供する、新しい AmazonECSInfrastructureRoleforExpressGatewayServices ポリシーを追加しました。	2025 年 11 月 21 日

変更	説明	日付
新しい the section called “AmazonECSInstanceRolePolicyForManagedInstances” の追加	Amazon ECS マネージドインスタンスから Amazon ECS クラスターに登録するためのアクセス許可を提供する、新しい AmazonECSInstanceRolePolicyForManagedInstances ポリシーを追加しました。	2025 年 9 月 30 日
新しい the section called “AmazonECSInfrastructureRolePolicyForManagedInstances” の追加	Amazon EC2 マネージドリソースを作成および管理するための Amazon ECS アクセスを提供する新しい AmazonECSInfrastructureRolePolicyForManagedInstances ポリシーを追加しました。	2025 年 9 月 30 日
新しい AmazonECSComputeServiceRolePolicy を追加	Amazon ECS が Amazon ECS マネージドインスタンスと関連リソースを管理できるようにします。	2025 年 8 月 31 日

変更	説明	日付
the section called “AmazonEC2ContainerServiceforEC2Role” へのアクセス許可を追加する	AmazonEC2ContainerServiceforEC2Role マネージド IAM ポリシーが更新され、ecs:ListTagsForResource アクセス許可が含まれるようになりました。このアクセス許可により、Amazon ECS エージェントはタスクメタデータエンドポイント (${ECS_CONTAINER_METADATA_URI_V4}/taskWithTags) を介してタスクおよびコンテナインスタンスのタグを取得できます。	2025 年 8 月 4 日
AmazonECSInfrastructureRolePolicyForLoadBalancers にアクセス許可を追加します。	AmazonECSInfrastructureRolePolicyForLoadBalancers マネージド IAM ポリシーが更新され、ターゲットグループを記述、登録解除、登録するための新しいアクセス許可が追加されました。	2025 年 7 月 25 日
新しい the section called “AmazonECSInfrastructureRolePolicyForLoadBalancers” ポリシーを追加する	Amazon ECS ワークロードに関連づけられているロードバランサーの管理に必要な他の AWS サービスリソースへのアクセスを提供する新しい AmazonECSInfrastructureRolePolicyForLoadBalancers ポリシーを追加しました。	2025 年 7 月 15 日

変更	説明	日付
AmazonECSServiceRolePolicy にアクセス許可を追加します。	AmazonECSServiceRolePolicy マネージド IAM ポリシーは、Amazon ECS が管理するサービスの AWS Cloud Map サービス属性を Amazon ECS が更新できる新しい AWS Cloud Map アクセス許可で更新されました。	2025 年 7 月 15 日
AmazonECSServiceRolePolicy にアクセス許可を追加する	AmazonECSServiceRolePolicy マネージド IAM ポリシーは、Amazon ECS が管理するサービスの AWS Cloud Map サービス属性を Amazon ECS が更新できる新しい AWS Cloud Map アクセス許可で更新されました。	2025 年 6 月 24 日
AmazonECSInfrastructureRolePolicyForVolumes へのアクセス許可を追加する	AmazonECSInfrastructureRolePolicyForVolumes ポリシーが更新され、ec2:DescribeInstances のアクセス許可が追加されました。アクセス許可は、同じコンテナインスタンスで実行される Amazon ECS タスクにアタッチされている Amazon EBS ボリュームのデバイス名のコリジョンを防ぐのに役立ちます。	2025 年 6 月 2 日

変更	説明	日付
新しい AmazonECSInfrastructureRolePolicyForVpcLattice の追加	ユーザーに代わって Amazon ECS ワークロードの VPC Lattice 機能を管理するために必要な他の AWS サービスリソースへのアクセスを提供します。	2024 年 11 月 18 日
AmazonECSInfrastructureRolePolicyForVolumes へのアクセス許可を追加する	AmazonECSInfrastructureRolePolicyForVolumes ポリシーが更新され、お客様がスナップショットから Amazon EBS ボリュームを作成できるようになりました。	2024 年 10 月 10 日
アクセス許可を the section called “AmazonECS_FullAccess” に追加しました。	AmazonECS_FullAccess ポリシーが更新され、ecsInfrastructureRole という名前のロールの IAM ロールの iam:PassRole アクセス許可が追加されました。これは、AWS マネジメントコンソール によって作成されたデフォルトの IAM ロールで、Amazon ECS が ECS タスクにアタッチされた Amazon EBS ボリュームを管理できる ECS インフラストラクチャロールとして使用されることを目的としています。	2024 年 8 月 13 日

変更	説明	日付
新しく、AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecurity ポリシーを追加しました。	新しく、AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecurity ポリシーを追加しました。これは AWS KMS、AWS Private Certificate Authority、Secrets Manager に管理アクセスを付与し、Amazon ECS Service Connect TLS 機能が正常に動作できるようにするものです。	2024 年 1 月 22 日
新しいポリシー AmazonECSInfrastructureRolePolicyForVolumes を追加しました。	`AmazonECSInfrastructureRolePolicyForVolumes` ポリシーが追加されました。このポリシーにより、Amazon ECS が AWS API コールを行い、Amazon ECS ワークロードに関連付けられた Amazon EBS ボリュームを管理するために必要な権限が付与されます。	2024 年 1 月 11 日
AmazonECSServiceRolePolicy にアクセス許可を追加する	`AmazonECSServiceRolePolicy` マネージド IAM ポリシーが更新され、新しい `events` アクセス許可および、追加権限 `autoscaling` 、`autoscaling-plans` が追加されました。	2023 年 12 月 4 日

変更	説明	日付
許可を AmazonEC2ContainerServiceEventsRole に追加する	AmazonECSServiceRolePolicy マネージド IAM ポリシーが AWS Cloud Map DiscoverInstancesRevision API 操作へのアクセスを許可するように更新されました。	2023 年 10 月 4 日
許可を AmazonEC2ContainerServiceforEC2Role に追加する	AmazonEC2ContainerServiceforEC2Role ポリシーが変更され、新しく作成されたクラスターと登録されたコンテナインスタンスのみに許可を制限する条件を含む ecs:TagResource 許可が追加されました。	2023 年 3 月 6 日
the section called “AmazonECS_FullAccess” へのアクセス許可を追加する	AmazonECS_FullAccess ポリシーが変更され、elasticloadbalancing:AddTags 権限が追加されました。これには、新しく作成されたロードバランサー、ターゲットグループ、ルール、および作成されたリスナーのみにアクセス許可を制限する条件が含まれています。この権限では、既に作成されている Elastic Load Balancing リソースにタグを追加することはできません。	2023 年 1 月 4 日
Amazon ECS が変更の追跡を開始しました。	Amazon ECS が AWS 管理ポリシーの変更の追跡を開始しました。	2021 年 6 月 8 日

AWS Amazon Elastic Container Service の マネージド IAM ポリシーを段階的に廃止します。

以下の AWS 管理 IAM ポリシーは段階的に廃止されます。これらのポリシーは、更新されたポリシーに置き換えられます。ユーザーまたはロールを更新して、更新されたポリシーを使用することをお勧めします。

AmazonEC2ContainerServiceFullAccess

> **Important**
>
> `AmazonEC2ContainerServiceFullAccess` 管理 IAM ポリシーは、2021 年 1 月 29 日に段階的に廃止されました。これは、`iam:passRole` 許可で発見されたセキュリティへの対応です。このアクセス許可は、アカウント内のロールへの認証情報を含むすべてのリソースへのアクセスを許可します。ポリシーが廃止されると、新しいユーザーまたはロールにポリシーをアタッチできなくなります。ポリシーがアタッチされているユーザーまたはロールは、そのポリシーを引き続き使用できます。ただし、ユーザーまたはロールを更新して、`AmazonECS_FullAccess` 管理ポリシーを代わりに使用することをお勧めします 詳細については、「AmazonECS_FullAccess 管理ポリシーへの移行」を参照してください。

AmazonEC2ContainerServiceRole

> **Important**
>
> `AmazonEC2ContainerServiceRole` マネージド IAM ポリシーは段階的に廃止されます。これで Amazon ECS サービスリンクロールに置き換えられます。詳細については、「Amazon ECS のサービスリンクロールの使用」を参照してください。

AmazonEC2ContainerServiceAutoscaleRole

> **Important**
>
> `AmazonEC2ContainerServiceAutoscaleRole` マネージド IAM ポリシーは段階的に廃止されます。これは、Amazon ECS の Application Auto Scaling サービスリンクロールに置き換えられました。詳細については、「Application Auto Scaling ユーザーガイド」の「Application Auto Scaling 用のサービスリンクロール」を参照してください。

AmazonECS_FullAccess 管理ポリシーへの移行

AmazonEC2ContainerServiceFullAccess 管理 IAM ポリシーは、2021 年 1 月 29 日に段階的に廃止されました。これは、iam:passRole アクセス許可で発見されたセキュリティへの対応です。このアクセス許可は、アカウント内のロールへの認証情報を含むすべてのリソースへのアクセスを許可します。ポリシーが廃止されると、新しいグループ、ユーザー、またはロールにポリシーをアタッチできなくなります。すでにポリシーがアタッチされているグループ、ユーザー、またはロールは、引き続きそのポリシーを使用できます。ただし、グループ、ユーザー、またはロールを更新して、AmazonECS_FullAccess が管理するポリシーを代わりに使用することをお勧めします。

AmazonECS_FullAccess によって付与される許可には、ECS を管理者として使用するために必要なアクセス許可の完全なリストが含まれます。AmazonECS_FullAccess ポリシーにはない AmazonEC2ContainerServiceFullAccess ポリシーによって付与されているアクセス許可を現在使用している場合、それらをインラインポリシーステートメントに追加できます。詳細については、「AWSAmazon Elastic Container Service に関する 管理ポリシー」を参照してください。

現在 AmazonEC2ContainerServiceFullAccess 管理 IAM ポリシーを使用しているグループ、ユーザー、またはロールがあるかを判断するには、次のステップを使用します。次に、これらのポリシーを更新して以前のポリシーをデタッチし、AmazonECS_FullAccess ポリシーをアタッチします。

AmazonECS_FullAccess ポリシーを使用するようにグループ、ユーザー、またはロールを更新するには (AWS マネジメントコンソール)

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで [Policies] を選択し、AmazonEC2ContainerServiceFullAccess ポリシーを検索して選択します。
3. [Policy usage (ポリシーの使用)] タブを選択すると、現在このポリシーを使用している IAM ロールが表示されます。
4. 現在 AmazonEC2ContainerServiceFullAccess ポリシーを使用している IAM ロールごとに、ロールを選択し、次の手順を使用して段階的廃止されるポリシーをデタッチし、AmazonECS_FullAccess ポリシーをアタッチします。

    a. [Permissions] (許可) タブで、AmazonEC2ContainerServiceFullAccess ポリシーの横にある X を選択します。

    b. [Add permissions (許可の追加)] を選択します。

    c. [Attach existing policies directly] を選択し、AmazonECS_FullAccess ポリシーを検索してクリックして、[Next: Review] を選択します。

d. 変更を確認し、[Add permissions] を選択します。

e. AmazonEC2ContainerServiceFullAccess ポリシーを使用しているグループ、ユーザー、またはロールごとに、これらの手順を繰り返します。

AmazonECS_FullAccess ポリシーを使用するようにグループ、ユーザー、またはロールを更新するには (AWS CLI)

1. generate-service-last-accessed-details コマンドを使用して、段階的に廃止されたポリシーが最後に使用された日時に関する詳細を含むレポートを生成します。

```
aws iam generate-service-last-accessed-details \
     --arn arn:aws:iam::aws:policy/AmazonEC2ContainerServiceFullAccess
```

出力例:

```
{
    "JobId": "32bb1fb0-1ee0-b08e-3626-ae83EXAMPLE"
}
```

2. get-service-last-accessed-details コマンドを使用して、前回の出力のジョブ ID を指定すると、サービスの最終アクセスレポートを取得できます。このレポートには、段階的に廃止されたポリシーを最後に使用した IAM エンティティの Amazon リソースネーム (ARN) が表示されます。

```
aws iam get-service-last-accessed-details \
      --job-id 32bb1fb0-1ee0-b08e-3626-ae83EXAMPLE
```

3. グループ、ユーザー、またはロールから AmazonEC2ContainerServiceFullAccess ポリシーをデタッチするには、次のコマンドのいずれかを使用します。

- detach-group-policy
- detach-role-policy
- detach-user-policy

4. AmazonECS_FullAccess ポリシーをグループ、ユーザー、またはロールにアタッチするには、次のコマンドのいずれかを使用します。

- attach-group-policy
- attach-role-policy

- attach-user-policy

Amazon ECS のサービスリンクロールの使用

Amazon Elastic Container Service は、AWS Identity and Access Management (IAM) のサービスにリンクされたロールを使用します。サービスリンクロールは、Amazon ECS に直接リンクされた特殊なタイプの IAM ロールです。サービスにリンクされたロールは Amazon ECS で事前定義されています。このロールには、サービスがユーザーに代わって他の AWS のサービスを呼び出すために必要なアクセス許可がすべて付与されています。

トピック

- ロールを使用して Amazon ECS にクラスターの管理を許可する
- ロールを使用して Amazon ECS マネージドインスタンスを管理する

ロールを使用して Amazon ECS にクラスターの管理を許可する

Amazon Elastic Container Service は、AWS Identity and Access Management (IAM) のサービスにリンクされたロールを使用します。サービスリンクロールは、Amazon ECS に直接リンクされた特殊なタイプの IAM ロールです。サービスにリンクされたロールは Amazon ECS で事前定義されています。このロールには、サービスがユーザーに代わって他の AWS のサービスを呼び出すために必要なアクセス許可がすべて付与されています。

サービスにリンクされたロールを使用すると、必要な許可を手動で追加する必要がないため、Amazon ECS のセットアップが簡単になります。サービスリンクロールの許可は Amazon ECS が定義し、特に定義されない限り、Amazon ECS のみがそのロールを引き受けることができます。定義される許可は信頼ポリシーと許可ポリシーに含まれており、その許可ポリシーを他の IAM エンティティにアタッチすることはできません。

サービスリンクロールを削除するには、最初に関連リソースを削除する必要があります。こうすることで、不注意によりリソースにアクセスするための許可を削除することがなくなるため、Amazon ECS リソースが保護されます。

サービスにリンクされたロールをサポートする他のサービスについては、「IAM と連携する AWS のサービス」を参照し、サービスにリンクされたロール列内で [はい] と表記されたサービスを確認してください。サービスリンクロールに関するドキュメントをサービスで表示するには、リンクで [はい] を選択します。

Amazon ECS のサービスリンクロール許可

Amazon ECS は、AWSServiceRoleForECS という名前のサービスにリンクされたロールを使用します。このロールにより、Amazon ECS はクラスターを管理できるようになります。

サービスにリンクされたロール AWSServiceRoleForECS は、次のサービスを信頼してロールを引き受けます。

- `ecs.amazonaws.com`

AmazonECSServiceRolePolicy という名前のロールのアクセス許可ポリシーは、指定したリソースに対して以下のアクションを実行することを Amazon ECS に許可します。

- アクション: `awsvpc` Amazon ECS タスクのネットワークモードを使用する場合、Amazon ECS はタスクに関連するエラスティックネットワークインターフェイスのライフサイクルを管理します。これには、Amazon ECS がエラスティックネットワークインターフェイスに追加するタグも含まれます。
- アクション: Amazon ECS サービスでロードバランサーを使用する場合、Amazon ECS は、ロードバランサーへのリソースの登録と登録解除を管理します。
- アクション: Amazon ECS サービス検出を使用する場合、Amazon ECS は必要な Route 53 を管理し、AWS Cloud Map サービス検出が機能するためのリソース
- アクション: Amazon ECS サービスのオートスケーリングを使用する場合、Amazon ECS は必要な Auto Scaling リソースを管理します。
- アクション: Amazon ECS は、Amazon ECS リソースのモニタリングに役立つ CloudWatch アラームとログストリームを作成および管理します。
- アクション: Amazon ECS Exec を使用する場合、Amazon ECS Exec セッションを開始するために必要なタスクへのアクセス権限は Amazon ECS が管理します。
- アクション: Amazon ECS Service Connect を使用する場合、その機能を使用するために必要な AWS Cloud Map リソースは Amazon ECS が管理します。
- アクション: Amazon ECS キャパシティープロバイダーを使用する場合、Auto Scaling グループとその Amazon EC2 インスタンスを変更するために必要なアクセス権限は Amazon ECS が管理します。
- アクション: Amazon ECS は、Amazon ECS が管理するサービスの AWS Cloud Map サービス属性を更新できます。
- アクション: Amazon ECS は、タスクの開始時と停止時に Amazon EC2 のプロビジョニング及びプロビジョニング解除用 ENI を呼び出すことができます。

- アクション: Amazon ECS は Event Windows に関連付けられたサービスとクラスターのために Amazon EC2 Event Windows を取得することが可能になります。

ユーザー、グループ、またはロールにサービスリンクロールの作成、編集、または削除を許可するには、アクセス許可を設定する必要があります。詳細については、「IAM ユーザーガイド」の「サービスにリンクされたロールのアクセス権限」() を参照してください。

Amazon ECS のサービスリンクロールの作成

サービスリンクロールを手動で作成する必要はありません。AWS マネジメントコンソール、AWS CLI、または AWS API でクラスターを作成したり、サービスを作成または更新したりすると、Amazon ECS によってサービスにリンクされたロールが自動的に作成されます。

> **Important**
>
> このサービスリンクロールはこのロールでサポートされている機能を使用する別のサービスでアクションが完了した場合にアカウントに表示されます。2017 年 1 月 1 日より前に Amazon ECS サービスを使用している場合、サービスにリンクされたロールのサポートの開始時に、Amazon ECS によって AWSServiceRoleForECS ロールがアカウントに作成されています。詳細については、「AWS アカウントに新しいロールが表示される」を参照してください。

AWSServiceRoleForECS ユースケースでサービスにリンクされたロールを作成するには、IAM コンソールを使用します。AWS CLI または AWS API で、IAM を使用してサービスにリンクされたロール (サービス名は `ecs.amazonaws.com`) で作成します。詳細については、IAM ユーザーガイドの「サービスリンクロールの作成」を参照してください。このサービスリンクロールを削除しても、同じ方法でロールを再作成できます。

このサービスリンクロールを削除した後で再度作成する必要が生じた場合は同じ方法でアカウントにロールを再作成できます。クラスターを作成したり、サービスを作成または更新したりすると、Amazon ECS によってサービスにリンクされたロールが自動的に作成されます。

このサービスにリンクされたロールを削除する場合、この同じ IAM プロセスを使用して、もう一度ロールを作成できます。

Amazon ECS のサービスリンクロールの編集

Amazon ECS では、AWSServiceRoleForECS サービスリンクロールを編集できません。サービスにリンクされたロールを作成すると、多くのエンティティによってロールが参照される可能性があるた

め、ロール名を変更することはできません。ただし、IAM を使用してロールの説明を編集することはできます。詳細については、「IAM ユーザーガイド」の「サービスリンクロールの編集」を参照してください。

Amazon ECS のサービスリンクロールの削除

AWSServiceRoleForECS ロールを手動で削除する必要はありません。AWS マネジメントコンソール、AWS CLI、AWS API の全リージョンのクラスターを削除すると、Amazon ECS がリソースをクリーンアップし、サービスリンクロールを削除します。

サービスリンクロールが必要な機能またはサービスが不要になった場合にはそのロールを削除することをお勧めします。そうすることで、モニタリングや保守が積極的に行われていない未使用のエンティティを排除できます。ただし、手動で削除する前に、サービスリンクロールをクリーンアップする必要があります。

サービスリンクロールのクリーンアップ

IAM を使用してサービスにリンクされたロールを削除するには最初に、そのロールで使用されているリソースをすべて削除する必要があります。

> **Note**
>
> リソースを削除しようとしたときに Amazon ECS サービスがロールを使用している場合は、削除が失敗する可能性があります。失敗した場合は数分待ってから操作を再試行してください。

AWSServiceRoleForECS が使用している Amazon ECS リソースを削除するには (コンソール)

1. すべての Amazon ECS サービスをすべてのリージョンで必要数 0 に縮小してから、サービスを削除します。詳細については、「Amazon ECS サービスを更新する」および「コンソールを使用して Amazon ECS サービスの削除」を参照してください。
2. 全リージョンのすべてのクラスターからすべてのコンテナインスタンスを強制的に登録解除します。詳細については、「Amazon ECS コンテナインスタンスの登録を解除する」を参照してください。
3. すべてのリージョンですべての Amazon ECS クラスターを削除します。詳細については、「Amazon ECS クラスターを削除する」を参照してください。

AWSServiceRoleForECS が使用している Amazon ECS リソースを削除するには (AWS CLI)

1. すべての Amazon ECS サービスをすべてのリージョンで必要数 0 に縮小してから、サービスを削除します。詳細については、「AWS Command Line Interfaceリファレンス」の「update–service」および「delete–service」を参照してください。
2. 全リージョンのすべてのクラスターからすべてのコンテナインスタンスを強制的に登録解除します。詳細については、「deregister–container–instance」を参照してください。
3. 全リージョンですべての Amazon ECS クラスターを削除します。詳細については、「delete–cluster」を参照してください。

AWSServiceRoleForECS が使用している Amazon ECS リソースを削除するには (API)

1. すべての Amazon ECS サービスをすべてのリージョンで必要数 0 に縮小してから、サービスを削除します。詳細については、Amazon ECS API リファレンスの「UpdateService」および「DeleteService」を参照してください。
2. 全リージョンのすべてのクラスターからすべてのコンテナインスタンスを強制的に登録解除します。詳細については、「DeregisterContainerInstance」を参照してください。
3. 全リージョンですべての Amazon ECS クラスターを削除します。詳細については、「DeleteCluster」を参照してください。

サービスリンクロールの手動による削除

IAM コンソール、AWS CLI、または AWS API を使用して、サービスにリンクされたロールである AWSServiceRoleForECS を削除します。詳細については、「IAM ユーザーガイド」の「サービスにリンクされたロールの削除」を参照してください。

Amazon ECS サービスリンクロールがサポートされるリージョン

Amazon ECS は、サービスが利用可能なすべてのリージョンでサービスにリンクされたロールの使用をサポートします。詳細については、「AWS リージョンとエンドポイント」を参照してください。

ロールを使用して Amazon ECS マネージドインスタンスを管理する

Amazon Elastic Container Service は、AWS Identity and Access Management (IAM) のサービスにリンクされたロールを使用します。サービスリンクロールは、Amazon ECS に直接リンクされた特殊なタイプの IAM ロールです。サービスにリンクされたロールは Amazon ECS で事前定義されていま

す。このロールには、サービスがユーザーに代わって他の AWS のサービスを呼び出すために必要なアクセス許可がすべて付与されています。

サービスにリンクされたロールを使用すると、必要な許可を手動で追加する必要がないため、Amazon ECS のセットアップが簡単になります。サービスリンクロールの許可は Amazon ECS が定義し、特に定義されない限り、Amazon ECS のみがそのロールを引き受けることができます。定義される許可は信頼ポリシーと許可ポリシーに含まれており、その許可ポリシーを他の IAM エンティティにアタッチすることはできません。

サービスリンクロールを削除するには、最初に関連リソースを削除する必要があります。こうすることで、不注意によりリソースにアクセスするための許可を削除することがなくなるため、Amazon ECS リソースが保護されます。

サービスにリンクされたロールをサポートする他のサービスについては、「IAM と連携する AWS のサービス」を参照し、サービスにリンクされたロール列内で [はい] と表記されたサービスを確認してください。サービスリンクロールに関するドキュメントをサービスで表示するには、リンクで [はい] を選択します。

Amazon ECS のサービスリンクロール許可

Amazon ECS は、AWSServiceRoleForECSCompute という名前のサービスリンクロールを使用して、Amazon ECS マネージドインスタンスキャパシティプロバイダーがプロビジョニングした Amazon EC2 マネージドインスタンスを Amazon ECS で管理できるようにします。

AWSServiceRoleForECSCompute サービスリンクロールは、次のサービスを信頼してロールを引き受けます。

- `ecs-compute.amazonaws.com`

AmazonECSComputeServiceRolePolicy という名前のロールのアクセス許可ポリシーは、Amazon ECS が次のアクションを完了することを許可します。

- Amazon ECS が起動テンプレートを記述および削除する。
- Amazon ECS が起動テンプレートのバージョンを記述および削除する。
- Amazon ECS がインスタンスを終了する。
- Amazon ECS では、次のインスタンスデータパラメータを記述できます。
  - インスタンス
  - インスタンスネットワークインターフェイス: Amazon ECS は、EC2 インスタンスのライフサイクルを管理するインターフェイスを記述できます。

- インスタンスイベントウィンドウ: Amazon ECS は、インスタンスのパッチ適用を目的としたワークフロー中断の可否を決定するためのイベントウィンドウ情報を記述できます。
- インスタンスステータス: Amazon ECS は、インスタンスの状態をモニタリングするためにインスタンスステータスを記述できます。

ユーザー、グループ、またはロールにサービスリンクロールの作成、編集、または削除を許可するには、アクセス許可を設定する必要があります。詳細については、「IAM ユーザーガイド」の「サービスにリンクされたロールのアクセス権限」() を参照してください。

Amazon ECS のサービスリンクロールの作成

サービスリンクロールを手動で作成する必要はありません。AWS マネジメントコンソール、AWS CLI、AWS API で Amazon ECS マネージドインスタンスのキャパシティプロバイダーを作成すると、Amazon ECS がサービスにリンクされたロールを作成します。

> **Important**
>
> このサービスリンクロールはこのロールでサポートされている機能を使用する別のサービスでアクションが完了した場合にアカウントに表示されます。2017 年 1 月 1 日より前に Amazon ECS サービスを使用している場合、サービスにリンクされたロールのサポートが開始された時点で、Amazon ECS が AmazonECSComputeServiceRolePolicy ロールをアカウントに作成しています。詳細については、「AWS アカウントに新しいロールが表示される」を参照してください。

このサービスリンクロールを削除した後で再度作成する必要が生じた場合は同じ方法でアカウントにロールを再作成できます。Amazon ECS マネージドインスタンスのキャパシティプロバイダーを作成すると、Amazon ECS はサービスにリンクされたロールを再度作成します。

このサービスにリンクされたロールを削除する場合、この同じ IAM プロセスを使用して、もう一度ロールを作成できます。

Amazon ECS のサービスリンクロールの編集

Amazon Connect では、サービスにリンクされたロール AmazonECSComputeServiceRolePolicy を編集することはできません。サービスにリンクされたロールを作成すると、多くのエンティティによってロールが参照される可能性があるため、ロール名を変更することはできません。ただし、IAM を使用してロールの説明を編集することはできます。詳細については、「IAM ユーザーガイド」の「サービスリンクロールの編集」を参照してください。

Amazon ECS のサービスリンクロールの削除

AmazonECSComputeServiceRolePolicy を手動で削除する必要はありません。AWS マネジメントコンソール、AWS CLI、AWS API の全リージョンの Amazon ECS マネージドインスタンスキャパシティプロバイダーを削除すると、Amazon ECS がリソースをクリーンアップし、サービスリンクロールを削除します。

サービスリンクロールの手動による削除

IAM コンソール、AWS CLI、AWS API を使用して、AmazonECSComputeServiceRolePolicy サービスリンクロールを削除します。詳細については、「IAM ユーザーガイド」の「サービスにリンクされたロールの削除」を参照してください。

Amazon ECS サービスリンクロールがサポートされるリージョン

Amazon ECS は、サービスが利用可能なすべてのリージョンでサービスにリンクされたロールの使用をサポートします。詳細については、「AWS リージョンとエンドポイント」を参照してください。

Amazon ECS の IAM ロール

IAM ロールは、特定の許可があり、アカウントで作成できるもう 1 つの IAM アイデンティティです。Amazon ECS では、コンテナやサービスなどの Amazon ECS リソースにアクセス許可を付与するロールを作成できます。

Amazon ECS が必要とするロールは、タスク定義の起動タイプと使用する機能によって異なります。次の表を参照して、Amazon ECS に必要な IAM ロールを決定します。

ロール	定義	必要な場合	詳細情報
タスク実行ロール	このロールにより、Amazon ECS がお客様に代わって他の AWS サービスを利用できるようにします。	タスクは AWS Fargate または外部インスタンスホストされています。また、 • Amazon ECR プライベートリポジトリからコンテナイメージをプルします。	Amazon ECS タスク実行IAM ロール

ロール	定義	必要な場合	詳細情報
		• タスクを実行するアカウントとは別のアカウントの Amazon ECR プライベートリポジトリからコンテナイメージをプルします。 • `awslogs` ログドライバーを使用して CloudWatch Logs にコンテナログを送信します。 タスクは、AWS Fargate または Amazon EC2 インスタンスでホストされています。また、 • プライベートレジストリの認証を使用します。 • Runtime Monitoring を使用します。 • タスク定義は、Secrets Manager のシークレットまたは AWS Systems Manager Parameter Store のパラメータを使用して機密データを参照します。	

ロール	定義	必要な場合	詳細情報
タスクロール	このロールにより、(コンテナ上の) アプリケーションコードが他の AWS サービスを使用できるようになります。	アプリケーションは、Amazon S3 などの他の AWS サービスにアクセスします。	Amazon ECS タスクの IAM ロール
コンテナインスタンスのロール	このロールにより、EC2 インスタンスまたは外部インスタンスをクラスターに登録できます。	タスクは Amazon EC2 インスタンスまたは外部インスタンスでホストされています。	Amazon ECS コンテナインスタンスの IAM ロール
Amazon ECS Anywhere ロール	このロールにより、外部インスタンスが AWS API にアクセスできるようになります。	タスクは外部インスタンスでホストされています。	Amazon ECS Anywhere IAM ロール
ロードバランサーロール用の Amazon ECS インフラストラクチャ	このロールにより、Amazon ECS はブルー/グリーンデプロイにおいて、ユーザーに代わってクラスター内のロードバランサーリソースを管理できます。	Amazon ECS のブルー/グリーンデプロイを使用できます。	ロードバランサー用の Amazon ECS インフラストラクチャの IAM ロール
Amazon ECS CodeDeploy ロール	このロールにより、CodeDeploy はサービスを更新できます。	CodeDeploy のブルー/グリーンデプロイタイプを使用して、サービスをデプロイします。	Amazon ECS CodeDeploy IAM ロール

ロール	定義	必要な場合	詳細情報
Amazon ECS EventBridge ロール	このロールにより、EventBridge はサービスを更新できます。	EventBridge のルールとターゲットを使用してタスクをスケジュールします。	Amazon ECS EventBridge IAM ロール
Amazon ECS インフラストラクチャロール	このロールにより、Amazon ECS はクラスター内のインフラストラクチャリソースを管理できます。	• 容量に Amazon ECS マネージドインスタンスを使用します。 • Amazon EBS ボリュームを Fargate または EC2 起動タイプの Amazon ECS タスクにアタッチできます。インフラストラクチャロールにより、Amazon ECS はタスクの Amazon EBS ボリュームを管理できます。 • Amazon ECS Service Connect サービス間のトラフィックを暗号化するには、Transport Layer Security (TLS) を使用します。 • VPC Lattice ターゲットグループを作成する場合。	Amazon ECS インフラストラクチャ IAM ロール

ロール	定義	必要な場合	詳細情報
インスタンスプロファイル	このロールにより、Amazon ECS マネージドインスタンスはインフラストラクチャロールを安全に引き受けることができます。	クラスターで Amazon ECS マネージドインスタンスを使用します。	Amazon ECS マネージドインスタンスのインスタンスプロファイル

Amazon ECS での IAM ロールのベストプラクティス

Amazon ECS が必要とするロールは、タスク定義の起動タイプと使用する機能によって異なります。ロールを共有するのではなく、テーブルに個別のロールを作成することをお勧めします。

ロール	定義	必要な場合	詳細情報
タスク実行ロール	このロールにより、Amazon ECS がお客様に代わって他の AWS サービスを利用できるようにします。	タスクは AWS Fargate または外部インスタンスホストされています。また、 • Amazon ECR プライベートリポジトリからコンテナイメージをプルします。 • タスクを実行するアカウントとは別のアカウントの Amazon ECR プライベートリポジトリからコンテナイメージをプルします。	Amazon ECS タスク実行IAM ロール

ロール	定義	必要な場合	詳細情報
		• `awslogs` ログドライバーを使用して CloudWatch Logs にコンテナログを送信します。 タスクは、AWS Fargate または Amazon EC2 インスタンスでホストされています。また、 • プライベートレジストリの認証を使用します。 • Runtime Monitoring を使用します。 • タスク定義は、Secrets Manager のシークレットまたは AWS Systems Manager Parameter Store のパラメータを使用して機密データを参照します。	
タスクロール	このロールにより、(コンテナ上の) アプリケーションコードが他の AWS サービスを使用できるようになります。	アプリケーションは、Amazon S3 などの他の AWS サービスにアクセスします。	Amazon ECS タスクの IAM ロール

ロール	定義	必要な場合	詳細情報
コンテナインスタンスのロール	このロールにより、EC2 インスタンスまたは外部インスタンスをクラスターに登録できます。	タスクは Amazon EC2 インスタンスまたは外部インスタンスでホストされています。	Amazon ECS コンテナインスタンスの IAM ロール
Amazon ECS Anywhere ロール	このロールにより、外部インスタンスが AWS API にアクセスできるようになります。	タスクは外部インスタンスでホストされています。	Amazon ECS Anywhere IAM ロール
Amazon ECS CodeDeploy ロール	このロールにより、CodeDeploy はサービスを更新できます。	CodeDeploy のブルー/グリーンデプロイタイプを使用して、サービスをデプロイします。	Amazon ECS CodeDeploy IAM ロール
Amazon ECS EventBridge ロール	このロールにより、EventBridge はサービスを更新できます。	EventBridge のルールとターゲットを使用してタスクをスケジュールします。	Amazon ECS EventBridge IAM ロール

ロール	定義	必要な場合	詳細情報
Amazon ECS インフラストラクチャロール	このロールにより、Amazon ECS はクラスター内のインフラストラクチャリソースを管理できます。	• Amazon EBS ボリュームを Fargate または EC2 起動タイプの Amazon ECS タスクにアタッチできます。インフラストラクチャロールにより、Amazon ECS はタスクの Amazon EBS ボリュームを管理できます。 • Amazon ECS Service Connect サービス間のトラフィックを暗号化するには、Transport Layer Security (TLS) を使用します。 • VPC Lattice ターゲットグループを作成する場合。	Amazon ECS インフラストラクチャ IAM ロール

タスクロール

タスクロールを割り当てることをお勧めします。このロールは、それが実行されている Amazon EC2 インスタンスのロールと区別することができます。各タスクにロールを割り当てることは、最小権限アクセスの原則に沿っており、アクションとリソースをよりきめ細かく制御できます。

タスクロールをタスク定義に追加すると、Amazon ECS コンテナエージェントはタスク用の固有の認証情報 ID (例: `12345678-90ab-cdef-1234-567890abcdef`) を持つトークンを自動的に作成します。その後、このトークンとロール認証情報はエージェントの内部キャッシュに追加されます。エージェントは、コンテナ内の環境変数 `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` に認証

情報 ID の URI (例: /v2/credentials/12345678-90ab-cdef-1234-567890abcdef) を入力します。

Amazon ECS コンテナエージェントの IP アドレスに環境変数を追加し、結果の文字列に対して curl コマンドを実行することで、コンテナ内から一時的なロール認証情報を手動で取得できます。

```
curl 169.254.170.2$AWS_CONTAINER_CREDENTIALS_RELATIVE_URI
```

予想される出力は次のようになります。

```
{
 "RoleArn": "arn:aws:iam::123456789012:role/SSMTaskRole-SSMFargateTaskIAMRole-
DASWWSF2WGD6",
 "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
 "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
 "Token": "IQoJb3JpZ2luX2VjEEM/Example==",
 "Expiration": "2021-01-16T00:51:53Z"
}
```

新しいバージョンの AWS SDK では、AWS API 呼び出しを行うと、これらの認証情報が AWS_CONTAINER_CREDENTIALS_RELATIVE_URI 環境変数から自動的に取得されます。認証情報を更新する方法については、rePost の「AWS 認証情報の更新」を参照してください。

出力には、シークレットアクセスキー ID で構成されるアクセスキーペアと、アプリケーションが AWS リソースにアクセスするために使用するシークレットキーが含まれます。また、認証情報が有効であることを確認するために AWS が使用するトークンも含まれています。デフォルトでは、タスクロールを使用してタスクに割り当てられた認証情報は6 時間有効です。その後は、Amazon ECS コンテナエージェントによって自動的にローテーションされます。

タスク実行ロール

タスク実行ロールは、ユーザーに代わって特定の AWS API アクションを呼び出すためのアクセス許可を Amazon ECS コンテナエージェントに付与するのに使用されます。たとえば、AWS Fargate を使用する場合、Fargate には Amazon ECR からイメージをプルして CloudWatch Logs にログを書き込むことができる IAM ロールが必要です。IAM ロールは、イメージプルシークレットなど、AWS Secrets Manager に保存されているシークレットをタスクが参照する場合にも必要です。

> **Note**
>
> 認証されたユーザーとしてイメージをプルする場合、Docker Hub のプルレート制限が変更されても、影響を受ける可能性は低くなります。詳細については、「コンテナインスタンスのプライベートレジストリの認証」を参照してください。
> Amazon ECR と Amazon ECR パブリックを使用することで、Docker によって課せられる制限を回避できます。Amazon ECR からイメージをプルすると、ネットワークのプル時間を短縮し、トラフィックが VPC を離れる際のデータ転送の変更を減らすのにも役立ちます。

> **Important**
>
> Fargate を使用するときは、`repositoryCredentials` を使用してプライベートイメージレジストリの認証を行う必要があります。Amazon ECS コンテナエージェントの環境変数である `ECS_ENGINE_AUTH_TYPE` および `ECS_ENGINE_AUTH_DATA` を設定したり、Fargate でホストされているタスクの `ecs.config` ファイルを変更したりすることはできません。詳細については、「タスクのプライベートレジストリの認証」を参照してください。

コンテナインスタンスのロール

`AmazonEC2ContainerServiceforEC2Role` 管理 IAM ポリシーには以下のアクセス許可が含まれています。最小権限を付与する標準のセキュリティアドバイスに従って、`AmazonEC2ContainerServiceforEC2Role` 管理ポリシーをガイドとして使用できます。ユースケースの管理ポリシーで付与されているアクセス許可が不要な場合、カスタムポリシーを作成し、必要なアクセス許可のみを追加します。

- `ec2:DescribeTags` – (オプション) Amazon EC2 インスタンスに関連付けられているタグをプリンシパルが記述できるようになります。このアクセス許可は、リソースタグの伝播をサポートするために Amazon ECS コンテナエージェントによって使用されます。詳細については、「リソースのタグ付け方法」を参照してください。
- `ecs:CreateCluster` – (任意) プリンシパルが Amazon ECS クラスターを作成できるようになります。このアクセス許可は、Amazon ECS コンテナエージェントによって `default` クラスターが存在しない場合、このクラスタを作成するために使用されます。
- `ecs:DeregisterContainerInstance` – (任意) プリンシパルがクラスターから Amazon ECS コンテナインスタンスを登録解除できるようになります。Amazon ECS コンテナエージェントは

この API 操作を呼び出しませんが、このアクセス許可は後方互換性を確保するために維持されます。

- ecs:DiscoverPollEndpoint – (必須) このアクションは、Amazon ECS コンテナエージェントが更新のポーリングに使用するエンドポイントを返します。
- ecs:Poll – (必須) Amazon ECS コンテナエージェントが Amazon ECS コントロールプレーンと通信し、タスクの状態の変更を報告できるようになります。
- ecs:RegisterContainerInstance – (必須) プリンシパルがコンテナインスタンスをクラスターに登録できるようになります。このアクセス許可は、Amazon ECS コンテナエージェントが Amazon EC2 インスタンスをクラスターに登録し、リソースタグの伝播をサポートするために使用されます。
- ecs:StartTelemetrySession – (任意) Amazon ECS コンテナエージェントが Amazon ECS コントロールプレーンと通信し、各コンテナおよびタスクのヘルス情報とメトリクスをレポートできるようになります。

  このアクセス許可は必須ではありませんが、コンテナインスタンスのメトリクスがスケールアクションを開始できるようにし、ヘルスチェックコマンドに関連するレポートを受信できるようにするために、これを追加することをお勧めします。
- ecs:TagResource – (任意) Amazon ECS コンテナエージェントが、作成時にクラスターにタグ付けしたり、コンテナインスタンスがクラスターに登録されたときにタグ付けしたりできるようになります。
- ecs:UpdateContainerInstancesState — プリンシパルが Amazon ECS コンテナインスタンスのステータスを変更できるようにします。このアクセス許可は、スポットインスタンスのドレイン用に Amazon ECS コンテナエージェントによって使用されます。
- ecs:Submit* – (必須) これには API アクション SubmitAttachmentStateChanges、SubmitContainerStateChange、および SubmitTaskStateChange が含まれています。これらは、Amazon ECS コンテナエージェントによって使用され、各リソースの状態変化を Amazon ECS コントロールプレーンに報告します。SubmitContainerStateChange アクセス許可は、Amazon ECS コンテナエージェントによって使用されなくなりますが、後方互換性を確保するために維持されます。
- ecr:GetAuthorizationToken – (オプション) プリンシパルに認可トークンの取得を許可します。認可トークンは IAM 認証情報を表し、IAM プリンシパルによってアクセスされる Amazon ECR レジストリへのアクセスに使用できます。受け取る認可トークンは 12 時間有効です。
- ecr:BatchCheckLayerAvailability – (任意) コンテナイメージが Amazon ECR プライベートリポジトリにプッシュされると、各イメージレイヤーに対してすでにプッシュされているかどうかが確認されます。その場合、そのイメージレイヤーはスキップされます。

- `ecr:GetDownloadUrlForLayer` – (任意) コンテナイメージが Amazon ECR プライベートリポジトリからプルされると、この API が、まだキャッシュされていない各イメージレイヤーに対して 1 回呼び出されます。
- `ecr:BatchGetImage` – (任意) コンテナイメージが Amazon ECR プライベートリポジトリからプルされると、この API が 1 回呼び出されてイメージマニフェストが取得されます。
- `logs:CreateLogStream` – (任意) プリンシパルが、指定したロググループの CloudWatch Logs ログストリームを作成できるようになります。
- `logs:PutLogEvents` – (任意) プリンシパルが、指定したログストリームにログイベントのバッチをアップロードできるようになります。

サービスリンクロール

Amazon ECS のサービス連動ロールを使用して、Amazon ECS サービスに他のサービス API を代理で呼び出す許可を付与できます。Amazon ECS には、ネットワークインターフェイスの作成と削除、ターゲットグループへのターゲットの登録と登録解除を行う許可が必要です。また、スケーリングポリシーの作成と削除に必要な許可も必要です。これらの許可は、サービスにリンクされたロールによって付与されます。このロールは、サービスを初めて使用するときに、ユーザーに代わって作成されます。

> **Note**
>
> サービスにリンクされたロールを誤って削除してしまった場合は、ロールを再作成できます。手順については、「サービスにリンクされたロールの作成」を参照してください。

ロールの推奨事項

タスクの IAM ロールとポリシーを設定するときは、以下を行うことをお勧めします。

Amazon EC2 メタデータへのアクセスをブロックする

Amazon EC2 インスタンスでタスクを実行するときは、Amazon EC2 インスタンスメタデータサービス (IMDS) へのコンテナアクセスをブロックして、コンテナがインスタンスのインスタンスプロファイル認証情報など、インスタンスレベルの機密データにアクセスできないようにすることを強くお勧めします。アプリケーションが AWS API アクションを呼び出す必要がある場合は、代わりに IAM ロールをタスクに使用してください。

ブリッジモードで実行されているタスクが Amazon EC2 メタデータにアクセスしないようにするには、次のコマンドを実行するか、インスタンスのユーザーデータを更新します。インスタンスのユーザーデータを更新する方法の詳細については、このAWS サポート記事を参照してください。タスク定義ブリッジモードについて詳しくは、「タスク定義ネットワークモード」を参照してください。

```
sudo yum install -y iptables-services; sudo iptables --insert FORWARD 1 --in-interface
 docker+ --destination 169.254.169.254/32 --jump DROP
```

再起動後もこの変更が持続するようにするには、Amazon マシンイメージ (AMI) 固有の次のコマンドを実行します。

- Amazon Linux 2

```
sudo iptables-save | sudo tee /etc/sysconfig/iptables && sudo systemctl enable --now
 iptables
```

- Amazon Linux

```
sudo service iptables save
```

awsvpc ネットワークモードを使用するタスクでは、/etc/ecs/ecs.config ファイル内の環境変数 ECS_AWSVPC_BLOCK_IMDS を true に設定します。

host ネットワーク内で実行されているコンテナが Amazon EC2 メタデータにアクセスできないように、ecs-agent config ファイルの ECS_ENABLE_TASK_IAM_ROLE_NETWORK_HOST 変数を false に設定する必要があります。

awsvpc ネットワークモードを使用する

awsvpc ネットワークモードを使用して、異なるタスクの間、またはタスクと Amazon VPC 内で実行される他のサービスとの間のトラフィックの流れを制限します。これにより、セキュリティの追加レイヤーが追加されます。awsvpc ネットワークモードは、Amazon EC2 で実行されるタスクをタスクレベルでネットワーク分離します。これは AWS Fargate のデフォルトモードであり、タスクにセキュリティグループを割り当てるのに使用できる唯一のネットワークモードです。

最終アクセス時間情報を使用してロールを絞り込む

一度も使用されていないか、しばらく使用されていないアクションは、削除することをお勧めします。そうすることで、望ましくないアクセスが発生するのを防ぐことができます。これを行うに

は、IAM によって提供される最終アクセス時間情報を確認し、使用されたことがない、または最近使用されていないアクションを削除します。これを行うには、次のステップに従ってください。

次のコマンドを実行して、参照されたポリシーの最終アクセス情報を示すレポートを生成します。

```
aws iam generate-service-last-accessed-details --arn arn:aws:iam::123456789012:policy/
ExamplePolicy1
```

出力に表示されていた JobId を使用して、次のコマンドを実行します。これを実行すると、レポートの結果を表示できます。

```
aws iam get-service-last-accessed-details --job-id 98a765b4-3cde-2101-2345-example678f9
```

詳細については、「最終アクセス情報を使用して AWS のアクセス許可を調整する」を参照してください。

AWS CloudTrail に不審なアクティビティがないか監視する

AWS CloudTrail に不審なアクティビティがないか監視することができます。ほとんどの AWS API 呼び出しは AWS CloudTrail イベントとして記録されます。これらは AWS CloudTrail Insights によって分析され、write API 呼び出しに関連する疑わしい動作があればアラートが表示されます。これには、呼び出しの急増が含まれる場合があります。これらのアラートには、異常なアクティビティが発生した時間や、API に寄与した上位のアイデンティティ ARN などの情報が含まれます。

IAM ロールが割り当てられているタスクが実行するアクションは、AWS CloudTrail でイベントの userIdentity プロパティを調べることで特定できます。次の例では、arn に引き受けたロールの名前 s3-write-go-bucket-role が含まれ、その後にタスクの名前 7e9894e088ad416eb5cab92afExample が続きます。

```
"userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROA36C6WWEJ2YEXAMPLE:7e9894e088ad416eb5cab92afExample",
    "arn": "arn:aws:sts::123456789012:assumed-role/s3-write-go-bucket-
role/7e9894e088ad416eb5cab92afExample",
    ...
}
```

> **Note**
>
> ロールを引き受けるタスクが Amazon EC2 コンテナインスタンスで実行されると、リクエストは Amazon ECS コンテナエージェントによって、`/var/log/ecs/audit.log.YYYY-MM-DD-HH` フォーマットのアドレスにあるエージェントの監査ログに記録されます。詳細については、「タスクの IAM ロール ログ」と「証跡のインサイトイベントの記録」を参照してください。

Amazon ECS タスク実行IAM ロール

タスク実行ロールは、ユーザーに代わって AWS API コールを実行するためのアクセス許可を Amazon ECS コンテナと Fargate エージェントに付与します。タスク実行 IAM ロールは、タスクの要件に応じて必要です。さまざまな目的とサービスのタスク実行ロールを、アカウントに複数関連付けることができます。

> **Note**
>
> これらのアクセス許可は、Amazon ECS が定期的にロールの一時的な認証情報を送信することで、インスタンスで実行されているエージェントが使用できるようになりますが、タスク内のコンテナからは直接アクセスできません。コンテナ内のアプリケーションコードの実行に必要な IAM 許可については、「Amazon ECS タスクの IAM ロール」を参照してください。

以下に示しているのは、タスク実行 IAM ロールの一般的なユースケースです。

- タスクは AWS Fargate、Amazon ECS マネージドインスタンス、または外部インスタンスでホストされており、次の動作を行います。
  - Amazon ECR プライベートリポジトリからコンテナイメージをプルします。
  - タスクを実行するアカウントとは別のアカウントの Amazon ECR プライベートリポジトリからコンテナイメージをプルします。
  - `awslogs` ログドライバーを使用して CloudWatch Logs にコンテナログを送信します。詳細については、「Amazon ECS ログを CloudWatch に送信する」を参照してください。
- タスクは、AWS Fargate または Amazon EC2 インスタンスでホストされています。また、
  - プライベートレジストリの認証を使用します。詳細については、「プライベートレジストリ認証のアクセス許可」を参照してください。

- Runtime Monitoring を使用します。
- タスク定義は、Secrets Manager のシークレットまたは AWS Systems Manager Parameter Store のパラメータを使用して機密データを参照します。詳細については、「Secrets Manager または Systems Manager のアクセス許可」を参照してください。

> **Note**
>
> タスク実行ロールは Amazon ECS コンテナエージェントバージョン 1.16.0 以降でサポートされています。

Amazon ECS は、`AmazonECSTaskExecutionRolePolicy` という管理ポリシーを提供します。このポリシーには、前述の一般的ユースケースで必要なアクセス許可が含まれています。詳細については、「AWS Managed Policy リファレンスガイド」の「AmazonECSTaskExecutionRolePolicy」を参照してください。特殊なユースケースでは、タスク実行ロールにインラインポリシーを追加する必要がある可能性があります。

Amazon ECS コンソールで、ECS タスク実行ロールを作成します。追加の機能や機能強化が導入された場合に Amazon ECS がそのアクセス許可を追加できるように、タスクのマネージド IAM ポリシーを手動でアタッチすることができます。IAM コンソールの検索を使用して `ecsTaskExecutionRole` を検索すると、アカウントにすでにタスク実行ロールがあるかどうかを確認できます。詳細については、「IAM ユーザーガイド」の「IAM コンソールの検索」を参照してください。

認証されたユーザーとしてイメージをプルすると、Docker Hub の使用量と制限が変更された場合でも、影響を受ける可能性が低くなります。詳細については、「コンテナインスタンスのプライベートレジストリの認証」を参照してください。

Amazon ECR と Amazon ECR パブリックを使用することで、Docker によって課せられる制限を回避できます。Amazon ECR からイメージをプルすると、ネットワークのプル時間を短縮し、トラフィックが VPC を離れる際のデータ転送の変更を減らすのにも役立ちます。

Fargate を使用するときは、`repositoryCredentials` を使用してプライベートイメージレジストリの認証を行う必要があります。Amazon ECS コンテナエージェントの環境変数である `ECS_ENGINE_AUTH_TYPE` および `ECS_ENGINE_AUTH_DATA` を設定したり、Fargate でホストされているタスクの `ecs.config` ファイルを変更したりすることはできません。詳細については、「タスクのプライベートレジストリの認証」を参照してください。

タスク実行 ロールの作成

アカウントにまだタスク実行ロールがない場合は、次のステップに従ってロールを作成します。

AWS マネジメントコンソール

Elastic Container Service のサービスロールを作成するには (IAM コンソール)

1. AWS マネジメントコンソール にサインインして、IAM コンソール https://console.aws.amazon.com/iam/ を開きます。
2. IAM コンソールのナビゲーションペインで、[ロール]、[ロールを作成] を選択します。
3. 信頼できるエンティティタイプ で、AWS のサービス を選択します。
4. [サービスまたはユースケース] で [エラスティックコンテナサービス] を選択し、次に [エラスティックコンテナのサービスタスク] を選択します。
5. [次へ] を選択します。
6. [アクセス許可の追加] セクションで [AmazonECSTaskExecutionRolePolicy] を検索し、ポリシーを選択します。
7. [次へ] を選択します。
8. [ロール名] に [ECSTaskExecutionRole] と入力します。
9. ロールを確認したら、[ロールを作成] を選択します。

AWS CLI

すべての *[######]* は、お客様の情報で置き換えてください。

1. IAM ロールに使用する信頼ポリシーが含まれている `ecs-tasks-trust-policy.json` という名前のファイルを作成します。ファイルには次の内容が含まれます。

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs-tasks.amazonaws.com"
```

```
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. 前のステップで作成した信頼ポリシーを使用した ecsTaskExecutionRole という名前の IAM ロールを作成します。

```
aws iam create-role \
      --role-name ecsTaskExecutionRole \
      --assume-role-policy-document file://ecs-tasks-trust-policy.json
```

3. AWS 管理 AmazonECSTaskExecutionRolePolicy ポリシー ecsTaskExecutionRole をロールにアタッチします。

```
aws iam attach-role-policy \
      --role-name ecsTaskExecutionRole \
      --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonECSTaskExecutionRolePolicy
```

ロールを作成したら、次の機能のアクセス許可をロールに追加します。

機能	追加のアクセス許可
Secrets Manager 認証情報を使用して、AWS の外部のプライベートレジストリ (Docker Hub、Quay.io、独自のプライベートレジストリなど) からコンテナイメージをプルする	プライベートレジストリ認証のアクセス許可
Systems Manager または Secrets Manager を使用して機密データを渡す	Secrets Manager または Systems Manager のアクセス許可
Fargate タスクが、インターネットエンドポイントを介して Amazon ECR イメージをプルできるようにする	インターフェイスエンドポイントのアクセス許可によって Amazon ECR イメージをプルする Fargate タスクです。
設定ファイルを Amazon S3 バケットでホストする	Amazon S3 ファイルストレージのアクセス許可

機能	追加のアクセス許可
Amazon ECS ライフサイクルイベントを表示するように Container Insights を設定する	Amazon ECS ライフサイクルイベントを Container Insights で表示するために必要なアクセス許可
Container Insights に Amazon ECS ライフサイクルイベントを表示する	Amazon ECS ライフサイクルイベントを Container Insights で表示するには、以下のアクセス許可が必要です。

プライベートレジストリ認証のアクセス許可

プライベートレジストリ認証を使用すると、Amazon ECS タスクは、認証情報を必要とする AWS の外部のプライベートレジストリ (Docker Hub、Quay.io、独自のプライベートレジストリなど) からコンテナイメージをプルできます。この機能は Secrets Manager を使用してレジストリ認証情報を安全に保存し、`repositoryCredentials` パラメータを使用してタスク定義で参照します。

プライベートレジストリ認証の設定の詳細については、「Amazon ECS での非 AWS コンテナイメージの使用」を参照してください。

プライベートレジストリ認証情報を含むシークレットにアクセスできるようにするには、以下のアクセス許可を、インラインポリシーとしてタスクの実行ロールに追加します。詳細については、「IAM ポリシーの追加と削除」を参照してください。

- `secretsmanager:GetSecretValue`— Secrets Manager からプライベートレジストリ認証情報を取得するために必要です。
- `kms:Decrypt` – シークレットでカスタムの KMS キーを使用し、デフォルトのキーを使用しない場合にのみ必須です。カスタムキーの Amazon リソースネーム (ARN) は、リソースとして追加する必要があります。

次の例では、インラインポリシーによりアクセス許可を追加しています。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "secretsmanager:GetSecretValue"
            ],
            "Resource": [
                "arn:aws:secretsmanager:us-
east-1:111122223333:secret:secret_name",
                "arn:aws:kms:us-east-1:111122223333:key/key_id"
            ]
        }
    ]
}
```

Secrets Manager または Systems Manager のアクセス許可

コンテナエージェントが必要な AWS Systems Manager または Secrets Manager のリソースをプルできるようにするためのアクセス許可です。詳細については、「Amazon ECS コンテナに機密データを渡す」を参照してください。

Secrets Manager の使用

作成した Secrets Manager シークレットにアクセスできるようにするには、タスクの実行ロールに対し、以下のアクセス許可を手動により追加します。アクセス許可の管理の詳細については、「IAM ユーザーガイド」の「IAM ID のアクセス許可の追加および削除」を参照してください。

- `secretsmanager:GetSecretValue` – Secrets Manager シークレットを参照する場合に必須です。Secrets Manager からシークレットを取得するための許可を追加します。

次のポリシーの例では、必須のアクセス許可を追加します。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue"
```

```
      ],
      "Resource": [
        "arn:aws:secretsmanager:us-east-1:111122223333:secret:secret_name"
      ]
    }
  ]
}
```

Systems Manager の使用

> **Important**
>
> EC2 起動タイプを使用するタスクの場合、この機能を使用するには ECS エージェント設定変数 `ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE=true` を使用する必要があります。コンテナインスタンスの作成時に `./etc/ecs/ecs.config` ファイルに追加するか、既存のインスタンスに追加して ECS エージェントを再起動できます。詳細については、「Amazon ECS コンテナエージェントの設定」を参照してください。

作成した Systems Manager Parameter Store のパラメータにアクセスできるようにするには、タスク実行ロールに対し、以下のアクセス許可を手動で追加する必要があります。アクセス許可の管理の詳細については、「IAM ユーザーガイド」の「IAM ID のアクセス許可の追加および削除」を参照してください。

- `ssm:GetParameters` – Systems Manager Parameter Store のパラメータをタスク定義で参照している場合は必須です。Systems Manager パラメータを取得するための許可を追加します。
- `secretsmanager:GetSecretValue` – Secrets Manager シークレットをユーザーが直接参照している、あるいは、Systems Manager Parameter Store のパラメータが、タスク定義で Secrets Manager シークレットを参照している場合は必須です。Secrets Manager からシークレットを取得するための許可を追加します。
- `kms:Decrypt` – シークレットが、デフォルトのキーではなく、カスタマーマネージドのキーを使用している場合にのみ必須です。そのカスタムキーの ARN はリソースとして追加されている必要があります。カスタマーマネージドキーを復号するための許可を追加します。

次の例のポリシーでは、必須のアクセス許可を追加します。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ssm:GetParameters",
        "secretsmanager:GetSecretValue",
        "kms:Decrypt"
      ],
      "Resource": [
        "arn:aws:ssm:us-east-1:111122223333:parameter/parameter_name",
        "arn:aws:secretsmanager:us-east-1:111122223333:secret:secret_name",
        "arn:aws:kms:us-east-1:111122223333:key/key_id"
      ]
    }
  ]
}
```

インターフェイスエンドポイントのアクセス許可によって Amazon ECR イメージをプルする Fargate タスクです。

Amazon ECR がインターフェイス VPC エンドポイントを使用するように設定されている場合、Amazon ECR からイメージをプルする Fargate を使用するタスクを起動するときは、特定の VPC または VPC エンドポイントへのアクセスにタスクを制限できます。この操作を行うには、IAM 条件キーを使用するタスクのタスク実行ロールを作成します。

次の IAM グローバル条件キーを使用して、特定の VPC または VPC エンドポイントへのアクセスを制限します。詳細については、「AWSvグローバル条件コンテキストキー」を参照してください。

- `aws:SourceVpc` - 特定の VPC へのアクセスを制限します。タスクとエンドポイントをホストする VPC に制限できます。
- `aws:SourceVpce` - 特定の VPC エンドポイントへのアクセスを制限します。

次のタスク実行ロールポリシーは、条件キーを追加する方法の例を示しています。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
            {
                "Effect": "Allow",
                "Action": [
                    "ecr:GetAuthorizationToken",
                    "logs:CreateLogStream",
                    "logs:PutLogEvents"
                ],
                "Resource": "*"
            },
            {
                "Effect": "Allow",
                "Action": [
                    "ecr:BatchCheckLayerAvailability",
                    "ecr:GetDownloadUrlForLayer",
                    "ecr:BatchGetImage"
                ],
                "Resource": "arn:aws:ecr:*:*:repository/*",
                "Condition": {
                    "StringEquals": {
                            "aws:sourceVpce": "vpce-0123456789abcdef0"
                    }
                }
            }
    ]
}
```

Amazon ECR のアクセス許可

Amazon ECR プライベートリポジトリからコンテナイメージをプルする必要がある場合は、以下のアクセス許可が必要です。タスクの実行ロールには、Amazon ECS コンテナと Fargate エージェントがユーザーに代わってコンテナイメージをプルできるように、これらのアクセス許可が必要です。基本的な ECS 実装の場合、これらのアクセス許可は、タスクの IAM ロールではなく、タスクの実行ロールに追加する必要があります。

Amazon ECS タスクの実行ロール管理ポリシー (AmazonECSTaskExecutionRolePolicy) には、Amazon ECR からイメージをプルするために必要なアクセス許可が含まれています。マネージドポリシーを使用している場合は、これらのアクセス許可を個別に追加する必要はありません。

カスタムポリシーを作成する場合は、以下のアクセス許可を含めて、Amazon ECR からのイメージのプルを許可します。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        }
    ]
}
```

これらのアクセス許可は、アプリケーションコードが Amazon ECR API と直接やり取りする必要がある場合に、タスクの IAM ロールで必要となるアクセス許可とは異なることに注意してください。Amazon ECR におけるタスクの IAM ロールのアクセス許可の詳細については、「Amazon ECR のアクセス許可」を参照してください。

Amazon S3 ファイルストレージのアクセス許可

Amazon S3 でホストされる設定ファイルを指定する場合、タスク実行ロールには、設定ファイルに対する s3:GetObject アクセス許可と、ファイルが格納されている Amazon S3 バケットでの s3:GetBucketLocation アクセス許可が含まれている必要があります。詳細については、「Amazon Simple Storage Service ユーザーガイド」の「Amazon S3 のポリシーアクション」を参照してください。

次のポリシーの例では、Amazon S3 からファイルを取得するために必要なアクセス許可を追加します。Amazon S3 バケットの名前と設定ファイル名を指定します。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket/folder_name/config_file_name"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetBucketLocation"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket"
      ]
    }
  ]
}
```

セキュリティに関する重要な考慮事項

Amazon S3 バケットと統合する Amazon ECS 機能を使用する場合は、バケットの所有権検証を実装して、バケット乗っ取り攻撃を防止します。適切な検証を実行しない場合、悪意のあるアクターが Amazon S3 バケットを削除し、同じ名前のバケットを再作成すると、知らないうちにタスクが悪意のある設定を読み込む、攻撃者が管理するバケットに機密データを送信するなどの事態が発生する可能性があります。

推奨される IAM ポリシー条件:

```
            "Condition": {
              "StringEquals": {
                "aws:ResourceAccount": "TRUSTED-ACCOUNT-ID"
              }
```

```
            }
```

TRUSTED-ACCOUNT-ID を、S3 バケットを所有する AWS アカウント ID に置き換えます。

この条件により、タスク実行ロールは、指定された信頼されたアカウントが所有する Amazon S3 バケットにのみアクセスできるようになります。

Amazon ECS タスクの IAM ロール

Amazon ECS タスクには IAM ロールを関連付けることができます。IAM ロールで付与されるアクセス許可は、タスクで実行中のコンテナに対し発行されます。このロールにより、(コンテナ上で実行) アプリケーションコードが他の AWS サービスを使用できるようになります。タスクロールは、アプリケーションが Amazon S3 などの他の AWS サービスにアクセスするときに必要です。

> **Note**
>
> これらのアクセス許可は、Amazon ECS コンテナと Fargate エージェントからはアクセスされません。Amazon ECS がコンテナイメージをプルしてタスクを実行するために必要な IAM 許可については、「Amazon ECS タスク実行IAM ロール」を参照してください。

タスクロールを使用する利点は次のとおりです。

- 懸念事項の分離: EC2 を使用している場合、タスク IAM ロールを使用すると、EC2 インスタンスプロファイルを使わずにコンテナの IAM アクセス許可を指定できます (詳細については、「AWS Identity and Access Management ユーザーガイド」の「インスタンスプロファイルの使用」を参照してください)。したがって、EC2 インスタンスに関連付けられた IAM アクセス許可を変更することなく、アプリケーションを ECS コンテナインスタンスに個別かつ均一にデプロイできます。
- 監査性: CloudTrail にアクセスしイベントのログ記録を利用することで、遡及的な監査を確実に行えます。タスクの認証情報には、セッションにアタッチされている `taskArn` のコンテキストがあるため、CloudTrail ログにはロールの認証情報が発行されたタスクが表示されます。
- 統一された認証情報配信: ECS は IAM ロール認証情報をコンテナに配信し、タスクに関連付けられたコンピューティングオプションに関係なく、明確に定義されたインターフェイスを介してアクセスできるようにします。ECS Fargate では、EC2 インスタンスプロファイルはタスク内のコンテナでは使用できません。タスク IAM ロールを使用すると、コンテナで AWS SDK または AWS CLI を使用する場合、コンピューティングオプションに関係なく、コンテナに IAM アクセス許可を関連付けることができます。AWS SDK がこれらの認証情報にアクセスする方法の詳細については、「コンテナ認証情報プロバイダー」を参照してください。

> **Important**
>
> コンテナはセキュリティ境界ではなく、タスク IAM ロールを使用しても変更されません。Fargate で実行されている各タスクは、独自の分離境界を持ち、基盤となるカーネル、CPU リソース、メモリリソース、Elastic Network Interface を別のタスクと共有しません。分離要件が厳しいワークロードの場合は、Fargate を使用してください。EC2、ECS マネージドインスタンス、ECS Anywhere コンテナインスタンスの場合、タスクの分離はなく、コンテナは同じコンテナインスタンス上の他のタスクの認証情報にアクセスできる可能性があります。また、ECS コンテナインスタンスロールに割り当てられたアクセス許可や、Amazon EC2 インスタンスメタデータサービス (IMDS) を介してアクセス可能なインスタンスメタデータや IAM ロール認証情報などのその他の機密データ、および共存タスクのデータにアクセスすることもできます。ロールの推奨事項に従って、コンテナの Amazon EC2 インスタンスメタデータサービスへのアクセスをブロックします (詳細については、「Amazon EC2 ユーザーガイド」の「インスタンスメタデータサービスを使用してインスタンスメタデータにアクセスする」を参照してください)。
> タスク用の IAM ロールを指定すると、そのタスクのコンテナ内の AWS CLI または他の SDK は、タスクロールによって提供された AWS 認証情報を排他的に使用し、Amazon EC2 または外部インスタンスで実行されていて、それらから IAM 許可を継承しなくなる点に注意してください。

タスクの IAM ロールを作成する

タスクで使用する IAM ポリシーを作成するときは、そのポリシーはタスクのコンテナが引き受けるタスクの許可を含める必要があります。既存の AWS マネージドポリシーを使用することも、特定のニーズを満たすカスタムポリシーを最初から作成することもできます。詳細については、「IAM ユーザーガイド」の「IAM ポリシーの作成」を参照してください。

> **Important**
>
> Amazon ECS タスク (すべての起動タイプ) では、タスクに IAM ポリシーとロールを使用することをお勧めします。これらの認証情報により、タスクは `sts:AssumeRole` を呼び出してタスクに既に関連付けられているのと同じロールを引き受けることなく、AWS API リクエストを行うことができます。タスクでロールがそれ自体を引き受けることが必要な場合、そのロールがそれ自体を引き受けることを明示的に許可する信頼ポリシーを作成する必要があります。詳細については、「IAM ユーザーガイド」の「ロール信頼ポリシーの更新」を参照してください。

IAM ポリシーを作成した後、Amazon ECS タスク定義で参照するポリシーを含む IAM ロールを作成できます。IAM コンソールで [Elastic Container Service Task] (Elastic Container Service タスク) ユースケースを使用してロールを作成できます。その後、タスクのコンテナに必要なアクセス許可を付与するロールに特定の IAM ポリシーをアタッチできます。以下の手順でこれを行う方法について説明します。

IAM のアクセス許可を必要とする複数のタスク定義またはサービスがある場合は、各タスクに提供するアクセスを最小限に抑えるために、タスクを操作するために必要な最小限のアクセス許可で特定のタスク定義またはサービスごとにロールを作成することを検討してください。

リージョンのサービスエンドポイントの詳細については、「Amazon Web Services 全般のリファレンス ガイド」の「サービスエンドポイント」を参照してください。

IAM タスクロールは、`ecs-tasks.amazonaws.com` サービスを特定する信頼ポリシーが必要です。`sts:AssumeRole` 許可は、Amazon EC2 インスタンスが使用するロールとは異なる IAM ロールをタスクに引き受けられるようにします。これにより、タスクは Amazon EC2 インスタンスに関連付けられたロールを継承しません。以下に示しているのは、信頼ポリシーの例です。リージョン識別子を置き換えて、タスク起動時に使用する AWS アカウント番号を特定します。

> **Important**
>
> タスク IAM ロールを作成するときは、ロールに関連付けられた信頼関係ポリシーで `aws:SourceAccount` または `aws:SourceArn` のいずれかの条件キーを使用してアクセス許可のスコープを明確にし、混乱した代理のセキュリティ問題の発生を防止するよう推奨します。特定のクラスターを指定する `aws:SourceArn` 条件キーの使用は現在サポートされていません。ワイルドカードを使用してすべてのクラスターを指定する必要があります。混乱した代理の問題と、お客様の AWS アカウントの保護方法の詳細について学ぶためには、[The confused deputy problem] (IAM ユーザーガイド) の[IAM User Guide] (混乱した代理の問題) を参照してください。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
```

```
            "Service":[
               "ecs-tasks.amazonaws.com"
            ]
         },
         "Action":"sts:AssumeRole",
         "Condition":{
            "ArnLike":{
            "aws:SourceArn":"arn:aws:ecs:us-west-2:111122223333:*"
            },
            "StringEquals":{
               "aws:SourceAccount":"111122223333"
            }
         }
      }
   ]
}
```

以下の手順では、Amazon S3 からオブジェクトを取得するポリシーを作成する方法を、ポリシー例とともに説明します。すべての######を自分の値に置き換えてください。

AWS マネジメントコンソール

JSON ポリシーエディタでポリシーを作成するには

1. AWS マネジメントコンソール にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左側のナビゲーションペインで、[ポリシー] を選択します。

   初めて [ポリシー] を選択する場合には、[管理ポリシーにようこそ] ページが表示されます。今すぐ始める を選択します。
3. ページの上部で、[ポリシーを作成] を選択します。
4. ポリシーエディタ セクションで、JSON オプションを選択します。
5. 次の JSON ポリシードキュメントを入力します。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
```

```
            "s3:GetObject"
         ],
         "Resource":[
            "arn:aws:s3:::my-task-secrets-bucket/*"
         ]
      }
   ]
}
```

6. 次へ を選択します。

> **Note**
>
> いつでも Visual と JSON エディタオプションを切り替えることができます。ただし、[ビジュアル] エディタで [次へ] に変更または選択した場合、IAM はポリシーを再構成してビジュアルエディタに合わせて最適化することがあります。詳細については、IAM ユーザーガイド の ポリシーの再構成 を参照してください。

7. 確認と作成 ページで、作成するポリシーの ポリシー名 と 説明 (オプション) を入力します。このポリシーで定義されているアクセス許可 を確認して、ポリシーによって付与されたアクセス許可を確認します。
8. ポリシーを作成 をクリックして、新しいポリシーを保存します。

AWS CLI

すべての######を自分の値に置き換えてください。

1. s3-policy.json というファイルを次の内容で作成します。

   JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "s3:GetObject"
         ],
```

```
            "Resource":[
               "arn:aws:s3:::my-task-secrets-bucket/*"
            ]
         }
      ]
}
```

2. JSON ポリシードキュメントファイルを使用して IAM ポリシーを作成するには、次のコマンドを使用します。すべての`######`を自分の値に置き換えてください。

```
aws iam create-policy \
      --policy-name taskRolePolicy \
      --policy-document file://s3-policy.json
```

次の手順を使用してサービスロールを作成します。

AWS マネジメントコンソール

Elastic Container Service のサービスロールを作成するには (IAM コンソール)

1. AWS マネジメントコンソール にサインインして、IAM コンソール https://console.aws.amazon.com/iam/ を開きます。
2. IAM コンソールのナビゲーションペインで、[ロール]、[ロールを作成] を選択します。
3. 信頼できるエンティティタイプ で、AWS のサービス を選択します。
4. [サービスまたはユースケース] で [エラスティックコンテナサービス] を選択し、次に [エラスティックコンテナのサービスタスク] を選択します。
5. [次へ] を選択します。
6. [アクセス許可の追加] で、作成したポリシーを検索し、選択します。
7. [次へ] を選択します。
8. ロール名 に、ロールの名前を入力します。この例では、`AmazonECSTaskS3BucketRole` と入力してロールに名前を付けます。
9. ロールを確認したら、[ロールを作成] を選択します。

AWS CLI

1. タスク IAM ロールに使用する信頼ポリシーが含まれている `ecs-tasks-trust-policy.json` という名前のファイルを作成します。ファイルには次の内容が含まれます。

リージョン識別子を置き換えて、タスク起動時に使用する AWS アカウント番号を特定します。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":[
               "ecs-tasks.amazonaws.com"
            ]
         },
         "Action":"sts:AssumeRole",
         "Condition":{
            "ArnLike":{
            "aws:SourceArn":"arn:aws:ecs:us-west-2:111122223333:*"
            },
            "StringEquals":{
               "aws:SourceAccount":"111122223333"
            }
         }
      }
   ]
}
```

2. 前のステップで作成した信頼ポリシーを使用した ecsTaskRole という名前の IAM ロールを作成します。

```
aws iam create-role \
      --role-name ecsTaskRole \
      --assume-role-policy-document file://ecs-tasks-trust-policy.json
```

3. 次のコマンドを使用して、作成した IAM ポリシーの ARN を取得します。*taskRolePolicy* は、作成したポリシーの名前に置き換えます。

```
aws iam list-policies --scope Local --query 'Policies[?
PolicyName==`taskRolePolicy`].Arn'
```

4. 作成した IAM ポリシーを、ecsTaskRole ロールにアタッチします。policy-arn を、作成したポリシーの ARN に置き換えます。

```
aws iam attach-role-policy \
      --role-name ecsTaskRole \
      --policy-arn arn:aws:iam:111122223333:aws:policy/taskRolePolicy
```

ロールを作成したら、次の機能のアクセス許可をロールに追加します。

機能	追加のアクセス許可
ECS Exec を使用する	ECS Exec のアクセス許可
プライベート Amazon ECR リポジトリからのイメージを使用する	Amazon ECR のアクセス許可
EC2 インスタンスを使用する (Windows および Linux)	Amazon EC2 インスタンスの追加設定
外部インスタンスを使用する	外部インスタンスの追加設定
EC2 Windows インスタンスを使用する	Amazon EC2 Windows インスタンスの追加設定

Amazon ECR のアクセス許可

アプリケーションコードが直接 Amazon ECR リポジトリとやり取りする必要がある場合は、次のアクセス許可が必要です。Amazon ECR からイメージをプルするだけの基本的な実装では、これらのアクセス許可はタスク IAM ロールレベルでは必要ありません。代わりに、Amazon ECS タスク実行ロールにこれらのアクセス許可が必要です。タスクの実行ロールの詳細については、「Amazon ECS タスク実行IAM ロール」を参照してください。

コンテナで実行されているアプリケーションコードが Amazon ECR API と直接やり取りする必要がある場合は、タスクの IAM ロールに次のアクセス許可を追加し、タスク定義にタスクの IAM ロールを含める必要があります。詳細については、IAM ユーザーガイドのIAM ポリシーの追加と削除を参照してください。

タスク IAM ロールに次のポリシーを使用して、Amazon ECR と直接やり取りする必要があるコンテナアプリケーションに必要な Amazon ECR アクセス許可を追加します。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        }
    ]
}
```

ECS Exec のアクセス許可

ECS Exec 機能には、マネージド型 SSM エージェント (execute-command エージェント) と SSM サービス間の通信に必要なアクセス許可をコンテナに付与するためのタスク IAM ロールが必要です。次のアクセス許可をタスク IAM ロールに追加し、タスク定義にタスク IAM ロールを含める必要があります。詳細については、IAM ユーザーガイドのIAM ポリシーの追加と削除を参照してください。

タスク IAM ロールに次のポリシーを使用して、必要な SSM アクセス許可を追加します。

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
       {
       "Effect": "Allow",
       "Action": [
            "ssmmessages:CreateControlChannel",
            "ssmmessages:CreateDataChannel",
```

```
            "ssmmessages:OpenControlChannel",
            "ssmmessages:OpenDataChannel"
        ],
        "Resource": "*"
        }
    ]
}
```

Amazon EC2 インスタンスの追加設定

コンテナインスタンスのロールの許可は、以下に提供されるマネージド型 AmazonEC2ContainerServiceforEC2Role IAM ポリシーの許可のミニマリストに制限することをお勧めします。

Amazon EC2 インスタンスは、タスクのロールを使用するために、コンテナエージェントのバージョン 1.11.0 以上が必要です。ただし、最新のコンテナエージェントのバージョンを使用することをお勧めします。エージェントのバージョンの確認と最新バージョンへの更新については、「Amazon ECS コンテナエージェントをアップデートする」を参照してください。Amazon ECS に最適化された AMI を使用している場合、インスタンスには、ecs-init パッケージの 1.11.0-1 以降が必要です。インスタンスが最新 Amazon ECS に最適化された AMI を使用している場合、コンテナエージェントおよび ecs-init の必要なバージョンが含まれます。詳細については、「Amazon ECS に最適化された Linux AMI」を参照してください。

コンテナインスタンスで Amazon ECS に最適化された AMI を使用していない場合、エージェントを開始する --net=host コマンドと、目的の設定に次のエージェント設定変数に docker run オプションを追加してください (詳細については、「Amazon ECS コンテナエージェントの設定」を参照してください)。

ECS_ENABLE_TASK_IAM_ROLE=true

bridge および default のネットワークモードに設定されたコンテナのタスク用の、IAM ロールを使用します。

ECS_ENABLE_TASK_IAM_ROLE_NETWORK_HOST=true

host ネットワークモードに設定されたコンテナのタスク用の、IAM ロールを使用します。この変数はエージェントバージョン 1.12.0 以降でのみサポートされています。

実行コマンドの例の詳細については、「Amazon ECS コンテナエージェントの手動更新 (Amazon ECS 最適化以外の AMI の場合)」を参照してください。また、タスクのコンテナが AWS 認証情報

を取得できるよう、次のネットワーキングコマンドをコンテナインスタンスで設定する必要があります。

```
sudo sysctl -w net.ipv4.conf.all.route_localnet=1
sudo iptables -t nat -A PREROUTING -p tcp -d 169.254.170.2 --dport 80 -j DNAT --to-destination 127.0.0.1:51679
sudo iptables -t nat -A OUTPUT -d 169.254.170.2 -p tcp -m tcp --dport 80 -j REDIRECT --to-ports 51679
```

再起動後も有効にするには、コンテナインスタンスにこれらの iptables ルールを保存する必要があります。iptables-save および iptables-restore コマンドを使用して、iptables ルールを保存し、起動時にそのルールを復元します。詳細については、特定のオペレーティングシステムのドキュメントを参照してください。

awsvpc ネットワークモードを使用するタスクで実行されたコンテナが、Amazon EC2 のインスタンスプロファイルに入力されている認証情報にアクセスを防止するためには (タスクロールで指定されている許可は有効)、エージェントの設定 ファイルで ECS_AWSVPC_BLOCK_IMDS エージェント設定変数を true に設定し、そのエージェントを再起動します。詳細については、「Amazon ECS コンテナエージェントの設定」を参照してください。

bridge ネットワークモードを使用するタスクで実行されたコンテナが、Amazon EC2 のインスタンスプロファイルに入力されている認証情報にアクセスを防止するためには (タスクロールで指定されている許可は有効)、Amazon EC2 インスタンスで次の iptables コマンドを実行します。このコマンドは、host または awsvpc ネットワークモードを使用するタスク内のコンテナに影響を与えることはありません。詳細については、「ネットワークモード」を参照してください。

- ```
  sudo yum install -y iptables-services; sudo iptables --insert DOCKER-USER 1 --in-interface docker+ --destination 169.254.169.254/32 --jump DROP
  ```

  再起動後も有効にするには、Amazon EC2 インスタンスでこの iptables ルールを保存する必要があります。Amazon ECS に最適化された AMI を使用する場合は、次のコマンドを使用します。他のオペレーティングシステムの場合、そのオペレーティングシステムのドキュメントを参照してください。

  ```
  sudo iptables-save | sudo tee /etc/sysconfig/iptables && sudo systemctl enable --now iptables
  ```
  ```

外部インスタンスの追加設定

外部インスタンスは、タスクの IAM ロールを使用にするために、コンテナエージェントのバージョン 1.11.0 以上が必要ですが、最新のコンテナエージェントのバージョンを使用することをお勧めします。エージェントのバージョンの確認と最新バージョンへの更新については、「Amazon ECS コンテナエージェントをアップデートする」を参照してください。Amazon ECS に最適化された AMI を使用している場合、インスタンスは、1.11.0-1 パッケージの ecs-init バージョン以降が必要です。インスタンスが最新 Amazon ECS に最適化された AMI を使用している場合、コンテナエージェントおよび ecs-init の必要なバージョンが含まれます。詳細については、「Amazon ECS に最適化された Linux AMI」を参照してください。

コンテナインスタンスで Amazon ECS に最適化された AMI を使用していない場合、エージェントを開始する --net=host コマンドと、目的の設定に次のエージェント設定変数に docker run オプションを追加してください (詳細については、「Amazon ECS コンテナエージェントの設定」を参照してください)。

ECS_ENABLE_TASK_IAM_ROLE=true

bridge および default のネットワークモードに設定されたコンテナのタスク用の、IAM ロールを使用します。

ECS_ENABLE_TASK_IAM_ROLE_NETWORK_HOST=true

host ネットワークモードに設定されたコンテナのタスク用の、IAM ロールを使用します。この変数はエージェントバージョン 1.12.0 以降でのみサポートされています。

実行コマンドの例の詳細については、「Amazon ECS コンテナエージェントの手動更新 (Amazon ECS 最適化以外の AMI の場合)」を参照してください。また、タスクのコンテナが AWS 認証情報を取得できるよう、次のネットワーキングコマンドをコンテナインスタンスで設定する必要があります。

```
sudo sysctl -w net.ipv4.conf.all.route_localnet=1
sudo iptables -t nat -A PREROUTING -p tcp -d 169.254.170.2 --dport 80 -j DNAT --to-
destination 127.0.0.1:51679
sudo iptables -t nat -A OUTPUT -d 169.254.170.2 -p tcp -m tcp --dport 80 -j REDIRECT --
to-ports 51679
```

再起動後も有効にするには、コンテナインスタンスにこれらの iptables ルールを保存する必要があります。iptables-save および iptables-restore コマンドを使用して、iptables ルールを保存し、起動時

にそのルールを復元します。詳細については、特定のオペレーティングシステムのドキュメントを参照してください。

Amazon EC2 Windows インスタンスの追加設定

> **Important**
>
> これは、タスクロールを使用する EC2 上にある Windows コンテナにのみ適用されます。

Windows 機能を使用するタスクロールには EC2 での追加設定が必要です。

- コンテナインスタンスを起動する際に、コンテナインスタンスのユーザーデータスクリプトの `-EnableTaskIAMRole` オプションを設定して、この機能を有効にする必要があります。`EnableTaskIAMRole`は、タスクの Task IAM ロール機能をオンにします。例:

```
<powershell>
Import-Module ECSTools
Initialize-ECSAgent -Cluster 'windows' -EnableTaskIAMRole
</powershell>
```

- Amazon ECS コンテナブートストラップスクリプト に提供されているネットワーキングコマンドを使用してコンテナをブートストラップする必要があります。
- タスク用の IAM ロールとポリシーを作成する必要があります。詳細については、「タスクの IAM ロールを作成する」を参照してください。
- タスク認証情報プロバイダーの IAM ロールは、コンテナインスタンスのポート 80 を使用します。したがって、コンテナインスタンスでタスクの IAM ロールを設定にすると、コンテナはポートマッピングのホストポートにポート 80 を使用できません。コンテナをポート 80 で公開するには、負荷分散を使用するサービスをコンテナ用に設定することをお勧めします。ロードバランサーのポート 80 を使用できます。これにより、コンテナインスタンス上の別のホストポートにトラフィックをルーティングできます。詳細については、「ロードバランサーを使用して Amazon ECS サービストラフィックを分散する」を参照してください。
- Windows インスタンスが再起動された場合は、プロキシインターフェイスを削除して、Amazon ECS コンテナエージェントを再度初期化し、認証情報プロキシを元に戻す必要があります。

Amazon ECS コンテナブートストラップスクリプト

コンテナからコンテナインスタンスの認証情報プロキシにアクセスする前に、必要なネットワーキングコマンドを使用してコンテナをブートストラップする必要があります。起動時にコンテナで次のコードのサンプルスクリプトを実行する必要があります。

> **Note**
>
> Windows で awsvpc ネットワークモードを使用している場合は、このスクリプトを実行する必要はありません。

Powershell を含む Windows コンテナを実行する場合は、次のスクリプトを使用します。

```
# Copyright Amazon.com Inc. or its affiliates. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License"). You may
# not use this file except in compliance with the License. A copy of the
# License is located at
#
# http://aws.amazon.com/apache2.0/
#
# or in the "license" file accompanying this file. This file is distributed
# on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
# express or implied. See the License for the specific language governing
# permissions and limitations under the License.

$gateway = (Get-NetRoute | Where { $_.DestinationPrefix -eq '0.0.0.0/0' } | Sort-Object
 RouteMetric | Select NextHop).NextHop
$ifIndex = (Get-NetAdapter -InterfaceDescription "Hyper-V Virtual Ethernet*" | Sort-
Object | Select ifIndex).ifIndex
New-NetRoute -DestinationPrefix 169.254.170.2/32 -InterfaceIndex $ifIndex -NextHop
 $gateway -PolicyStore ActiveStore # credentials API
New-NetRoute -DestinationPrefix 169.254.169.254/32 -InterfaceIndex $ifIndex -NextHop
 $gateway -PolicyStore ActiveStore # metadata API
```

コマンドシェルのみを持つ Windows コンテナを実行する場合は、次のスクリプトを使用します。

```
# Copyright Amazon.com Inc. or its affiliates. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License"). You may
# not use this file except in compliance with the License. A copy of the
```

```
# License is located at
#
# http://aws.amazon.com/apache2.0/
#
# or in the "license" file accompanying this file. This file is distributed
# on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
# express or implied. See the License for the specific language governing
# permissions and limitations under the License.

for /f "tokens=1" %i  in ('netsh interface ipv4 show interfaces ^| findstr /x /r
 ".*vEthernet.*"') do set interface=%i
for /f "tokens=3" %i  in ('netsh interface ipv4 show addresses %interface% ^| findstr /
x /r ".*Default.Gateway.*"') do set gateway=%i
netsh interface ipv4 add route prefix=169.254.170.2/32 interface="%interface%"
 nexthop="%gateway%" store=active # credentials API
netsh interface ipv4 add route prefix=169.254.169.254/32 interface="%interface%"
 nexthop="%gateway%" store=active # metadata API
```

Amazon ECS コンテナインスタンスの IAM ロール

Amazon ECS コンテナインスタンス (Amazon EC2 と外部インスタンスの両方を含む) では、Amazon ECS コンテナエージェントを実行し、エージェントがユーザーに属していることをサービスに伝える IAM ロールが必要です。コンテナインスタンスを起動してクラスターに登録する前に、使用するコンテナインスタンス用の IAM ロールを作成する必要があります。ロールは、コンソールへのログインまたは AWS CLI コマンドの実行に使用するアカウントで作成されます。

> **Important**
>
> クラスターに外部インスタンスを登録する場合、使用する IAM ロールには Systems Manager のアクセス許可も必要です。詳細については、「Amazon ECS Anywhere IAM ロール」を参照してください。

Amazon ECS では、`AmazonEC2ContainerServiceforEC2Role` 管理 IAM ポリシーを提供し、これには Amazon ECS の完全な機能セットを使用するために必要なアクセス権限が含まれます。この管理ポリシーは、IAM ロールにアタッチし、コンテナインスタンスに関連付けることができます。または、使用するカスタムポリシーを作成するときに、管理ポリシーをガイドとして使用することもできます。コンテナインスタンスロールには、Amazon ECS コンテナエージェントと Docker デーモンがユーザーに変わって AWS API を呼び出すために必要な許可があります。管理ポリシーの詳細については、「AmazonEC2ContainerServiceforEC2Role」を参照してください。

コンテナインスタンスロールを作成する

> **Important**
>
> クラスターに外部インスタンスを登録する場合は、Amazon ECS Anywhere IAM ロール を参照してください。

機能や機能強化が今後導入されたときに Amazon ECS がそれらに対するアクセス許可を追加できるように、手動でロールを作成し、その IAM 管理ポリシーをコンテナインスタンスにアタッチできます。必要に応じて、以下の手順を使用してマネージド IAM ポリシーをアタッチします。

AWS マネジメントコンソール

Elastic Container Service のサービスロールを作成するには (IAM コンソール)

1. AWS マネジメントコンソール にサインインして、IAM コンソール https://console.aws.amazon.com/iam/ を開きます。
2. IAM コンソールのナビゲーションペインで、[ロール]、[ロールを作成] を選択します。
3. 信頼できるエンティティタイプ で、AWS のサービス を選択します。
4. [サービスまたはユースケース] で [エラスティックコンテナサービス] を選択し、次に [エラスティックコンテナサービスのユースケースの EC2 ロール] を選択します。
5. [次へ] を選択します。
6. [アクセス許可ポリシー] セクションで、[AmazonEC2ContainerServiceforEC2Role] ポリシーが選択されていることを確認します。

   > **Important**
   >
   > AmazonEC2ContainerServiceforEC2Role 管理ポリシーは、コンテナインスタンス IAM ロールにアタッチする必要があります。アタッチされていない場合、AWS マネジメントコンソール をクリックして、クラスターを作成します。

7. [次へ] を選択します。
8. [ロール名] には、[ecsInstanceRole] と入力します。
9. ロールを確認したら、[ロールを作成] を選択します。

AWS CLI

すべての######を自分の値に置き換えてください。

1. instance-role-trust-policy.json というファイルを次の内容で作成します。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": { "Service": "ec2.amazonaws.com"},
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. 信頼ポリシードキュメントを使用してインスタンスの IAM ロールを作成するには、次のコマンドを使用します。

```
aws iam create-role \
    --role-name ecsInstanceRole \
    --assume-role-policy-document file://instance-role-trust-policy.json
```

3. create-instance-profile コマンドを使用して、ecsInstanceRole-profile という名前のインスタンスプロファイルを作成します。

```
aws iam create-instance-profile --instance-profile-name ecsInstanceRole-profile
```

レスポンスの例

```
{
    "InstanceProfile": {
        "InstanceProfileId": "AIPAJTLBPJLEGREXAMPLE",
        "Roles": [],
        "CreateDate": "2022-04-12T23:53:34.093Z",
        "InstanceProfileName": "ecsInstanceRole-profile",
        "Path": "/",
        "Arn": "arn:aws:iam::123456789012:instance-profile/ecsInstanceRole-
profile"
    }
```

```
}
```

4. *ecsInstanceRole* インスタンスプロファイルに *ecsInstanceRole-profile* ロールを追加します。

```
aws iam add-role-to-instance-profile \
    --instance-profile-name ecsInstanceRole-profile \
    --role-name ecsInstanceRole
```

5. 次のコマンドを使用して、AmazonEC2ContainerServiceForEC2Role マネージドポリシーをロールにアタッチします。

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonEC2ContainerServiceforEC2Role \
    --role-name ecsInstanceRole
```

ロールを作成したら、次の機能のアクセス許可をロールに追加します。

機能	追加のアクセス許可
Amazon ECR でコンテナイメージを取得する	Amazon ECR のアクセス許可
CloudWatch Logs でコンテナインスタンスをモニタリングする	コンテナインスタンスのモニタリングに必要なアクセス許可
設定ファイルを Amazon S3 バケットでホストする	Amazon S3 への読み取り専用アクセス

Amazon ECR のアクセス許可

コンテナインスタンスで使用する Amazon ECS コンテナインスタンスロールには、Amazon ECR に対する以下の IAM ポリシーのアクセス許可が必要です。

JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        }
    ]
}
```

コンテナインスタンスに AmazonEC2ContainerServiceforEC2Role 管理ポリシーを使用すると、ロールに適切なアクセス権限が付与されます。

awsvpcTrunking アカウント設定の設定に必要なアクセス許可

Amazon ECS は、サポートされている Amazon EC2 インスタンスタイプを使用して、ENI 密度が高いコンテナインスタンスの起動をサポートします。この機能を使用する場合は、2 つのコンテナインスタンスロールを作成することをお勧めします。1 つのロールで awsvpcTrunking アカウント設定を有効にし、そのロールを ENI トランキングを必要とするタスクに使用します。awsvpcTrunking アカウント設定については、「アカウント設定による Amazon ECS 機能へのアクセス」を参照してください。

コンテナインスタンスで使用するコンテナインスタンスロールには、アカウント設定を設定するための次の IAM ポリシーのアクセス許可が必要です

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:ListAccountSettings",
                "ecs:ListAttributes",
                "ecs:PutAccountSetting"
            ],
```

```
            "Resource": "*"
        }
    ]
}
```

コンテナインスタンスロールを使用するには、インスタンスユーザーデータに以下を追加します。

```
#!/bin/bash
aws ecs put-account-setting --name awsvpcTrunking --value enabled --region region
ECS_CLUSTER=MyCluster>> /etc/ecs/ecs.config
EOF
```

EC2 インスタンスへのユーザーデータの追加の詳細については、「Amazon EC2 ユーザーガイド」の「起動時に Linux インスタンスでコマンドを実行する」を参照してください。

Amazon S3 への読み取り専用アクセス

設定情報を Amazon S3 のプライベートバケットに保存し、コンテナインスタンスの IAM ロールに読み取り専用アクセス権限を付与するのが、コンテナインスタンスの起動時に設定を許可する安全で便利な方法です。`ecs.config` ファイルのコピーをプライベートバケットに保存し、Amazon EC2 ユーザーデータを使用して AWS CLI をインストールします。その後、インスタンスの起動時に設定情報を `/etc/ecs/ecs.config` にコピーします。

`ecs.config` ファイルの作成と Amazon S3 への保存、およびこの構成を使用したインスタンスの起動の詳細については、「Amazon S3 に Amazon ECS コンテナインスタンスの設定を保存する」を参照してください。

次の AWS CLI コマンドを使用すると、Amazon S3 の読み取り専用アクセス許可をコンテナインスタンスのロールに許可できます。前に作成したロールの名前で *ecsInstanceRole* を置き換えます。

```
aws iam attach-role-policy \
      --role-name ecsInstanceRole \
      --policy-arn arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess
```

IAM コンソールを使用して、Amazon S3 の読み取り専用アクセス許可 (`AmazonS3ReadOnlyAccess`) をロールに追加することもできます。詳細については、「AWS Identity and Access Management ユーザーガイド」の「ロールに対するアクセス許可を更新する」を参照してください。

コンテナインスタンスのモニタリングに必要なアクセス許可

コンテナインスタンスが CloudWatch Logs にログデータを送信する前に、Amazon ECS エージェントがお客様のアプリケーションログを CloudWatch に書き込むことを許可する IAM ポリシーを作成する必要があります (通常は `awslogs` ドライバーを介して処理されます)。作成が完了したポリシーは、`ecsInstanceRole` にアタッチします。

AWS マネジメントコンソール

JSON ポリシーエディタでポリシーを作成するには

1. AWS マネジメントコンソール にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左側のナビゲーションペインで、[ポリシー] を選択します。

   初めて [ポリシー] を選択する場合には、[管理ポリシーにようこそ] ページが表示されます。今すぐ始める を選択します。
3. ページの上部で、[ポリシーを作成] を選択します。
4. ポリシーエディタ セクションで、JSON オプションを選択します。
5. 次の JSON ポリシードキュメントを入力します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:DescribeLogStreams"
            ],
            "Resource": ["arn:aws:logs:*:*:*"]
        }
    ]
}
```

6. 次へ を選択します。

> **Note**
>
> いつでも Visual と JSON エディタオプションを切り替えることができます。ただし、[ビジュアル] エディタで [次へ] に変更または選択した場合、IAM はポリシーを再構成してビジュアルエディタに合わせて最適化することがあります。詳細については、IAM ユーザーガイド の ポリシーの再構成 を参照してください。

7. 確認と作成 ページで、作成するポリシーの ポリシー名 と 説明 (オプション) を入力します。このポリシーで定義されているアクセス許可 を確認して、ポリシーによって付与されたアクセス許可を確認します。
8. ポリシーを作成 をクリックして、新しいポリシーを保存します。

ポリシーを作成したら、そのポリシーをコンテナインスタンスロールにアタッチします。ポリシーをロールにアタッチする方法については、「AWS Identity and Access Management ユーザーガイド」の「ロールに対するアクセス許可を更新する」を参照してください。

AWS CLI

1. `instance-cw-logs.json` というファイルを次の内容で作成します。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:DescribeLogStreams"
            ],
            "Resource": ["arn:aws:logs:*:*:*"]
        }
    ]
}
```

2. JSON ポリシードキュメントファイルを使用して IAM ポリシーを作成するには、次のコマンドを使用します。

```
aws iam create-policy \
      --policy-name cwlogspolicy \
      --policy-document file://instance-cw-logs.json
```

3. 次のコマンドを使用して、作成した IAM ポリシーの ARN を取得します。*cwlogspolicy* は、作成したポリシーの名前に置き換えます。

```
aws iam list-policies --scope Local --query 'Policies[?
PolicyName==`cwlogspolicy`].Arn'
```

4. ポリシー ARN を使用してコンテナインスタンスの IAM ロールにポリシーをアタッチするには、次のコマンドを使用します。

```
aws iam attach-role-policy \
      --role-name ecsInstanceRole \
      --policy-arn arn:aws:iam:111122223333:aws:policy/cwlogspolicy
```

Amazon ECS Anywhere IAM ロール

オンプレミスサーバーまたは仮想マシン (VM) をクラスタ-に登録する場合、サーバーまたは VM は AWS APIと通信するために IAM ロールを必要とします。IAM ロールの作成は AWS アカウントごとに一度のみ行う必要があります。ただし、この IAM ロールは、クラスターに登録する各サーバーまたは VM に関連付ける必要があります。このロールはECSAnywhereRole。このロールを手動で作成することもできます。または、Amazon ECS は、AWS マネジメントコンソール に外部インスタンスを登録するときに、ユーザーに代わってロールを作成することもできます。IAM コンソールの検索を使用して ecsAnywhereRole を検索すると、アカウントにすでにそのロールがあるかどうかを確認できます。詳細については、「IAM ユーザーガイド」の「IAM コンソールの検索」を参照してください。

AWSには、ECS Anywhere IAM ロールの作成時に使用できる 2 つの管理対象 IAM ポリシー、AmazonSSMManagedInstanceCoreおよびAmazonEC2ContainerServiceforEC2Roleポリシーが用意されています。AmazonEC2ContainerServiceforEC2Roleポリシーには、必要以上に多くのアクセスを提供するアクセス許可が含まれています。したがって、特定のユースケースに応じて、必要なポリシーからのアクセス許可のみを追加するカスタムポリシーを作成することをお勧めします。詳細については、「Amazon ECS コンテナインスタンスの IAM ロール」を参照してください。

タスク実行 IAM ロールは、ユーザーに代わって AWS API コールを実行するアクセス許可を Amazon ECS コンテナエージェントに付与します。タスク実行 IAM ロールを使用する場合は、タスク定義で指定する必要があります。詳細については、「Amazon ECS タスク実行IAM ロール」を参照してください。

次のいずれかの条件が適用される場合は、タスクの実行ロールが必要です。

- コンテナログを CloudWatch Logs に送信するには、awslogsログドライバーを使用します。
- タスク定義では、Amazon ECR プライベートリポジトリでホストされるコンテナイメージを指定します。ただし、外部インスタンスに関連付けられている ECSAnywhereRole ロールに、Amazon ECR からイメージをプルするために必要なアクセス許可が含まれている場合は、タスク実行ロールにそれらを含める必要はありません。

Amazon ECS Anywhere ロールを作成する

すべての *[######]* は、お客様の情報で置き換えてください。

1. 以下のポリシーを持つ、ssm-trust-policy.json という名前のローカルファイルを作成します。

   JSON

   ```
   {
     "Version":"2012-10-17",
     "Statement": {
       "Effect": "Allow",
       "Principal": {"Service": [
         "ssm.amazonaws.com"
       ]},
       "Action": "sts:AssumeRole"
     }
   }
   ```

2. 次の AWS CLI コマンドを使用して、ロールを作成し、信頼ポリシーをアタッチします。

   ```
   aws iam create-role --role-name ecsAnywhereRole --assume-role-policy-document
    file://ssm-trust-policy.json
   ```

3. 次のコマンドを使用して AWS マネージドポリシーをアタッチします。

```
aws iam attach-role-policy --role-name ecsAnywhereRole --policy-arn
 arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
aws iam attach-role-policy --role-name ecsAnywhereRole --policy-arn
 arn:aws:iam::aws:policy/service-role/AmazonEC2ContainerServiceforEC2Role
```

IAM のカスタム信頼ポリシーワークフローを使用してロールを作成することもできます。詳細については、「IAM ユーザーガイド」の「カスタム信頼ポリシーを使用してロールを作成する (コンソール)」を参照してください。

Amazon ECS インフラストラクチャ IAM ロール

Amazon ECS インフラストラクチャ IAM ロールを使用すると、Amazon ECS はユーザーに代わってクラスター内のインフラストラクチャリソースを管理できます。次の場合に使用します。

- Amazon EBS ボリュームを Fargate または EC2 起動タイプの Amazon ECS タスクにアタッチできます。インフラストラクチャロールにより、Amazon ECS はタスクの Amazon EBS ボリュームを管理できます。

  `AmazonECSInfrastructureRolePolicyForVolumes` 管理ポリシーを使用できます。
- Amazon ECS Service Connect サービス間のトラフィックを暗号化するには、Transport Layer Security (TLS) を使用します。

  `AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecurity` 管理ポリシーを使用できます。
- Amazon VPC Lattice ターゲットグループを作成する場合。

  `AmazonECSInfrastructureRolePolicyForVpcLattice` 管理ポリシーを使用できます。
- Amazon ECS クラスターで Amazon ECS マネージドインスタンスを使用します。インフラストラクチャロールにより、Amazon ECS はマネージドインスタンスのライフサイクルを管理できます。

  `AmazonECSInfrastructureRolePolicyForManagedInstances` 管理ポリシーを使用できます。
- Express Mode の使用を希望する場合。インフラストラクチャロールは、Amazon ECS が Express Mode サービスに必要なインフラストラクチャコンポーネント (ロードバランシング、セキュリティグループ、SSL 証明書、自動スケーリング設定など) のプロビジョニングと管理を行えるようにします。

AmazonECSInfrastructureRoleforExpressGatewayServices 管理ポリシーを使用できます。

Amazon ECS がこの役割を引き受け、ユーザーに代わってアクションを実行すると、イベントが AWS CloudTrail に表示されます。Amazon ECS がそのロールを使用してタスクにアタッチされた Amazon EBS ボリュームを管理する場合、CloudTrail ログ roleSessionName は ECSTaskVolumesForEBS となります。ロールを使用して Service Connect サービス間のトラフィックを暗号化する場合、CloudTrail ログ roleSessionName は ECSServiceConnectForTLS になります。ロールを使用して VPC Lattice のターゲットグループを作成する場合、CloudTrail ログ roleSessionName は ECSNetworkingWithVPCLattice になります。ロールを使用して Amazon ECS マネージドインスタンスを管理する場合、CloudTrail ログ roleSessionName は ECSManagedInstancesForCompute になります。この名前を使用して、[ユーザー名]でフィルタリングすることで CloudTrail コンソールでイベントを検索できます。

Amazon ECS では、ボリュームのアタッチメント、TLS、VPC Lattice、マネージドインスタンスに必要なアクセス許可を含むマネージドポリシーを提供しています。詳細については、「AWS マネージドポリシーのリファレンスガイド」の「AmazonECSInfrastructureRolePolicyForVolumes」、「AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecurity」、「AmazonECSInfrastructureRolePolicyForVpcLattice」、「AmazonECSInfrastructureRolePolicyForManagedInstances」、および「AmazonECSInfrastructureRoleforExpressGatewayServices」を参照してください。

Amazon ECS インフラストラクチャロールを作成する

すべての *[######]* は、お客様の情報で置き換えてください。

1. IAM ロールに使用する信頼ポリシーが含まれている ecs-infrastructure-trust-policy.json という名前のファイルを作成します。ファイルには次の内容が含まれます。

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAccessToECSForInfrastructureManagement",
      "Effect": "Allow",
      "Principal": {
```

```
        "Service": "ecs.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. 前のステップで作成した信頼ポリシーを使用して、ecsInfrastructureRole という名前のロールを作成するには、次の AWS CLI コマンドを使用します。

```
aws iam create-role \
      --role-name ecsInfrastructureRole \
      --assume-role-policy-document file://ecs-infrastructure-trust-policy.json
```

3. ユースケースに応じて、マネージドポリシーを ecsInfrastructureRole ロールにアタッチします。

   - Amazon EBS ボリュームを Fargate または EC2 起動タイプの Amazon ECS タスクにアタッチするには、AmazonECSInfrastructureRolePolicyForVolumes マネージドポリシーをアタッチします。
   - Transport Layer Security (TLS) を使用して Amazon ECS Service Connect サービス間のトラフィックを暗号化するには、AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecurity マネージドポリシーをアタッチします。
   - Amazon VPC Lattice ターゲットグループを作成するには、AmazonECSInfrastructureRolePolicyForVpcLattice マネージドポリシーをアタッチします。
   - Amazon ECS クラスターで Amazon ECS マネージドインスタンスを使用するには、AmazonECSInfrastructureRolePolicyForManagedInstances マネージドポリシーをアタッチします。

```
aws iam attach-role-policy \
      --role-name ecsInfrastructureRole \
      --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonECSInfrastructureRolePolicyForVolumes
```

```
aws iam attach-role-policy \
      --role-name ecsInfrastructureRole \
```

```
      --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecurity
```

```
aws iam attach-role-policy \
      --role-name ecsInfrastructureRole \
      --policy-arn arn:aws:iam::aws:policy/
AmazonECSInfrastructureRolePolicyForManagedInstances
```

IAM コンソールの[カスタム信頼ポリシー]ワークフローを使用してロールを作成することもできます。詳細については、「IAM ユーザーガイド」の「カスタム信頼ポリシーを使用してロールを作成する (コンソール)」を参照してください。

> **Important**
>
> Amazon ECS がタスクにアタッチされた Amazon EBS ボリュームを管理するためにインフラストラクチャロールを使用している場合は、Amazon EBS ボリュームを使用するタスクを停止する前に、次の点を確認してください。
>
> - ロールが削除されていません。
> - ロールの信頼ポリシーは、Amazon ECS アクセス (`ecs.amazonaws.com`) を削除するように変更されていません。
> - マネージドポリシー `AmazonECSInfrastructureRolePolicyForVolumes` は削除されていません。ロールのアクセス許可を変更する必要がある場合は、ボリュームを削除するために、少なくとも `ec2:DetachVolume`、`ec2:DeleteVolume`、`ec2:DescribeVolumes` を残してください。
>
> Amazon EBS ボリュームがアタッチされたタスクを停止する前にロールを削除または変更すると、タスクが `DEPROVISIONING` で停止し、関連する Amazon EBS ボリュームは削除されません。Amazon ECS は、必要なアクセス許可が回復するまで、定期的に自動的に再試行してタスクを停止し、ボリュームを削除します。DescribeTasks API を使用して、タスクのボリュームアタッチ状態と関連するステータス理由を表示できます。

ファイルを作成したら、Amazon ECS にロールを渡すためのアクセス許可をユーザーに付与する必要があります。

インフラストラクチャロールを Amazon ECS に渡すためのアクセス許可

ECS インフラストラクチャの IAM ロールを使用するには、そのロールを Amazon ECS に渡すためのユーザー権限を付与する必要があります。ユーザーに、次の `iam:PassRole` 権限をアタッチします。前に作成したインフラストラクチャロールの名前で *`ecsInfrastructureRole`* を置き換えます。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [

        {
            "Action": "iam:PassRole",
            "Effect": "Allow",
            "Resource": ["arn:aws:iam::*:role/ecsInfrastructureRole"],
            "Condition": {
                "StringEquals": {"iam:PassedToService": "ecs.amazonaws.com"}
            }
        }
    ]
}
```

`iam:Passrole` およびユーザー権限の更新の詳細については、「AWS Identity and Access Management ユーザーガイド」の「AWS サービスにロールを渡すアクセス許可をユーザーに付与する」と「IAM ユーザーのアクセス許可を変更する」を参照してください。

Amazon ECS マネージドインスタンスのインスタンスプロファイル

インスタンスプロファイルは、1 つの IAM ロールのみを保持し、Amazon ECS マネージドインスタンスがそのロールを安全に引き受けることを許可する IAM コンテナです。インスタンスプロファイルには、クラスターにインスタンスを登録し、ECS サービスと通信するために ECS エージェントが引き受けるインスタンスロールが含まれています。

> **Important**
>
> `AmazonECSInfrastructureRolePolicyForManagedInstances` マネージドポリシーで Amazon ECS マネージドインスタンスを使用する場合、インスタンスロール名

は ecsInstanceRole で始まる必要があります。ポリシーの範囲は iam:PassRole から arn:aws:iam::*:role/ecsInstanceRole* であるため、名前が一致しない場合、タスクの起動時に認証エラーが発生します。CloudFormation は MyStack-InstanceRole-ABC123 のような名前を自動生成するため、これは RoleName を省略した場合に CloudFormation でよく発生します。
代わりにカスタムインフラストラクチャロールポリシーを使用する場合、ポリシーにインスタンスロール ARN をターゲットとする iam:PassRole の付与が含まれている限り、インスタンスロールには任意の名前を付けることができます。

信頼ポリシーでロールを作成する

すべての *[######]* は、お客様の情報で置き換えてください。

1. IAM ロールに使用する信頼ポリシーが含まれている ecsInstanceRole-trust-policy.json という名前のファイルを作成します。ファイルには次の内容が含まれます。

   JSON

   ```
   {
     "Version":"2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": { "Service": "ec2.amazonaws.com"},
         "Action": "sts:AssumeRole"
       }
     ]
   }
   ```

2. 前のステップで作成した信頼ポリシーを使用して、ecsInstanceRole という名前のロールを作成するには、次の AWS CLI コマンドを使用します。

   ```
   aws iam create-role \
         --role-name ecsInstanceRole \
         --assume-role-policy-document file://ecsInstanceRole-trust-policy.json
   ```

3. AWS 管理ポリシー AmazonECSInstanceRolePolicyForManagedInstances を ecsInstanceRole ロールにアタッチします。

```
aws iam attach-role-policy \
      --role-name ecsInstanceRole \
      --policy-arn arn:aws:iam::aws:policy/
AmazonECSInstanceRolePolicyForManagedInstances
```

> **Note**
>
> その代わりにアクセスの最小特権を適用し、独自のアクセス許可を指定する場合は、次のアクセス権限を追加して、Amazon ECS マネージドインスタンスのタスク関連の問題のトラブルシューティングに役立てることができます。
>
> - `ecs:StartTelemetrySession`
> - `ecs:PutSystemLogEvents`

IAM コンソールの[カスタム信頼ポリシー]ワークフローを使用してロールを作成することもできます。詳細については、「IAM ユーザーガイド」の「カスタム信頼ポリシーを使用してロールを作成する (コンソール)」を参照してください。

ファイルを作成したら、Amazon ECS にロールを渡すためのアクセス許可をユーザーに付与する必要があります。

AWS CLI を使用してインスタンスプロファイルを作成する

ロールを作成したら、AWS CLI を使用してインスタンスプロファイルを作成します。

```
aws iam create-instance-profile --instance-profile-name ecsInstanceRole
```

次のように、インスタンスプロファイルにロールを追加します。

```
aws iam add-role-to-instance-profile \
   --instance-profile-name ecsInstanceRole \
   --role-name ecsInstanceRole
```

プロファイルが正常に作成されたことを確認します。

```
aws iam get-instance-profile --instance-profile-name ecsInstanceRole
```

CloudFormation を使用してインスタンスプロファイルを作成する

AWS CloudFormation を使用すると、インスタンスロールとインスタンスプロファイルを作成できます。AWS マネージドインフラストラクチャポリシーとカスタムポリシーのどちらを使用するかに基づいて、次のいずれかのオプションを選択します。

オプション 1: ecsInstanceRole 命名規則を使用する (推奨)

AWS マネージドインフラストラクチャポリシーを使用する場合は、`RoleName` を `ecsInstanceRole` で始まる値に明示的に設定する必要があります。`RoleName` を省略すると、CloudFormation はマネージドポリシーの `iam:PassRole` 条件に一致しない名前を自動生成し、タスクは起動に失敗します。

```
Resources:
  EcsInstanceRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: ecsInstanceRole
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Principal:
              Service: ec2.amazonaws.com
            Action: sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/AmazonECSInstanceRolePolicyForManagedInstances

  EcsInstanceProfile:
    Type: AWS::IAM::InstanceProfile
    Properties:
      InstanceProfileName: ecsInstanceRole
      Roles:
        - !Ref EcsInstanceRole
```

オプション 2: カスタムロール名を使用する

CloudFormation でロール名を生成する場合、または `ecsInstanceRole` で始まらないカスタム名を使用する場合は、`iam:PassRole` をインスタンスロールに付与するインフラストラクチャロールにインラインポリシーを追加する必要があります。

```
Resources:
```

```
  EcsInstanceRole:
    Type: AWS::IAM::Role
    Properties:
      # No RoleName — CFN auto-generates
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Principal:
              Service: ec2.amazonaws.com
            Action: sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/AmazonECSInstanceRolePolicyForManagedInstances

  EcsInstanceProfile:
    Type: AWS::IAM::InstanceProfile
    Properties:
      Roles:
        - !Ref EcsInstanceRole

  EcsInfrastructureRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Principal:
              Service: ecs.amazonaws.com
            Action: sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/AmazonECSInfrastructureRolePolicyForManagedInstances
      Policies:
        - PolicyName: PassInstanceRoleToEC2
          PolicyDocument:
            Version: "2012-10-17"
            Statement:
              - Effect: Allow
                Action: iam:PassRole
                Resource: !GetAtt EcsInstanceRole.Arn
                Condition:
                  StringLike:
                    iam:PassedToService: "ec2.*"
```

トラブルシューティング

タスクが iam:PassRole 認証エラーで失敗する

タスクが `iam:PassRole` に言及している `ResourceInitializationError` で失敗する場合は、インスタンスロール名が `ecsInstanceRole` で始まることを確認します。自動生成された名前は、スタックの [リソース] タブの CloudFormation コンソールで確認できます。名前が一致しない場合は、次のいずれかを実行します:

- `AWS::IAM::Role` リソースに `RoleName: ecsInstanceRole` を追加します。
- インフラストラクチャロールに明示的な `iam:PassRole` インラインポリシーを追加します。詳細については、「the section called “オプション 2: カスタムロール名を使用する”」を参照してください。

ロードバランサー用の Amazon ECS インフラストラクチャの IAM ロール

ロードバランサー用 Amazon ECS インフラストラクチャの IAM ロールは、Amazon ECS がユーザーに代わって、クラスター内のロードバランサーリソースを管理できるようにするためのものであり、次の場合に使用します。

- Amazon ECS でブルー/グリーンデプロイを使用したい場合。インフラストラクチャロールにより、Amazon ECS でデプロイのロードバランサーリソースを管理できるようになります。
- デプロイ操作中に、Amazon ECS でターゲットグループやリスナーなどのロードバランサーリソースを作成、変更、または削除する必要がある場合。

Amazon ECS がこの役割を引き受け、ユーザーに代わってアクションを実行すると、イベントが AWS CloudTrail に表示されます。Amazon ECS がこのロールを使用してブルー/グリーンデプロイのロードバランサーリソースを管理する場合、CloudTrail ログ `roleSessionName` は、`ECSNetworkingWithELB` または `ecs-service-scheduler` になります。この名前を使用して、[ユーザー名]でフィルタリングすることで CloudTrail コンソールでイベントを検索できます。

Amazon ECS は、ロードバランサー管理に必要となるアクセス許可を含む管理ポリシーを提供します。詳細については、「AWS マネージドポリシーリファレンスガイド」の「AmazonECSInfrastructureRolePolicyForLoadBalancers」を参照してください。

ロードバランサー用の Amazon ECS インフラストラクチャロールの作成

すべての *[######]* は、お客様の情報で置き換えてください。

1. IAM ロールに使用する信頼ポリシーが含まれている ecs-infrastructure-trust-policy.json という名前のファイルを作成します。ファイルには次の内容が含まれます。

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAccessToECSForInfrastructureManagement",
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. 前のステップで作成した信頼ポリシーを使用して、ecsInfrastructureRoleForLoadBalancers という名前のロールを作成するには、次の AWS CLI コマンドを使用します。

```
aws iam create-role \
      --role-name ecsInfrastructureRoleForLoadBalancers \
      --assume-role-policy-document file://ecs-infrastructure-trust-policy.json
```

3. AWS 管理ポリシー AmazonECSInfrastructureRolePolicyForLoadBalancers を ecsInfrastructureRoleForLoadBalancers ロールにアタッチします。

```
aws iam attach-role-policy \
      --role-name ecsInfrastructureRoleForLoadBalancers \
      --policy-arn arn:aws:iam::aws:policy/
AmazonECSInfrastructureRolePolicyForLoadBalancers
```

IAM コンソールの[カスタム信頼ポリシー]ワークフローを使用してロールを作成することもできます。詳細については、「IAM ユーザーガイド」の「カスタム信頼ポリシーを使用してロールを作成する (コンソール)」を参照してください。

Important

Amazon ECS がブルー/グリーンデプロイのロードバランサーリソースを管理するためにインフラストラクチャロールを使用している場合は、ロールを削除または変更する前に以下の点を確認してください。

- アクティブなデプロイの進行中に、ロールが削除されない。
- ロールの信頼ポリシーは、Amazon ECS アクセス (`ecs.amazonaws.com`) を削除するように変更されていません。
- アクティブなデプロイの進行中に、管理ポリシー `AmazonECSInfrastructureRolePolicyForLoadBalancers` が削除されない。

ブルー/グリーンデプロイ中にロールを削除または変更すると、デプロイが失敗し、サービスが不整合な状態になる可能性があります。

ファイルを作成したら、Amazon ECS にロールを渡すためのアクセス許可をユーザーに付与する必要があります。

インフラストラクチャロールを Amazon ECS に渡すためのアクセス許可

ロードバランサー用の ECS インフラストラクチャの IAM ロールを使用するには、そのロールを Amazon ECS に渡すためのユーザー権限を付与する必要があります。ユーザーに、次の `iam:PassRole` 権限をアタッチします。*`ecsInfrastructureRoleForLoadBalancers`* を作成したインフラストラクチャロールの名前に置き換えます。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": "iam:PassRole",
            "Effect": "Allow",
            "Resource":
 ["arn:aws:iam::*:role/ecsInfrastructureRoleForLoadBalancers"],
            "Condition": {
                "StringEquals": {"iam:PassedToService": "ecs.amazonaws.com"}
            }
```

```
        }
    ]
}
```

`iam:Passrole` およびユーザー権限の更新の詳細については、「AWS Identity and Access Management ユーザーガイド」の「AWS サービスにロールを渡すアクセス許可をユーザーに付与する」と「IAM ユーザーのアクセス許可を変更する」を参照してください。

Amazon ECS CodeDeploy IAM ロール

Amazon ECS で CodeDeploy ブルー/グリーンデプロイタイプを使用するには、ユーザーの代わりに Amazon ECS サービスを更新するためのアクセス許可を事前に CodeDeploy サービスに付与しておく必要があります。これらのアクセス権限は、CodeDeploy IAM ロール (`ecsCodeDeployRole`) によって付与されます。

> **Note**
>
> ユーザーには CodeDeploy を使用するアクセス許可も必要です。これらの権限については、「必要な IAM 許可」で説明しています。

2 つの管理ポリシーが用意されています。詳細については、「AWS マネージドポリシーリファレンスガイド」で、次のいずれかを参照してください。

- AWSCodeDeployRoleForECS - 関連付けられたアクションを使用して、リソースを更新するためのアクセス許可を CodeDeploy に付与します。
- AWSCodeDeployRoleForECSLimited - より制限されたアクセス許可を CodeDeploy に付与します。

CodeDeploy ロールの作成

以下の手順を使用して、Amazon ECS 用 CodeDeploy ロールを作成できます。

AWS マネジメントコンソール

CodeDeploy のサービスロールを作成するには (IAM コンソール)

1. AWS マネジメントコンソール にサインインして、IAM コンソール https://console.aws.amazon.com/iam/ を開きます。

2. IAM コンソールのナビゲーションペインで、[ロール]、[ロールを作成] を選択します。
3. 信頼できるエンティティタイプ で、AWS のサービス を選択します。
4. [サービスまたはユースケース] で [CodeDeploy] を選択し、次に [CodeDeploy - ECS] ユースケースを選択します。
5. [次へ] を選択します。
6. 「[アクセス許可ポリシーをアタッチする]」セクションで、[AWSCodeDeployRoleForECS] ポリシーが選択されていることを確認します。
7. [次へ] を選択します。
8. [ロール名] に [ECSodeDeployRole] と入力します。
9. ロールを確認したら、[ロールを作成] を選択します。

AWS CLI

すべての *[######]* は、お客様の情報で置き換えてください。

1. CodeDeploy IAM ロールに使用する信頼ポリシーを含む、`codedeploy-trust-policy.json` という名前のファイルを作成します。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": ["codedeploy.amazonaws.com"]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

2. 前のステップで作成した信頼ポリシーを使用した `ecsCodedeployRole` という名前の IAM ロールを作成します。

```
aws iam create-role \
```

```
      --role-name ecsCodedeployRole \
      --assume-role-policy-document file://codedeploy-trust-policy.json
```

3. AWSCodeDeployRoleForECS または AWSCodeDeployRoleForECSLimited マネージドポリシー を ecsTaskRole ロールにアタッチします。

```
aws iam attach-role-policy \
      --role-name ecsCodedeployRole \
      --policy-arn arn:aws:iam::aws:policy/AWSCodeDeployRoleForECS
```

```
aws iam attach-role-policy \
      --role-name ecsCodedeployRole \
      --policy-arn arn:aws:iam::aws:policy/AWSCodeDeployRoleForECSLimited
```

サービス内のタスクにタスク実行ロールが必要な場合は、各タスク実行ロールまたはタスクロールの上書きの iam:PassRole アクセス許可を、ポリシーとして CodeDeploy ロールに追加する必要があります。

タスク実行ロールのアクセス許可

サービス内のタスクにタスク実行ロールが必要な場合は、各タスク実行ロールまたはタスクロールの上書きの iam:PassRole アクセス許可を、ポリシーとして CodeDeploy ロールに追加する必要があります。詳細については、「Amazon ECS タスク実行IAM ロール」および「Amazon ECS タスクの IAM ロール」を参照してください。次に、そのポリシーを CodeDeploy ロールにアタッチします。

ポリシーの作成

AWS マネジメントコンソール

JSON ポリシーエディタでポリシーを作成するには

1. AWS マネジメントコンソール にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左側のナビゲーションペインで、[ポリシー] を選択します。

   初めて [ポリシー] を選択する場合には、[管理ポリシーにようこそ] ページが表示されます。今すぐ始める を選択します。
3. ページの上部で、[ポリシーを作成] を選択します。
4. ポリシーエディタ セクションで、JSON オプションを選択します。

5. 次の JSON ポリシードキュメントを入力します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": ["arn:aws:iam::<aws_account_id>:role/
<ecsCodeDeployRole>"]
        }
    ]
}
```

6. 次へ を選択します。

> **Note**
>
> いつでも Visual と JSON エディタオプションを切り替えることができます。ただし、[ビジュアル] エディタで [次へ] に変更または選択した場合、IAM はポリシーを再構成してビジュアルエディタに合わせて最適化することがあります。詳細については、IAM ユーザーガイド の ポリシーの再構成 を参照してください。

7. 確認と作成 ページで、作成するポリシーの ポリシー名 と 説明 (オプション) を入力します。このポリシーで定義されているアクセス許可 を確認して、ポリシーによって付与されたアクセス許可を確認します。
8. ポリシーを作成 をクリックして、新しいポリシーを保存します。

ポリシーを作成したら、そのポリシーを CodeDeploy ロールにアタッチします。ポリシーをロールにアタッチする方法については、「AWS Identity and Access Management ユーザーガイド」の「ロールに対するアクセス許可を更新する」を参照してください。

AWS CLI

すべての *[######]* は、お客様の情報で置き換えてください。

1. `blue-green-iam-passrole.json` というファイルを次の内容で作成します。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": ["arn:aws:iam::*:role/code-deploy-role"],
            "Condition": {
                    "StringEquals": {"iam:PassedToService":
 "ecs.amazonaws.com"}
            }
        }
    ]
}
```

2. JSON ポリシードキュメントファイルを使用して IAM ポリシーを作成するには、次のコマンドを使用します。

```
aws iam create-policy \
      --policy-name cdTaskExecutionPolicy \
      --policy-document file://blue-green-iam-passrole.json
```

3. 次のコマンドを使用して、作成した IAM ポリシーの ARN を取得します。

```
aws iam list-policies --scope Local --query 'Policies[?
PolicyName==`cdTaskExecutionPolicy`].Arn'
```

4. 次のコマンドを使用して、ポリシーを CodeDeploy IAM ロールにアタッチします。

```
aws iam attach-role-policy \
      --role-name ecsCodedeployRole \
      --policy-arn arn:aws:iam:111122223333:aws:policy/cdTaskExecutionPolicy
```

Amazon ECS EventBridge IAM ロール

Amazon ECS でスケジュールされたタスクを EventBridge のルールとターゲットで送信するには、ユーザーの代わりに Amazon ECS タスクを実行するためのアクセス許可が EventBridge サービスに

必要です。これらのアクセス許可は、EventBridge IAM ロール (`ecsEventsRole`) から付与されます。

ポリシー `AmazonEC2ContainerServiceEventsRole` は以下のとおりです。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["ecs:RunTask"],
            "Resource": ["*"]
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": ["*"],
            "Condition": {
                "StringLike": {"iam:PassedToService": "ecs-tasks.amazonaws.com"}
            }
        },
        {
            "Effect": "Allow",
            "Action": "ecs:TagResource",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "ecs:CreateAction": ["RunTask"]
                }
            }
        }
    ]
}
```

スケジュールされたタスクでタスク実行ロールの使用、タスクロール、またはタスクロール上書きが必要な場合、タスク実行ロール、タスクロール、またはタスクロール上書きごとに `iam:PassRole` アクセス許可を EventBridge IAM ロールに追加する必要があります。タスクの実行ロールの詳細については、「Amazon ECS タスク実行IAM ロール」を参照してください。

> **Note**
>
> タスク実行ロールまたはタスクロール上書きの完全 ARN を指定します。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [

 "arn:aws:iam::111122223333:role/ecsTaskExecutionRole_or_TaskRole_name"
            ]
        }
    ]
}
```

スケジュールされたタスクを設定するときに、AWS マネジメントコンソール が EventBridge ロールを作成するようにすることができます。詳細については、「Amazon EventBridge スケジューラを使用して Amazon ECS タスクをスケジュールする」を参照してください。

Amazon ECS EventBridge ロールを作成する

すべての *[######]* は、お客様の情報で置き換えてください。

1. IAM ロールに使用する信頼ポリシーが含まれている `eventbridge-trust-policy.json` という名前のファイルを作成します。ファイルには次の内容が含まれます。

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "",
```

```
      "Effect": "Allow",
      "Principal": {
        "Service": "events.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. 前のステップで作成した信頼ポリシーを使用して、ecsEventsRole という名前の IAM ロールを作成するには、次のコマンドを使用します。

```
aws iam create-role \
      --role-name ecsEventsRole \
      --assume-role-policy-document file://eventbridge-trust-policy.json
```

3. 次のコマンドを使用して、AWS マネージド AmazonEC2ContainerServiceEventsRole を ecsEventsRole ロールにアタッチします。

```
aws iam attach-role-policy \
      --role-name ecsEventsRole \
      --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonEC2ContainerServiceEventsRole
```

IAM コンソールの[カスタム信頼ポリシーワークフロー] (https://console.aws.amazon.com/iam/) を使用してロールを作成することもできます。詳細については、「IAM ユーザーガイド」の「カスタム信頼ポリシーを使用してロールを作成する (コンソール)」を参照してください。

ecsEventsRole ロールへのポリシーのアタッチ

以下の手順を使用して、タスク実行ロールのアクセス許可を EventBridge IAM ロールに追加できます。

AWS マネジメントコンソール

JSON ポリシーエディタでポリシーを作成するには

1. AWS マネジメントコンソール にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左側のナビゲーションペインで、[ポリシー] を選択します。

初めて [ポリシー] を選択する場合には、[管理ポリシーにようこそ] ページが表示されます。今すぐ始める を選択します。

3. ページの上部で、[ポリシーを作成] を選択します。
4. ポリシーエディタ セクションで、JSON オプションを選択します。
5. 次の JSON ポリシードキュメントを入力します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": ["arn:aws:iam::111122223333:role/
<ecsTaskExecutionRole_or_TaskRole_name>"]
        }
    ]
}
```

6. 次へ を選択します。

> **Note**
>
> いつでも Visual と JSON エディタオプションを切り替えることができます。ただし、[ビジュアル] エディタで [次へ] に変更または選択した場合、IAM はポリシーを再構成してビジュアルエディタに合わせて最適化することがあります。詳細については、IAM ユーザーガイド の ポリシーの再構成 を参照してください。

7. 確認と作成 ページで、作成するポリシーの ポリシー名 と 説明 (オプション) を入力します。このポリシーで定義されているアクセス許可 を確認して、ポリシーによって付与されたアクセス許可を確認します。
8. ポリシーを作成 をクリックして、新しいポリシーを保存します。

ポリシーを作成したら、そのポリシーを EventBridge ロールにアタッチします。ポリシーをロールにアタッチする方法については、「AWS Identity and Access Management ユーザーガイド」の「ロールに対するアクセス許可を更新する」を参照してください。

AWS CLI

すべての *[######]* は、お客様の情報で置き換えてください。

1. ev-iam-passrole.json というファイルを次の内容で作成します。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [

 "arn:aws:iam::111122223333:role/ecsTaskExecutionRole_or_TaskRole_name"
            ]
        }
    ]
}
```

2. JSON ポリシードキュメントファイルを使用して IAM ポリシーを作成するには、次の AWS CLI コマンドを使用します。

```
aws iam create-policy \
      --policy-name eventsTaskExecutionPolicy \
      --policy-document file://ev-iam-passrole.json
```

3. 次のコマンドを使用して、作成した IAM ポリシーの ARN を取得します。

```
aws iam list-policies --scope Local --query 'Policies[?
PolicyName==`eventsTaskExecutionPolicy`].Arn'
```

4. ポリシー ARN を使用して EventBridge の IAM ロールにポリシーをアタッチするには、次のコマンドを使用します。

```
aws iam attach-role-policy \
      --role-name ecsEventsRole \
      --policy-arn arn:aws:iam:111122223333:aws:policy/eventsTaskExecutionPolicy
```

Amazon ECS コンソールに必要なアクセス許可

最小権限の付与のベストプラクティスに従い、AmazonECS_FullAccess 管理ポリシーを、独自のカスタムポリシーを作成するためのテンプレートとして使用できます。これにより、特定の要件に基づいて、管理ポリシーに権限を追加し、管理ポリシーから権限を追加、または権限を取り上げることができます。詳細については、「AWS マネージドポリシーリファレンスガイド」の「AmazonECS_FullAccess」を参照してください。

IAM ロールを作成するための許可

以下のアクションでは、操作を完了するために追加のアクセス許可が必要です。

- 外部インスタンスの登録 - 詳細は、Amazon ECS Anywhere IAM ロール を参照してください。
- タスク定義の登録 - 詳細は、Amazon ECS タスク実行IAM ロール を参照してください。
- タスクのスケジューリングに使用する EventBridge ルールの作成 - 詳細は、Amazon ECS EventBridge IAM ロール を参照してください。

Amazon ECS コンソールで使用する前に IAM でロールを作成することで、これらのアクセス許可を追加できます。ロールを作成しない場合、Amazon ECS コンソールがユーザーに代わってロールを作成します。

外部インスタンスをクラスターに登録するために必要なアクセス許可

外部インスタンスをクラスターに登録し、新しい外部インスタンス (ecsExternalInstanceRole) ロールを作成する場合は、追加のアクセス許可が必要です。

以下に示す追加のアクセス許可が必要です。

- iam — プリンシパルが IAM ロールとそれにアタッチするポリシーを作成し、一覧表示できるようにします。
  - iam:AttachRolePolicy
  - iam:CreateRole
  - am:CreateInstanceProfile
  - iam:AddRoleToInstanceProfile
  - iam:ListInstanceProfilesForRole
  - iam:GetRole

- ssm — プリンシパルが外部インスタンスを Systems Manager に登録できるようにします。

> **Note**
>
> 既存の ecsExternalInstanceRole を選択するには、iam:GetRole および iam:PassRole のアクセス許可が必要です。

次のポリシーには必要なアクセス許可が含まれており、アクションは ecsExternalInstanceRole ロールに限定されます。

JSON

```
{
"Statement": [
      {
          "Effect": "Allow",
          "Action": [
              "iam:AttachRolePolicy",
              "iam:CreateRole",
              "iam:CreateInstanceProfile",
              "iam:AddRoleToInstanceProfile",
              "iam:ListInstanceProfilesForRole",
              "iam:GetRole"
          ],
          "Resource": "arn:aws:iam::*:role/ecsExternalInstanceRole"
      },
      {
          "Effect": "Allow",
          "Action": ["iam:PassRole","ssm:CreateActivation"],
          "Resource": "arn:aws:iam::*:role/ecsExternalInstanceRole"
      }
    ]
}
```

タスク定義を登録するために必要なアクセス許可

タスク定義を登録し、新しいタスク実行 (ecsTaskExecutionRole) ロールを作成する場合は、追加のアクセス許可が必要です。

以下に示す追加のアクセス許可が必要です。

- `iam` — プリンシパルが IAM ロールとそれにアタッチするポリシーを作成し、一覧表示できるようにします。
  - iam:AttachRolePolicy
  - iam:CreateRole
  - iam:GetRole

> **Note**
>
> 既存の `ecsTaskExecutionRole` を選択するには、`iam:GetRole` のアクセス許可が必要です。

次のポリシーには必要なアクセス許可が含まれており、アクションは `ecsTaskExecutionRole` ロールに限定されます。

JSON

```
{
"Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "iam:AttachRolePolicy",
            "iam:CreateRole",
            "iam:GetRole"
        ],
        "Resource": "arn:aws:iam::*:role/ecsTaskExecutionRole"
        }
    ]
}
```

Amazon Q Developer を使用して、コンソールで推奨事項を提供するために必要なアクセス許可

Amazon Q Developer が Amazon ECS で推奨事項を提供するには、IAM ユーザーまたはロールの正しい IAM アクセス許可を有効にする必要があります。`codewhisperer:GenerateRecommendations` アクセス許可を追加する必要があります。

JSON

```
{
"Statement": [
      {
            "Sid": "AmazonQDeveloperPermissions",
            "Effect": "Allow",
            "Action": ["codewhisperer:GenerateRecommendations"],
            "Resource": "*"
        }
    ]
}
```

Amazon ECS コンソールでインラインチャットを使用するには、IAM ユーザーまたはロールの正しい IAM アクセス許可を有効にする必要があります。`q:SendMessage` アクセス許可を追加する必要があります。

JSON

```
{
"Statement": [
    {
        "Sid": "AmazonQDeveloperInlineChatPermissions",
        "Effect": "Allow",
        "Action": ["q:SendMessage"],
        "Resource": "*"
    }
  ]
}
```

スケジュールされたタスクの EventBridge ルールを作成するために必要なアクセス許可

タスクをスケジュールし、新しい CloudWatch Events ロール (ecsEventsRole) ロールを作成する場合は、追加のアクセス許可が必要です。

以下に示す追加のアクセス許可が必要です。

- iam — プリンシパルが IAM ロールとそれにアタッチするポリシーを作成して一覧表示し、Amazon ECS がそのロールを他のサービスに渡してロールを引き継ぐことができるようにします。

> **Note**
>
> 既存の ecsEventsRole を選択するには、iam:GetRole および iam:PassRole のアクセス許可が必要です。

次のポリシーには必要なアクセス許可が含まれており、アクションは ecsEventsRole ロールに限定されます。

JSON

```
{
 "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "iam:AttachRolePolicy",
            "iam:CreateRole",
            "iam:GetRole",
            "iam:PassRole"
        ],
        "Resource": "arn:aws:iam::*:role/ecsEventsRole"
        }
    ]
}
```

サービスデプロイを表示するために必要なアクセス許可

最小権限のベストプラクティスに従うと、コンソールでサービスデプロイを表示するには、アクセス許可を追加する必要があります。

次のアクションへのアクセスが必要です。

- ListServiceDeployments
- DescribeServiceDeployments
- DescribeServiceRevisions

次のリソースへのアクセスが必要です。

- サービス
- サービスデプロイ
- サービスリビジョン

次のポリシー例には必要なアクセス許可が含まれており、アクションは指定されたサービスに限定されます。

`account`、`cluster-name`、および `service-name` を独自の値に置き換えます。

JSON

```
{
"Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "ecs:ListServiceDeployments",
            "ecs:DescribeServiceDeployments",
            "ecs:DescribeServiceRevisions"
        ],
        "Resource": [
            "arn:aws:ecs:us-east-1:123456789012:service/cluster-name/service-
name",
            "arn:aws:ecs:us-east-1:123456789012:service-deployment/cluster-name/
service-name/*",
```

```
            "arn:aws:ecs:us-east-1:123456789012:service-revision/cluster-name/
service-name/*"
            ]
        }
    ]
}
```

Amazon ECS ライフサイクルイベントを Container Insights で表示するには、以下のアクセス許可が必要です。

ライフサイクルイベントを表示するには、以下のアクセス権限が必要です。次のアクセス許可をインラインポリシーとしてロールに追加します。詳細については、「IAM ポリシーの追加と削除」を参照してください。

- events:DescribeRule
- events:ListTargetsByRule
- logs:DescribeLogGroups

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "events:DescribeRule",
        "events:ListTargetsByRule",
        "logs:DescribeLogGroups"
      ],
      "Resource": "*"
    }
  ]
}
```

Amazon ECS ライフサイクルイベントを Container Insights で表示するために必要なアクセス許可

ライフサイクルイベントを設定するには、以下のアクセス権限が必要です。

- events:PutRule
- events:PutTargets
- logs:CreateLogGroup

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "events:PutRule",
        "events:PutTargets",
        "logs:CreateLogGroup"
      ],
      "Resource": "*"
    }
  ]
}
```

CloudFormation を使用する Amazon ECS コンソールに必要なアクセス許可

AWS マネジメントコンソール を使用してリソースを作成する前に、適切な IAM アクセス許可を持っていることを確認する必要があります。最初に Amazon ECS コンソールのアクセス許可を設定する方法については、「Amazon ECS コンソールに必要なアクセス許可」を参照してください。

Amazon ECS コンソールは AWS CloudFormation を利用しているため、以下の場合には追加の IAM アクセス許可が必要になります。

- クラスターの作成
- サービスの作成
- キャパシティープロバイダーの作成

追加のアクセス許可のポリシーを作成して、コンソールへのアクセスに使用する IAM ロールにアタッチできます。詳細については、「IAM ユーザーガイド」の「IAM ポリシーの作成」を参照してください。

クラスターを作成するために必要なアクセス許可

コンソールでクラスターを作成するときには、CloudFormation スタックを管理するための追加のアクセス許可が必要です。

以下に示す追加のアクセス許可が必要です。

- `cloudformation` — プリンシパルの作成と管理を許可する CloudFormation スタック。これは、AWS マネジメントコンソール を使用して Amazon ECS クラスターを作成し、それらのクラスターのその後の管理に必要です。
- `ssm` – CloudFormation が最新の Amazon ECS 最適化 AMI を参照できるようにします。これは、AWS マネジメントコンソール を使用して Amazon ECS クラスターを作成するときに必要です。

次のポリシーには必要な CloudFormation アクセス許可が含まれており、アクションは Amazon ECS コンソールで作成されたリソースに限定されます。

JSON

```
{
  "Statement": [
      {
          "Effect": "Allow",
          "Action": [
                "cloudformation:CreateStack",
                "cloudformation:DeleteStack",
                "cloudformation:DescribeStack*",
                "cloudformation:UpdateStack"
            ],
            "Resource": [
                "arn:*:cloudformation:*:*:stack/Infra-ECS-Cluster-*"
            ]
      },
      {
          "Effect": "Allow",
          "Action": "ssm:GetParameters",
```

```
          "Resource": [
            "arn:aws:ssm:*:*:parameter/aws/service/ecs/optimized-ami/amazon-
linux-2*/*",
            "arn:aws:ssm:*:*:parameter/aws/service/ecs/optimized-ami/amazon-
linux-2023*/*"
          ]
      }
   ]
}
```

Amazon ECS コンテナインスタンスロール (ecsInstanceRole) をまだ作成していない状態で Amazon EC2 インスタンスを使用するクラスターを作成する場合は、コンソールがユーザーに代わってロールを作成します。

さらに、Auto Scaling グループを使用する場合は、クラスターの自動スケーリング機能を使用する際にコンソールが Auto Scaling グループにタグを追加できるようにするための追加のアクセス許可が必要です。

以下に示す追加のアクセス許可が必要です。

- autoscaling — コンソールが Amazon EC2 Auto Scaling グループにタグ付けできるようにします。これは、クラスターのオートスケーリング機能を使用する場合、Amazon EC2 Auto Scaling グループを管理する場合に必要です。このタグは、ECS によって管理され、コンソールで作成されたことを示すためにコンソールによって自動的にグループに追加されます。
- iam — プリンシパルに IAM ロールとアタッチされたポリシーの一覧表示を許可します。プリンシパルは、Amazon EC2 インスタンスで利用できるインスタンスプロファイルを一覧表示することもできます。

次のポリシーには必要な IAM アクセス許可が含まれており、アクションは ecsInstanceRole ロールに限定されます。

自動スケーリング許可は制限されません。

JSON

```
{
  "Statement": [
      {
          "Effect": "Allow",
```

```
            "Action": [
              "iam:AttachRolePolicy",
              "iam:CreateRole",
              "iam:CreateInstanceProfile",
              "iam:AddRoleToInstanceProfile",
              "iam:ListInstanceProfilesForRole",
              "iam:GetRole"
            ],
            "Resource": "arn:aws:iam::*:role/ecsInstanceRole"
        },
        {
            "Effect": "Allow",
            "Action": "autoscaling:CreateOrUpdateTags",
            "Resource": "*"
        }
    ]
}
```

サービスを作成するために必要なアクセス許可

コンソールでサービスを作成するときには、CloudFormation スタックを管理するための追加のアクセス許可が必要です。以下に示す追加のアクセス許可が必要です。

- `cloudformation` — プリンシパルの作成と管理を許可する CloudFormation スタック。これは、AWS マネジメントコンソール を使用して Amazon ECS サービスを作成する際に、またその後それらのサービスを管理する際に必要です。

次のポリシーには必要なアクセス許可が含まれており、アクションは Amazon ECS コンソールで作成されたリソースに限定されます。

JSON

```
{
  "Statement": [
      {
          "Effect": "Allow",
          "Action": [
                "cloudformation:CreateStack",
                "cloudformation:DeleteStack",
                "cloudformation:DescribeStack*",
```

```
                "cloudformation:UpdateStack"
            ],
            "Resource": [
                "arn:*:cloudformation:*:*:stack/ECS-Console-V2-Service-*"
            ]
        }
    ]
}
```

Amazon ECS ブルー/グリーンデプロイの Lambda 関数に必要となるアクセス許可

Amazon ECS ブルー/グリーンデプロイでデプロイライフサイクルフックとして Lambda 関数を使用する場合は、特定のアクセス許可を持つ IAM ロールを作成する必要があります。このロールを使用することで、Amazon ECS はデプロイライフサイクルのさまざまな段階で Lambda 関数を呼び出すことができます。

以下に示す追加のアクセス許可が必要です。

- `lambda:InvokeFunction` – Amazon ECS がデプロイプロセス中にライフサイクルフックとして設定された Lambda 関数を呼び出すことができるようになります。

信頼ポリシーでは、サービスが次のロールを引き受けられるように許可する必要があります。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "ecs.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
```

```
}
```

Amazon ECS のサービス自動スケーリングに必要な IAM アクセス許可

Service Auto Scalingは、Amazon ECS、Amazon CloudWatch、およびアプリケーション Auto Scaling API の組み合わせによって可能になります。サービスは Amazon ECS で作成および更新され、アラームは CloudWatch で作成され、スケーリングポリシーは Application Auto Scaling で作成されます。

サービスの作成および更新のための標準の IAM アクセス許可に加えて、Service Auto Scaling 設定の操作には次のアクセス許可が必要です。

- `application-autoscaling:*`
- `ecs:DescribeServices`
- `ecs:UpdateService`
- `cloudwatch:DescribeAlarms`
- `cloudwatch:PutMetricAlarm`
- `cloudwatch:DeleteAlarms`
- `cloudwatch:DescribeAlarmHistory`
- `cloudwatch:DescribeAlarmsForMetric`
- `cloudwatch:GetMetricStatistics`
- `cloudwatch:ListMetrics`
- `cloudwatch:DisableAlarmActions`
- `cloudwatch:EnableAlarmActions`
- `iam:CreateServiceLinkedRole`
- `sns:CreateTopic`
- `sns:Subscribe`
- `sns:Get*`
- `sns:List*`

Application Auto Scaling サービスには、Amazon ECS サービスおよび CloudWatch アラームを記述するアクセス許可が必要です。また、ユーザーの代わりにサービスの必要タスク数を変更するアクセス許可も必要です。`sns:` 権限は、しきい値を超えたときに

CloudWatch が Amazon SNS トピックに送信する通知を対象とするものです。Amazon ECS サービスの自動スケーリングを使用にする場合、サービスにリンクされたロールが AWSServiceRoleForApplicationAutoScaling_ECSService という名前で作成されます。このサービスにリンクされたロールでは、ポリシーのアラームを記述し、サービスの現在実行中のタスクの数をモニタリングし、サービスの必要なタスクの数を変更するアクセス許可を Application Auto Scaling に付与します。Application Auto Scaling の元のマネージド型の Amazon ECS ロールは ecsAutoscaleRole ですが、これは不要になりました。サービスにリンクされたロールは、Application Auto Scaling のデフォルトロールです。詳細については、「Application Auto Scaling ユーザーガイド」の「Application Auto Scaling 用のサービスリンクロール」を参照してください。

CloudWatch メトリクスが Amazon ECS で使用可能になる前に Amazon ECS コンテナインスタンスロールを作成した場合は、このアクセス ecs:StartTelemetrySession 許可の追加が必要になることがあります。詳細については、「考慮事項」を参照してください。

リソース作成時にタグ付けするための許可を付与する

次の作成時のタグ付けの Amazon ECS API アクションを使用すると、リソースの作成時にタグを指定できます。リソース作成アクションでタグが指定されている場合、AWS は追加の認可を実行して、タグを作成するための正しい許可が割り当てられていることを検証します。

- CreateCapacityProvider
- CreateCluster
- CreateService
- CreateTaskSet
- RegisterContainerInstance
- RegisterTaskDefinition
- RunTask
- StartTask

リソースタグを使用して、属性ベースの制御 (ABAC) を実装できます。詳細については、「the section called “リソースタグを使用して Amazon ECS リソースへのアクセスを制御する”」および「リソースのタグ付け」を参照してください。

作成時のタグ付けを許可するには、ポリシーを作成または変更して、ecs:CreateCluster や ecs:RunTask などのリソースを作成するアクションと ecs:TagResource アクションを使用するための両方の許可を含めます。

次の例は、ユーザーがクラスターを作成し、クラスター作成時にタグを追加できるポリシーを示しています。ユーザーには既存のリソースへのタグ付けが許可されません (ecs:TagResource アクションを直接呼び出すことはできません)。

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
         "ecs:CreateCluster"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
         "ecs:TagResource"
      ],
      "Resource": "*",
      "Condition": {
         "StringEquals": {
                  "ecs:CreateAction": [
                      "CreateCluster",
                      "CreateCapacityProvider",
                      "CreateService",
                      "CreateTaskSet",
                      "RegisterContainerInstance",
                      "RegisterTaskDefinition",
                      "RunTask",
                      "StartTask"
                  ]
          }
       }
    }
  ]
}
```

ecs:TagResource アクションはタグがリソース作成アクション時に適用された場合のみ評価されます。したがって、リクエストでタグが指定されていない場合、リソースを作成するアクセス許可を持っているユーザー (タグ付け条件がないと仮定) にはecs:TagResource アクションを実行するアクセス許可は必要ありません。ただし、ユーザーがタグを使用してリソースを作成しようとした場

合、ユーザーが ecs:TagResource アクションを使用するアクセス許可を持っていない場合はリクエストに失敗します。

特定のタグへの Amazon ECS アクセス制御

IAM ポリシーの Condition 要素で追加の条件を使用して、リソースに適用できるタグキーとタグ値を制御できます。

次の条件キーは前のセクションの例で使用できます。

- aws:RequestTag: 特定のタグキーまたはタグキーと値がリクエストに存在している必要があることを指定する場合に使用します。リクエストでは他のタグも指定できます。
  - StringEquals 条件演算子とともに使用して、特定のタグキーと値の組み合わせを適用します。例えば、タグ cost-center=cc123: を適用します。

    ```
    "StringEquals": { "aws:RequestTag/cost-center": "cc123" }
    ```

  - StringLike 条件演算子とともに使用して、リクエストで特定のタグキーを適用します。例えば、タグキー purpose を適用します。

    ```
    "StringLike": { "aws:RequestTag/purpose": "*" }
    ```

- aws:TagKeys: リクエストで使用されるタグキーを適用する場合に使用します。
  - リクエストにタグが指定されている場合はForAllValues 修飾子を使用して特定のタグキーのみを適用します (リクエストにタグが指定されている場合、特定のタグキーのみが許可されます。他のタグは許可されません)。例えば、タグキー environment または cost-center が適用されます:

    ```
    "ForAllValues:StringEquals": { "aws:TagKeys": ["environment","cost-center"] }
    ```

  - ForAnyValue 修飾子とともに使用して、指定されたタグキーの少なくとも 1 つがリクエストに存在することを要求します。例えば、タグキー environment または webserver のうち少なくとも 1 つがリクエストに存在している必要があります。

    ```
    "ForAnyValue:StringEquals": { "aws:TagKeys": ["environment","webserver"] }
    ```

これらの条件キーは、ecs:TagResource アクションだけでなく、タグ付けをサポートするリソース作成アクションにも適用できます。Amazon ECS API アクションがタグ付けをサポートしている

かどうかについては、「Amazon ECS のアクション、リソース、および条件キー」を参照してください。

リソースの作成時にタグを指定するようにユーザーに強制するにはリソース作成アクションで aws:RequestTag 修飾子とともに aws:TagKeys 条件キーまたは ForAnyValue 条件キーを使用する必要があります。ユーザーがリソース作成アクションのタグを指定しない場合、ecs:TagResource アクションは評価されません。

条件においては条件キーでは大文字と小文字が区別されず、条件値では大文字と小文字が区別されます。したがって、タグキーの大文字と小文字を区別するには条件の値としてタグキーが指定される aws:TagKeys 条件キーを使用します。

複数値条件の詳細については、「IAM ユーザーガイド」の「複数のコンテキストキーまたは値による条件」を参照してください。

リソースタグを使用して Amazon ECS リソースへのアクセスを制御する

Amazon ECS リソースを使用するための許可をユーザーに付与する IAM ポリシーを作成する場合、ポリシーの Condition 要素にタグ情報を含めることで、タグに基づいてアクセスをコントロールできます。これは属性ベースのアクセス制御 (ABAC) と呼ばれます。ABAC を使用すると、ユーザーが変更、使用、または削除できるリソースをより適切に制御できます。詳細については、AWS の ABAC とはを参照してください。

例えば、ユーザーによるクラスターの削除を許可するが、クラスターにタグ environment=production がある場合はそのアクションを拒否するポリシーを作成できます。これを行うにはaws:ResourceTag 条件キーを使用し、リソースにアタッチされているタグに基づいてリソースへのアクセスを許可または拒否します。

```
"StringEquals": { "aws:ResourceTag/environment": "production" }
```

Amazon ECS API アクションが aws:ResourceTag 条件キーを使用したアクセスの制御をサポートしているかどうかについては、「Amazon ECS のアクション、リソース、および条件キー」を参照してください。Describe アクションはリソースレベルのアクセス権限をサポートしないため、条件のない別のステートメントでそれらのアクセス権限を指定する必要があることに注意してください。

IAM ポリシーの例は Amazon ECS ポリシーの例 を参照してください。

タグに基づいてリソースへのユーザーのアクセスを許可または拒否する場合はユーザーが同じリソースに対してそれらのタグを追加または削除することを明示的に拒否することを検討する必要がありま

す。そうしないと、ユーザーはそのリソースのタグを変更することで、制限を回避してリソースにアクセスできてしまいます。

Amazon ECS ポリシーの例

IAM ポリシーを使用して、Amazon ECS コンソールで特定のリソースを表示、および操作するための許可をユーザーに付与することができます。上記のセクションのサンプルポリシーを使用することはできますが、これらは AWS CLI または AWS SDK で作成されたリクエスト向けに設計されています。

例: タグに基づいてユーザーが Amazon ECS クラスターを削除するのを許可する

次のポリシーは、タグに「Purpose/Testing」のキー/値のペアがある場合、ユーザーがクラスターを削除することを許可します。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "ecs:DeleteCluster"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:ecs:us-east-1:111122223333:cluster/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Purpose": "Testing"
                }
            }
        }
    ]
}
```

Amazon Elastic Container Service のアイデンティティとアクセスのトラブルシューティング

次の情報は、Amazon ECS と IAM の使用に伴って発生する可能性がある一般的な問題の診断や修復に役立ちます。

トピック

- Amazon ECS でアクションを実行する権限がない
- iam:PassRole を実行する権限がありません
- 自分の AWS アカウント 以外のユーザーに Amazon ECS リソースへのアクセスを許可したい
- Amazon ECS マネージドインスタンスのインスタンスプロファイルに問題がある
- その他のトラブルシューティングリソース

Amazon ECS でアクションを実行する権限がない

アクションを実行する権限がないというエラーが表示された場合は、そのアクションを実行できるようにポリシーを更新する必要があります。

次のエラー例は、mateojackson IAM ユーザーがコンソールを使用して、ある *my-example-widget* リソースに関する詳細情報を表示しようとしたことを想定して、その際に必要な ecs:*GetWidget* アクセス許可を持っていない場合に発生するものです。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 ecs:GetWidget on resource: my-example-widget
```

この場合、ecs:*GetWidget* アクションを使用して *my-example-widget* リソースへのアクセスを許可するように、mateojackson ユーザーのポリシーを更新する必要があります。

サポートが必要な場合は、AWS 管理者に問い合わせてください。サインイン認証情報を提供した担当者が管理者です。

iam:PassRole を実行する権限がありません

iam:PassRole アクションを実行する権限がないというエラーが表示された場合は、ポリシーを更新して Amazon ECS にロールを渡せるようにする必要があります。

一部の AWS のサービス では、新しいサービスロールやサービスリンクロールを作成せずに、既存のロールをサービスに渡すことができます。そのためには、サービスにロールを渡すアクセス許可が必要です。

以下の例のエラーは、marymajor という IAM ユーザーがコンソールを使用して Amazon ECS でアクションを実行しようする場合に発生します。ただし、このアクションをサービスが実行するには、サービスロールから付与されたアクセス許可が必要です。Mary には、ロールをサービスに渡すアクセス許可がありません。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

この場合、Mary のポリシーを更新してメアリーに `iam:PassRole` アクションの実行を許可する必要があります。

サポートが必要な場合は、AWS 管理者に問い合わせてください。サインイン認証情報を提供した担当者が管理者です。

Amazon ECS マネージドインスタンスを使用している時にこのエラーが発生した場合、そのインスタンスロール名が、マネージドポリシーで義務付けられている命名規則に従っていない可能性があります。詳細については、「Amazon ECS マネージドインスタンスのインスタンスプロファイルに問題がある」を参照してください。

自分の AWS アカウント 以外のユーザーに Amazon ECS リソースへのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセスコントロールリスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください:

- Amazon ECS がこれらの機能をサポートしているかどうかについては、「IAM を使用するAmazon Elastic Container Service」を参照してください。
- 所有している AWS アカウント 全体のリソースへのアクセス権を提供する方法については、IAM ユーザーガイド の 所有している別の AWS アカウント へのアクセス権を IAM ユーザーに提供 を参照してください。
- サードパーティの AWS アカウント にリソースへのアクセス権を提供する方法については、「IAM ユーザーガイド」の「サードパーティが所有する AWS アカウント へのアクセス権を付与する」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、IAM ユーザーガイド の 外部で認証されたユーザー (ID フェデレーション) へのアクセスの許可 を参照してください。
- クロスアカウントアクセスにおけるロールとリソースベースのポリシーの使用方法の違いについては、「IAM ユーザーガイド」の「IAM でのクロスアカウントのリソースへのアクセス」を参照してください。

Amazon ECS マネージドインスタンスのインスタンスプロファイルに問題がある

AmazonECSInfrastructureRolePolicyForManagedInstances マネージドポリシーを使用する場合、インスタンスロール名は ecsInstanceRole で始まる必要があります。ポリシーの範囲は iam:PassRole から arn:aws:iam::*:role/ecsInstanceRole* であるため、名前が一致しない場合、タスクの起動時に認証エラーが発生します。

これは AWS::IAM::Role リソースから RoleName を省略する場合に CloudFormation に一般的です。その理由は、CloudFormation によってポリシー条件に合致しない MyStack-InstanceRole-ABC123 のような名前が自動生成されるためです。

この問題を解決するには、次のいずれかを実行します。

- 名前がマネージドポリシーに合致するように、AWS::IAM::Role リソースに RoleName: ecsInstanceRole を追加します。
- インスタンスロール ARN を対象とするインフラストラクチャロールに明示的な iam:PassRole インラインポリシーを追加します。

CloudFormation テンプレートと詳細な手順については、「CloudFormation を使用してインスタンスプロファイルを作成する」を参照してください。

その他のトラブルシューティングリソース

次のページに、エラーコードに関する情報が記載されています。

- Amazon ECS の停止したタスクのエラーメッセージ
- Amazon ECS のサービスイベントメッセージを表示する

Amazon ECS の IAM ベストプラクティス

AWS Identity and Access Management (IAM) を使用すると、認証と認可を目的としたルールベースのポリシーを通じて、AWS のサービスとリソースへのアクセスの管理と制御を行うことができます。具体的には、このサービスを通じてユーザー、グループ、またはロールに適用されるポリシーを使用して、AWS リソースへのアクセスを制御します。この 3 つのうち、ユーザーはリソースにアクセスできるアカウントです。また、IAM ロールは、認証された ID が引き受けることができる一連のアクセス許可であり、IAM 外の特定の ID には関連付けられていません。詳細については、「Amazon ECS overview of access management: Permissions and policies」を参照してください。

最小権限アクセスのポリシーに従う

ユーザーが所定の仕事を遂行できるようにポリシーの範囲を設定してください。たとえば、開発者が定期的にタスクを停止する必要がある場合、その特定のアクションのみを許可するポリシーを作成します。次の例では、特定の Amazon リソースネーム (ARN) を持つクラスター上の　特定の `task_family` に属するタスクを停止させることができます。リソースレベルのアクセス許可を使用する別の例として、条件で ARN を参照することが挙げられます。リソースレベルのアクセス許可を使用して、アクションを適用するリソースを指定できます。詳細については、「Amazon ECS のポリシーリソース」を参照してください。

クラスターリソースを管理上の境界として使用する

ポリシーの範囲が狭すぎると、ロールが急増し、管理オーバーヘッドが増える可能性があります。特定のタスクやサービスのみを対象とするロールを作成するのではなく、クラスターを対象とするロールを作成し、そのクラスターを主要な管理上の境界として使用してください。

自動パイプラインを作成して API からエンドユーザーを隔離する

アプリケーションを自動的にパッケージ化して Amazon ECS クラスターにデプロイするパイプラインを作成することで、ユーザーが使用できるアクションを制限できます。これにより、タスクを作成、更新、削除するジョブをパイプラインに効果的に委任することができます。詳細については、「AWS CodePipeline ユーザーガイド」の「チュートリアル: CodePipeline を使用した Amazon ECS 標準デプロイ」を参照してください。

ポリシー条件を使用してセキュリティをより一層強化する

セキュリティを強化する必要がある場合は、ポリシーに条件を追加します。これは、権限のある操作を実行する場合や、特定のリソースに対して実行できるアクションを制限する必要がある場合に役立ちます。以下は、クラスターを削除する際に多要素認可を必要とするポリシーの例です。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:DeleteCluster"
      ],
```

```
      "Condition": {
        "Bool": {
          "aws:MultiFactorAuthPresent": "true"
        }
      },
    "Resource": ["*"]
    }
  ]
}
```

サービスに適用されたタグは、そのサービスに含まれるすべてのタスクに反映されます。したがって、特定のタグを使用して Amazon ECS リソースを対象とするロールを作成できます。次のポリシーでは、タグキーが `Department` でタグ値が `Accounting` であるすべてのタスクを IAM プリンシパルが開始および停止します。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:StartTask",
                "ecs:StopTask",
                "ecs:RunTask"
            ],
            "Resource": "arn:aws:ecs:us-east-1:123456789012:cluster/my-cluster",
            "Condition": {
                "StringEquals": {"ecs:ResourceTag/Department": "Accounting"}
            }
        }
    ]
}
```

API へのアクセスを定期的に監査する

ユーザーがロールを変更する場合があります。ロールを変更した後、以前に付与されていたアクセス許可が適用されなくなる場合があります。どのユーザーが Amazon ECS API にアクセスでき、その

アクセスが引き続き保証されているかどうかを必ず監査してください。ユーザーが組織を離れると自動的にアクセス権を取り消すユーザーライフサイクル管理ソリューションと IAM を統合することを検討してください。詳細については、「AWS Identity and Access Management ユーザーガイド」の「AWS セキュリティ監査のガイドライン」を参照してください。

Amazon Elastic Container Service でのログとモニタリング

モニタリングは、Amazon Elastic Container Service および AWS ソリューションの信頼性、可用性、パフォーマンスを維持する上で重要な部分です。マルチポイント障害が発生した場合は、その障害をより簡単にデバッグできるように、AWS ソリューションのすべての部分からモニタリングデータを収集する必要があります。AWS には、Amazon ECS リソースをモニタリングし、潜在的なインシデントに対応するための複数のツールが用意されています。

Amazon CloudWatch アラーム

指定した期間にわたって単一のメトリクスを監視し、複数の期間にわたり特定のしきい値に関連するメトリクス値に基づいて 1 つ以上のアクションを実行します。アクションは、Amazon Simple Notification Service (Amazon SNS) のトピックまたは Amazon EC2 Auto Scaling のポリシーに送信される通知です。CloudWatch アラームは、特定の状態にあるという理由だけでアクションを呼び出すことはありません。状態が変更され、指定された期間維持されている必要があります。詳細については、「CloudWatch を使用して Amazon ECS をモニタリングする」を参照してください。

Fargate を使用するタスクがあるサービスでは、CloudWatch アラームを使用して、CPU やメモリの使用率などの CloudWatch メトリクスに基づいてサービス内のタスクをスケールインおよびスケールアウトできます。詳細については、「Amazon ECS サービスを自動的にスケールする」を参照してください。

EC2 を使用するタスクまたはサービスがあるクラスターでは、CloudWatch アラームを使用して、クラスターのメモリ使用率などの CloudWatch メトリクスに基づいてコンテナインスタンスをスケールインおよびスケールアウトできます。

Amazon CloudWatch Logs

Amazon ECS タスク定義で `awslogs` ログドライバーを指定することで、Amazon ECS タスクのコンテナからのログファイルをモニタリング、保存、およびアクセスできます。詳細については、「awslogs ログドライバーを使用する」を参照してください。

Amazon ECS コンテナインスタンスからのオペレーティングシステムおよび Amazon ECS コンテナエージェントのログファイルをモニタリング、保存、アクセスすることもできます。このログへのアクセス方法は、EC2 を使用するコンテナの場合に使うことができます。

Amazon CloudWatch Events

イベントを一致させ、イベントを 1 つ以上のターゲット関数やストリームにルーティングして、変更を加えたり、状態情報を取得したり、修正アクションを実行したりします。詳細については、このガイドの「EventBridge を使用して Amazon ECS エラーへの対応を自動化する」および「Amazon EventBridge ユーザーガイド」の「EventBridge は、Amazon CloudWatch Events の進化形です」を参照してください。

AWS CloudTrail ログ

CloudTrail では、Amazon ECS のユーザー、ロール、または AWS のサービスによって実行されたアクションの記録を確認できます。CloudTrail で収集された情報を使用して、Amazon ECS に対するリクエスト、リクエスト元の IP アドレス、リクエスト者、リクエスト日時などの詳細を確認できます。詳細については、「AWS CloudTrail を使用して Amazon ECS API コールをログに記録する」を参照してください。

AWS Trusted Advisor

Trusted Advisor は、AWS の数十万のお客様にサービスを提供することにより得られた、運用実績から学んだベストプラクティスを活用しています。Trusted Advisor はお客様の AWS 環境を検査し、システムの可用性とパフォーマンスを向上させたりセキュリティギャップを埋めたりする機会がある場合には、推奨事項を作成します。すべての AWS のお客様は、Trusted Advisor の 5 つのチェックにアクセスできます。ビジネスまたはエンタープライズサポートプランをご利用のお客様は、すべての Trusted Advisor チェックを表示できます。

詳細については、サポート ユーザーガイドの AWS Trusted Advisor を参照してください。

AWS Compute Optimizer

AWS Compute Optimizer は、AWS リソースの設定と使用率のメトリクスを分析するサービスです。Compute Optimizer は、リソースが最適かどうかを報告し、最適化に関するレコメンデーションを生成してコストを削減およびワークロードのパフォーマンスを改善します。

詳細については、「AWS Compute OptimizerAmazon ECS に関する 推奨事項」を参照してください。

Amazon ECS のモニタリングでもう 1 つ重要な点は、CloudWatch のアラームの対象外の項目を手動でモニタリングすることです。Trusted Advisor、CloudWatch、その他の AWS コンソールのダッ

シュボードには、AWS 環境の状態が一目でわかるように表示されます。コンテナインスタンスおよびタスクのコンテナのログファイルも確認することをお勧めします。

Amazon Elastic Container Service でのコンプライアンス検証

新しい AWS サインアップエクスペリエンスは、規制対象ワークロード向けには設計されていません。新しい AWS サインアップエクスペリエンスを使用しているものの、規制対象ワークロードに AWS を使用する場合は、AWS (高度) にサインアップしたり、AWS 環境の高度な機能をアクティブ化したりできます。

AWS のサービスが特定のコンプライアンスプログラムの対象であるかどうかを確認するには、「コンプライアンスプログラムによる対象範囲内の AWS のサービス」を参照して、関心のあるコンプライアンスプログラムを選択してください。一般的な情報については、AWSコンプライアンスプログラム を参照してください。

AWS Artifact を使用して、サードパーティの監査レポートをダウンロードできます。詳細については、「AWS Artifact でレポートをダウンロードする」「」を参照してください。

AWS のサービスを使用する際のお客様のコンプライアンス責任は、お客様のデータの機密性や貴社のコンプライアンス目的、適用可能な法律および規制によって決定されます。AWS のサービス を使用する際のコンプライアンス責任の詳細については、「AWS セキュリティドキュメント」を参照してください。

Amazon ECS のセキュリティとコンプライアンスのベストプラクティス

Amazon ECS を使用する際のお客様のコンプライアンス責任は、お客様のデータの機密性や貴社のコンプライアンス目標、および該当する法規制によって決定されます。

決済カード業界のデータセキュリティ基準 (PCI DSS)

PCI DSS を順守する際には、環境内のカード所有者データ (CHD) の流れ全体を理解することが重要です。CHD フローは PCI DSS の適用性を決定し、カード会員データ環境 (CDE) の境界と構成要素を定義し、PCI DSS 評価の範囲を定義します。PCI DSS の範囲を正確に決定することは、セキュリティ体制を定義し、最終的に評価を成功させるための鍵となります。範囲の完全性を保証し、範囲からの変更や逸脱を検出する範囲決定手順は、お客様で用意する必要があります。

コンテナ化されたアプリケーションは一時的な性質を持つため、構成の監査がさらに複雑になります。そのため、コンテナのライフサイクルの全段階でコンプライアンス要件に対応できるよう、お客様はコンテナのすべての設定パラメータを常に把握しておく必要があります。

Amazon ECS での PCI DSS コンプライアンスの達成方法の詳細については、次のホワイトペーパーを参照してください。

- Amazon ECS での PCI DSS コンプライアンスに向けたアーキテクチャ構築

HIPAA (米国の医療保険の相互運用性と説明責任に関する法令)

Amazon ECS で保護対象医療情報 (PHI) を処理するワークロードを使用する場合、追加の設定は不要です。Amazon ECS は、Amazon EC2 でのコンテナの起動を調整するオーケストレーションサービスとして機能します。オーケストレーション対象のワークロード内のデータに対して動作したり、データに基づいて動作したりすることはありません。HIPAA 規制および AWS ビジネスアソシエイト補遺に従い、Amazon ECS で起動されたコンテナが PHI にアクセスする場合は、転送中も保存中も暗号化する必要があります。

AWS ストレージオプションごとに Amazon S3、Amazon EBS、AWS KMS など、保存時の暗号化のためのさまざまなメカニズムを利用できます。オーバーレイネットワーク (VNS3 や Weave Net など) をデプロイして、コンテナ間で転送される PHI を完全に暗号化したり、あるいは暗号化の冗長レイヤーを提供することもできます。完全なログを使用し、すべてのコンテナログが Amazon CloudWatch に送信されるようにしてください。インフラストラクチャセキュリティのベストプラクティスの使用については、「セキュリティの柱 – AWS Well-Architected フレームワーク」の「インフラストラクチャの保護」を参照してください。

AWS Security Hub CSPM

AWS Security Hub CSPM を使用します。この AWS のサービス は、AWS 内のセキュリティ状態の包括的なビューを提供します。Security Hub CSPM では、セキュリティコントロールを使用して AWS リソースを評価し、セキュリティ業界標準とベストプラクティスに対するコンプライアンスをチェックします。サポートされているサービスとコントロールの一覧については、「Security Hub CSPM のコントロールリファレンス」を参照してください。

Amazon ECS Runtime Monitoring を使用した Amazon GuardDuty

Amazon GuardDuty は、AWS 環境内のアカウント、コンテナ、ワークロード、データを保護する脅威検知サービスです。GuardDuty は、機械学習（ML）モデル、異常および脅威検出機能を使用して、さまざまなログソースとランタイムアクティビティを継続的に監視し、環境内の潜在的なセキュリティリスクと悪意のあるアクティビティを特定して優先順位を付けます。

GuardDuty のランタイムモニタリングを使用して、悪意のある、または不正な動作を特定します。ランタイムモニタリングは、AWS ログとネットワークアクティビティを継続的に監視して悪意のあ

る動作や不正な動作を特定することで、Fargate および EC2 で実行されているワークロードを保護します。ランタイムモニタリングは、軽量でフルマネージド型の GuardDuty セキュリティエージェントを使用して、ファイルアクセス、プロセス実行、ネットワーク接続などのホスト上動作を分析します。これは、Amazon EC2 インスタンスおよびコンテナワークロードでの権限の昇格、流出した認証情報の使用、悪意のある IP アドレスやドメインとの通信、マルウェアの存在などの問題に対応しています。詳細については、「GuardDuty ユーザーガイド」の「GuardDuty ランタイムモニタリング」を参照してください。

コンプライアンスに関する推奨事項

関連するコンプライアンスプログラムを効果的に運用するには、早めに社内のコンプライアンスプログラムの所有者に働きかけることで、AWS 責任共有モデルを利用してコンプライアンス管理責任の所在を明確化することをお勧めします。詳細については、「AWSAmazon ECS の 責任共有モデル。」を参照してください。

AWS Fargate 連邦情報処理標準 (FIPS-140)

連邦情報処理規格 (Federal Information Processing Standards/FIPS–140) は、機密情報を保護する暗号モジュールのセキュリティ要件を規定する、米国およびカナダ政府の基準です。FIPS-140 は、転送中のデータと保管中のデータの暗号化に使用できる、一連の検証済みの暗号化関数を定めています。

FIPS-140 コンプライアンスをオンにすると、FIPS-140 に準拠した態様で Fargate でワークロードを実行できます。FIPS-140 コンプライアンスの詳細については、「連邦情報処理規格 (FIPS) 140-3」を参照してください。

AWS Fargate FIPS-140 に関する考慮事項

Fargate で FIPS-140 コンプライアンスを使用する場合は、次の点を考慮してください。

- FIPS-140 コンプライアンスは、AWS GovCloud (US) リージョンでのみ利用できます。
- Fargate が FIPS-140 バージョン 140.3 をサポート
- FIPS-140 コンプライアンスは、デフォルトでオフになっています。オンにする必要があります。
- Amazon CloudWatch は、FIPS–140 コンプライアンスを使用する IPv6 のみの設定で、Amazon ECS タスクのモニタリングに使用できるデュアルスタック FIPS エンドポイントをサポートしていません。
- タスクでは、FIPS-140 コンプライアンスのために次の設定を使用する必要があります。

- `operatingSystemFamily` は `LINUX` でなければなりません。
- `cpuArchitecture` は `X86_64` でなければなりません。
- Fargate プラットフォームバージョンは `1.4.0` 以降である必要があります。

Fargate で FIPS を使用する

Fargate で FIPS-140 コンプライアンスを使用するには、次の手順を実行します。

1. FIPS-140 コンプライアンスをオンにします。詳細については、「the section called “AWS Fargate 連邦情報処理標準 (FIPS-140) コンプライアンス”」を参照してください。
2. オプションで、ECS Exec を使用して次のコマンドを実行し、クラスターの FIPS-140 コンプライアンスステータスを検証できます。

   cluster-name をクラスターの名前に、*task-id* をタスクの ID または ARN に、*container-name* をコマンドを実行するタスク内のコンテナ名に置き換えます。

   戻り値「1」は、FIPS が使用されていることを示します。

   ```
   aws ecs execute-command \
       --cluster cluster-name \
       --task task-id \
       --container container-name \
       --interactive \
       --command "cat /proc/sys/crypto/fips_enabled"
   ```

Fargate FIPS-140 監査に CloudTrail を使用する

CloudTrail は、アカウント作成時に AWS で有効になります。Amazon ECS で API およびコンソールアクティビティが発生すると、そのアクティビティは [イベント履歴] で AWS のその他のサービスのイベントとともに CloudTrail イベントにレコードされます。最近のイベントは、AWS アカウントで表示、検索、ダウンロードできます。詳細については、CloudTrail イベント履歴でのイベントの表示を参照してください。

Amazon ECS のイベントを含む、AWS アカウントでのイベントを継続的に記録するために、CloudTrail がログファイルを Amazon S3 バケットに配信する際に使用する証跡を作成します。デフォルトでは、コンソールで追跡を作成するときに、追跡がすべてのリージョンに適用されます。証跡は、AWS パーティションのすべてのリージョンからのイベントをログに記録し、指定し

た Amazon S3 バケットにログファイルを配信します。さらに、CloudTrail ログで収集したイベントデータをより詳細に分析し、それに基づいて対応するため、他の AWS サービスを構成できます。詳細については、「the section called “AWS CloudTrail を使用して Amazon ECS API コールをログに記録する”」を参照してください。

PutAccountSettingDefault API アクションを示す CloudTrail ログエントリの例を次に示します。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDAIV5AJI5LXF5EXAMPLE",
         "arn": "arn:aws:iam::123456789012:user/jdoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIPWIOFC3EXAMPLE",
    },
    "eventTime": "2023-03-01T21:45:18Z",
    "eventSource": "ecs.amazonaws.com",
    "eventName": "PutAccountSettingDefault",
    "awsRegion": "us-gov-east-1",
    "sourceIPAddress": "52.94.133.131",
    "userAgent": "aws-cli/2.9.8 Python/3.9.11 Windows/10 exe/AMD64 prompt/off command/
ecs.put-account-setting",
    "requestParameters": {
        "name": "fargateFIPSMode",
        "value": "enabled"
    },
    "responseElements": {
        "setting": {
            "name": "fargateFIPSMode",
            "value": "enabled",
            "principalArn": "arn:aws:iam::123456789012:user/jdoe"
        }
    },
    "requestID": "acdc731e-e506-447c-965d-f5f75EXAMPLE",
    "eventID": "6afced68-75cd-4d44-8076-0beEXAMPLE",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management",
    "tlsDetails": {
```

```
        "tlsVersion": "TLSv1.2",
        "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
        "clientProvidedHostHeader": "ecs-fips.us-gov-east-1.amazonaws.com"
    }
}
```

Amazon Elastic Container Service におけるインフラストラクチャセキュリティ

マネージドサービスとして、Amazon Elastic Container Service は AWS グローバルネットワークセキュリティによって保護されています。AWSセキュリティサービスと AWS がインフラストラクチャを保護する方法については、AWS クラウドセキュリティ を参照してください。インフラストラクチャセキュリティのベストプラクティスを使用して AWS 環境を設計するには、セキュリティの柱 - AWS Well-Architected Framework の インフラストラクチャ保護 を参照してください。

AWS が公開している API コールを使用し、ネットワーク経由で Amazon ECS にアクセスします。クライアントは次をサポートする必要があります。

- Transport Layer Security (TLS)。TLS 1.2 が必須で、TLS 1.3 をお勧めします。
- DHE (楕円ディフィー・ヘルマン鍵共有) や ECDHE (楕円曲線ディフィー・ヘルマン鍵共有) などの完全前方秘匿性 (PFS) による暗号スイート。これらのモードは Java 7 以降など、ほとんどの最新システムでサポートされています。

これらの API オペレーションは任意のネットワークの場所から呼び出すことができます。Amazon ECS は、リソースベースのアクセスポリシーをサポートしており、送信元 IP アドレスに基づく制限を含めることができるため、ポリシーがネットワークロケーションの IP アドレスを考慮していることを確認してください。また、Amazon ECS ポリシーを使用して、特定の Amazon Virtual Private Cloud エンドポイントまたは特定の VPC からのアクセスを制御することもできます。これにより、実質的に AWS ネットワーク内の特定の VPC からの特定の Amazon ECS リソースへのネットワークアクセスが分離されます。詳細については、「Amazon ECS とインターフェイス VPC エンドポイント (AWS PrivateLink)」を参照してください。

Amazon ECS とインターフェイス VPC エンドポイント (AWS PrivateLink)

インターフェイス VPC エンドポイントを使用するように Amazon ECS を設定することで、VPC のセキュリティ体制を強化できます。インターフェイスエンドポイントは、AWS PrivateLink (プライベート IP アドレスを使用した Amazon ECS API へのプライベートなアクセスを可能にするテクノロ

ジ) により動作しています。AWS PrivateLink は、VPC とAmazon ECS 間のすべてのネットワークトラフィックを、Amazon ネットワークに制限します。インターネットゲートウェイ、NAT デバイス、または仮想プライベートゲートウェイは必要ありません。

AWS PrivateLink および VPC エンドポイントの詳細については、「Amazon VPC ユーザーガイド」の「VPC エンドポイント」を参照してください。

考慮事項

2023 年 12 月 23 日以降に導入されたリージョンのエンドポイントに関する考慮事項

Amazon ECS 用のインターフェイス VPC エンドポイントを設定する前に、以下の考慮事項に注意してください：

- 次のリージョン固有の VPC エンドポイントが必要です。

  > **Note**
  >
  > すべてのエンドポイントを設定しなかった場合、トラフィックが経由するのは VPC エンドポイントではなく、パブリックエンドポイントになります。

  - `com.amazonaws.region.ecs-agent`
  - `com.amazonaws.region.ecs-telemetry`
  - `com.amazonaws.region.ecs`

  例えば、カナダ西部 (カルガリー) (ca-west-1) リージョンでは、次の VPC エンドポイントが必要です。

  - `com.amazonaws.ca-west-1.ecs-agent`
  - `com.amazonaws.ca-west-1.ecs-telemetry`
  - `com.amazonaws.ca-west-1.ecs`
- テンプレートを使用して新しいリージョンに AWS リソースを作成する際に、2023 年 12 月 23 日以前に導入されたリージョンのテンプレートをコピーした場合は、コピー元のリージョンに応じて次のいずれかの操作を実行してください。

  例えば、コピー元のリージョンが米国東部 (バージニア北部) (us-east-1) であるとします。コピー先のリージョンは、カナダ西部 (カルガリー) (ca-west-1) です。

設定	アクション	
コピー元のリージョンには VPC エンドポイントがありません。	新しいリージョンの 3 つの VPC エンドポイントをすべて作成します (例: `com.amazonaws.ca-west-1.ecs-agent`)。	
コピー元のリージョンに、リージョン固有の VPC エンドポイントが含まれています。	a. 新しいリージョンの 3 つの VPC エンドポイントをすべて作成します (例: `com.amazonaws.ca-west-1.ecs-agent`)。 b. コピー元のリージョンの 3 つの VPC エンドポイントをすべて削除します (例: `com.amazonaws.us-east-1.ecs-agent`)。	

Fargate の Amazon ECS VPC エンドポイントに関する考慮事項

Fargate タスクがデプロイされているのと同じ VPC に `ecr.dkr` および `ecr.api` の VPC エンドポイントがある場合は、その VPC エンドポイントが使用されます。VPC エンドポイントがない場合は、Fargate インターフェースが使用されます。

Amazon ECS 用のインターフェイス VPC エンドポイントを設定する前に、以下の考慮事項に注意してください：

- Fargate を使用するタスクでは、Amazon ECS 用のインターフェイス VPC エンドポイントは必要ありませんが、以下のように Amazon ECR のインターフェイス VPC エンドポイント、Secrets Manager、または Amazon CloudWatch Logs が必要になる場合があります。
  - Amazon ECR からプライベートイメージをプルできるようにするには、Amazon ECR 用のインターフェイス VPC エンドポイントを 作成する必要があります。詳細については、Amazon Elastic Container Registry ユーザーガイドの「インターフェイス VPC エンドポイント (AWS PrivateLink)」を参照してください。

> **Important**
>
> - インターフェイス VPC エンドポイントを使用するよう Amazon ECR を設定する場合、特定の VPC または VPC エンドポイントへのアクセスを制限する条件キーを含むタスク実行ロールを作成できます。詳細については、「インターフェイスエンドポイントのアクセス許可によって Amazon ECR イメージをプルする Fargate タスクです。」を参照してください。
> - タスクが IPv6 のみの設定で、Amazon ECR デュアルスタックイメージ URI を使用している場合、Amazon ECR はデュアルスタックインターフェイス VPC エンドポイントに対応しませんので注意してください。詳しくは Amazon Elastic Container Registry ユーザーガイドの「IPv6 を使用したリクエストの実行の開始方法」をご確認ください。

  - タスクで Secrets Manager から機密データをプルできるようにするには、Secrets Manager 用のインターフェイス VPC エンドポイントを作成する必要があります。詳細については、AWS Secrets Managerユーザーガイドの「VPC エンドポイントで Secrets Manager を作成する」を参照してください。
  - VPC にインターネットゲートウェイがなく、タスクで `awslogs` ログドライバーを使用してログ情報を CloudWatch Logs に送信する場合は、CloudWatch Logs 用のインターフェイス VPC エンドポイントを作成する必要があります。詳細については、Amazon CloudWatch Logs ユーザーガイドの「インターフェイス VPC エンドポイントでの CloudWatch Logs の使用」を参照してください。
- 現在、VPC エンドポイントはクロスリージョンリクエストをサポートしていません。Amazon ECS に対して API コールを発行するリージョンと同じリージョンにエンドポイントを作成してください。 例えば、タスクを米国東部 (バージニア北部) で実行することを考えてみます。その場合、Amazon ECS VPC エンドポイントは、米国東部 (バージニア北部) に作成する必要があります。他のリージョンで作成された Amazon ECS VPC エンドポイントは、米国東部 (バージニア北部) 内のタスクを実行できません。
- VPC エンドポイントでは、Amazon Route 53 を介して Amazon 提供の DNS のみがサポートされています。独自の DNS を使用したい場合は、条件付き DNS 転送を使用できます。詳細については、Amazon VPC ユーザーガイド の DHCP Options Sets を参照してください。
- VPC エンドポイントにアタッチされたセキュリティグループは、VPCのプライベートサブネットからの着信接続をポート 443 で許可する必要があります。
- Envoy プロキシの Service Connect 管理では、`com.amazonaws.region.ecs-agent` VPC エンドポイントを使用します。VPC エンドポイントを使用しない場合、Envoy プロキシの Service

Connect 管理は、そのリージョンの ecs-sc エンドポイントを使用します。各リージョンの Amazon ECS エンドポイントのリストについては、「Amazon ECS のエンドポイントとクォータ」を参照してください。

EC2 の Amazon ECS VPC エンドポイントに関する考慮事項

Amazon ECS 用のインターフェイス VPC エンドポイントを設定する前に、以下の考慮事項に注意してください：

- EC2 を使用するタスクでは、起動されたコンテナインスタンスが Amazon ECS コンテナエージェントのバージョン 1.25.1 以降を実行する必要があります。詳細については、「Amazon ECS Linux コンテナインスタンスの管理」を参照してください。
- タスクで Secrets Manager から機密データをプルできるようにするには、Secrets Manager 用のインターフェイス VPC エンドポイントを作成する必要があります。詳細については、AWS Secrets Managerユーザーガイドの「VPC エンドポイントで Secrets Manager を作成する」を参照してください。
- VPC にインターネットゲートウェイがなく、タスクで awslogs ログドライバーを使用してログ情報を CloudWatch Logs に送信する場合は、CloudWatch Logs 用のインターフェイス VPC エンドポイントを作成する必要があります。詳細については、Amazon CloudWatch Logs ユーザーガイドの「インターフェイス VPC エンドポイントでの CloudWatch Logs の使用」を参照してください。
- 現在、VPC エンドポイントはクロスリージョンリクエストをサポートしていません。Amazon ECS に対して API コールを発行するリージョンと同じリージョンにエンドポイントを作成してください。 例えば、タスクを米国東部 (バージニア北部) で実行することを考えてみます。その場合、Amazon ECS VPC エンドポイントは、米国東部 (バージニア北部) に作成する必要があります。他のリージョンで作成された Amazon ECS VPC エンドポイントは、米国東部 (バージニア北部) 内のタスクを実行できません。
- VPC エンドポイントでは、Amazon Route 53 を介して Amazon 提供の DNS のみがサポートされています。独自の DNS を使用したい場合は、条件付き DNS 転送を使用できます。詳細については、Amazon VPC ユーザーガイド の DHCP Options Sets を参照してください。
- VPC エンドポイントにアタッチされたセキュリティグループは、VPCのプライベートサブネットからの着信接続をポート 443 で許可する必要があります。

Amazon ECS エンドポイントの命名パターンについて

Amazon ECS エージェントは、次のような番号付きサフィックスを備えたエンドポイントにリクエストを行う場合があることを理解することが重要です。

- エージェントエンドポイントの場合 `ecs-a-1.region.amazonaws.com`、`ecs-a-2.region.amazonaws.com` など
- テレメトリエンドポイントの場合 `ecs-t-1.region.amazonaws.com`、`ecs-t-2.region.amazonaws.com` など

この動作は、Amazon ECS エージェントが DiscoverPollEndpoint API を使用して接続する特定のエンドポイントを動的に決定するため発生します。VPC エンドポイントがこれらの番号付きエンドポイントのバリエーションを適切に処理しない場合、ベース名に VPC エンドポイントを設定していても、エージェントはパブリックエンドポイントの使用にフォールバックします。

DiscoverPollEndpoint API の役割

DiscoverPollEndpoint API は、タスクをポーリングする適切なエンドポイントを検出するため Amazon ECS エージェントによって使用されます。エージェントがこの API を呼び出すと、番号付きサフィックスを含む場合がある特定のエンドポイント URL を受け取ります。VPC エンドポイントが正しく動作するようにするには、ネットワーク設定でエージェントに以下を許可する必要があります。

1. DiscoverPollEndpoint API へのアクセス
2. 番号付きサフィックスを含む、返されたエンドポイント URL への接続

VPC エンドポイントの接続の問題をトラブルシューティングする場合は、エージェントがベースエンドポイントと、DiscoverPollEndpoint API によって返される可能性のある番号付きバリエーションの両方に到達できることを確認します。

Amazon ECS 用の VPC エンドポイントの作成

Amazon ECS サービス用の VPC エンドポイントを作成するには、「Amazon VPC ユーザーガイド」の「インターフェイス VPC エンドポイント手順を使用して AWS サービスにアクセスする」の手順を使用して、以下のエンドポイントを作成します。VPC 内に既存のコンテナインスタンスがある場合は、一覧表示されている順にエンドポイントを作成する必要があります。VPC エンドポイントが作成された後にコンテナインスタンスを作成する場合、順序は関係ありません。

> **Note**
>
> すべてのエンドポイントを設定しなかった場合、トラフィックが経由するのは VPC エンドポイントではなく、パブリックエンドポイントになります。
> エンドポイントを作成すると、Amazon ECS はエンドポイントのプライベート DNS 名も作成します。例えば、ecs-agent の `ecs-a.region.amazonaws.com` とecs-telemetry の `ecs-t.region.amazonaws.com` です。

- `com.amazonaws.region.ecs-agent`
- `com.amazonaws.region.ecs-telemetry`
- `com.amazonaws.region.ecs`

> **Note**
>
> *region* は、米国東部 (オハイオ) リージョンの `us-east-2` のように、Amazon ECS でサポートされている AWS リージョンのリージョン識別子を表します。

`ecs-agent` エンドポイントは `ecs:poll` API を使用し、`ecs-telemetry` エンドポイントは `ecs:poll` および `ecs:StartTelemetrySession` API を使用します。

EC2 起動タイプを使用する既存のタスクがある場合は、VPC エンドポイントを作成した後に、各コンテナインスタンスで新しい設定が選択される必要があります。そのためには、各コンテナインスタンスを再起動するか、各コンテナインスタンスで Amazon ECS コンテナエージェントを再起動する必要があります。コンテナエージェントを再起動するには、以下を実行します。

Amazon ECS コンテナエージェントを再起動するには

1. SSH 経由でコンテナインスタンスにログインします。
2. コンテナエージェントを停止します。

```
sudo docker stop ecs-agent
```

3. コンテナエージェントを開始します。

```
sudo docker start ecs-agent
```

VPC エンドポイントを作成し、各コンテナインスタンスで Amazon ECS コンテナエージェントを再起動したら、新しく起動されるすべてのタスクで新しい設定が選択されます。

Amazon ECS 用の VPC エンドポイントポリシーの作成

VPC エンドポイントに Amazon ECS へのアクセスをコントロールするエンドポイントポリシーをアタッチできます。このポリシーでは、以下の情報を指定します。

- アクションを実行できるプリンシパル。
- 実行可能なアクション。
- このアクションを実行できるリソース。

詳細については、Amazon VPC ユーザーガイドの「VPC エンドポイントによるサービスのアクセスコントロール」を参照してください。

例: Amazon ECS アクションの VPC エンドポイントポリシー

Amazon ECS のエンドポイントポリシーの例を次に示します。このポリシーは、エンドポイントに接続すると、クラスターの作成や一覧表示が行えるようにアクセスを許可します。`CreateCluster` と `ListClusters` のアクションはリソースを受け入れないため、すべてのリソースでリソース定義は * に設定されます。

```
{
   "Statement":[
    {
      "Principal":"*",
      "Effect": "Allow",
      "Action": [
        "ecs:CreateCluster",
        "ecs:ListClusters"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

AWSAmazon ECS の 責任共有モデル。

セキュリティとコンプライアンスに関して、AWS とお客様の間で責任を共有します。この共有モデルにより、ホストオペレーティングシステムや仮想化レイヤーからサービスが運用されている施設の物理的なセキュリティに至るまで、さまざまなコンポーネントを AWS が運用、管理、制御するため、お客様の運用の負担が軽減されます。お客様は、AWS が提供するセキュリティグループのファイアウォール設定に加えて、ゲストオペレーティングシステム (更新やセキュリティパッチを含む) およびその他の関連アプリケーションソフトウェアを管理し、責任を持って管理する必要があります。お客様の責任範囲は、使用するサービス、IT 環境へのサービス統合、適用される法規制に応じて異なります。このため、お客様は選択するサービスを注意深く検討する必要があります。この責任共有モデルの性質によって柔軟性が得られ、お客様はデプロイを統制できます。

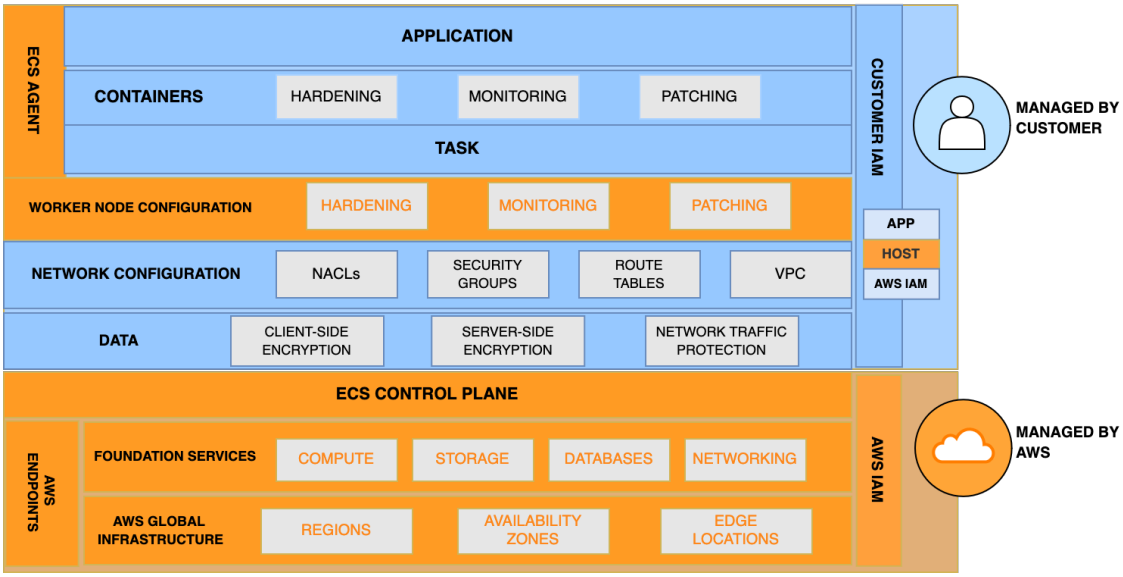
Fargate

次の図は、Fargate 起動タイプの責任共有モデルを示しています。Fargate は分離されたハードウェア仮想化環境で各ワークロードを実行します。その結果、各タスクは専用のインフラストラクチャキャパシティを取得します。Fargate で実行されているコンテナ化されたワークロードは、オペレーティングシステム、Linux カーネル、ネットワークインターフェイス、エフェメラルストレージ、CPU、またはメモリを他のタスクと共有しません。Fargate を使用する場合、お客様はコンテナを実行するコンピューティングインフラストラクチャのセキュリティを管理する責任を負いません。Fargate は、お客様のワークロードを実行するインフラストラクチャをプロビジョニングしてパッチを適用します。詳細については、「AWS Fargate on Amazon ECS のタスクの廃止とメンテナンス」を参照してください。

お客様は次のリソースを管理する責任があります。

- VPC、NACL、セキュリティグループ、ルートテーブルなどのネットワーク設定
- クライアントとサービスストレージの暗号化。詳細については、「Amazon ECS タスクのストレージオプション」を参照してください。
- コンテナイメージ 詳細については、「Amazon ECS タスクおよびコンテナのセキュリティのベストプラクティス」を参照してください。
- タスクロールを使用したアプリケーションの IAM アクセス許可。詳細については、「Amazon ECS タスクの IAM ロール」を参照してください。

AWS Shared Responsibility Model for Amazon ECS with Fargate

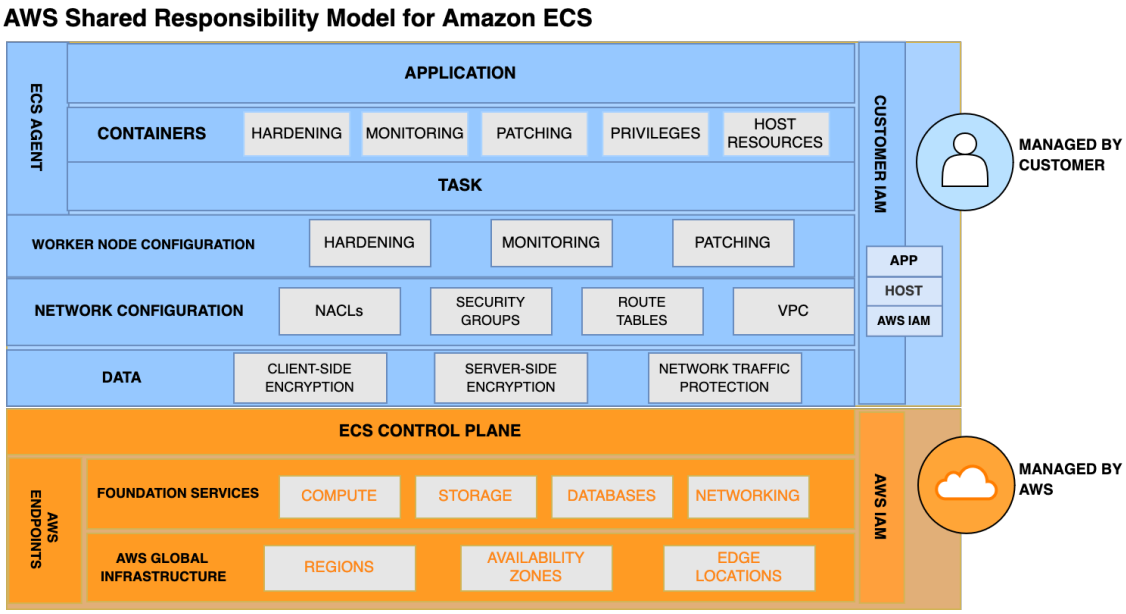


EC2

次の図は、EC2 の責任共有を示しています。EC2 インスタンスでタスクを実行する場合、次のリソースに加えて EC2 インスタンスを保守する責任があります。

- Amazon ECS エージェント。
- パッチ適用と強化を含む EC2 インスタンス AMI。
- VPC、NACL、セキュリティグループ、ルートテーブルなどのネットワーク設定。
- クライアントとサービスストレージの暗号化。詳細については、「Amazon ECS タスクのストレージオプション」を参照してください。
- コンテナイメージ 詳細については、「Amazon ECS タスクおよびコンテナのセキュリティのベストプラクティス」を参照してください。
- タスクロールを使用したアプリケーションの IAM アクセス許可。詳細については、「Amazon ECS タスクの IAM ロール」を参照してください。
- Amazon EC2 インスタンスメタデータサービス (IMDS) へのコンテナアクセスをブロックするためのコンテナインスタンスの設定。詳細については、「ロールの推奨事項」を参照してください。
- 共存ワークロードのセキュリティ上の考慮事項。Fargate と異なり、EC2 コンテナインスタンスにタスク分離はありません。コンテナは、同じインスタンス上の他のタスクから、以前実行されたタスクによって残されたデータなどの認証情報、環境変数、一時ファイルにアクセスする可能性があります。機密性の高いアプリケーションの場合、各アプリケーションがハードウェア専有インスタンスで実行されるように環境を設定します。

次の図は、EC2 の責任共有モデルを示しています。どのセキュリティ責任が AWS によって管理され、どのセキュリティ責任がお客様のものかを示しています。



Amazon ECS マネージドインスタンスの責任共有モデル

Amazon ECS マネージドインスタンスは、Fargate のシンプルな運用と、幅広い Amazon EC2 インスタンスタイプと機能へのアクセスを組み合わせて、コンテナ化されたワークロードによるマネージドソリューションを提供します。AWS はインフラストラクチャのプロビジョニング、パッチ適用、スケーリング、メンテナンスを処理し、お客様はアプリケーションと特定の設定を制御します。

Fargate とは異なり、Amazon ECS マネージドインスタンスで実行されているコンテナ化されたワークロードは、オペレーティングシステム、Linux カーネル、ネットワークインターフェイス、エフェメラルストレージ、CPU、メモリ、GPU リソースを、同じインスタンス上の他のタスクと共有します。Amazon ECS は、未使用の容量を最小限に抑えるために、より大きいインスタンスに複数のタスクを配置することで、インフラストラクチャの使用率を最適化します。

AWS の責任

Amazon ECS マネージドインスタンスを使用する場合、AWS は以下を担当します。

- インスタンスのプロビジョニングとライフサイクル管理
- オペレーティングシステムのパッチ適用とセキュリティ更新
- インフラストラクチャのスケーリングと最適化
- インスタンスの置換とメンテナンス (最大 21 日間のインスタンス有効期間)

- アクセスコントロールの制限 (SSH アクセスなし、SSM セッションマネージャーのアクセスなし)
- Amazon EC2 インスタンスストレージは暗号化され、インスタンスに直接アタッチされたストレージとなります。詳細については、「Amazon EC2 でのデータ保護」を参照してください。
- Amazon ECS は、ルートボリュームやデータボリュームなど、作成時に Amazon EC2 インスタンスにアタッチされたボリュームを管理します。
- Amazon ECS は内部で Amazon EC2 マネージドインスタンスを使用します。Amazon EC2 マネージドインスタンスの詳細については、「Amazon EC2 のセキュリティ」を参照してください。

お客様の責任

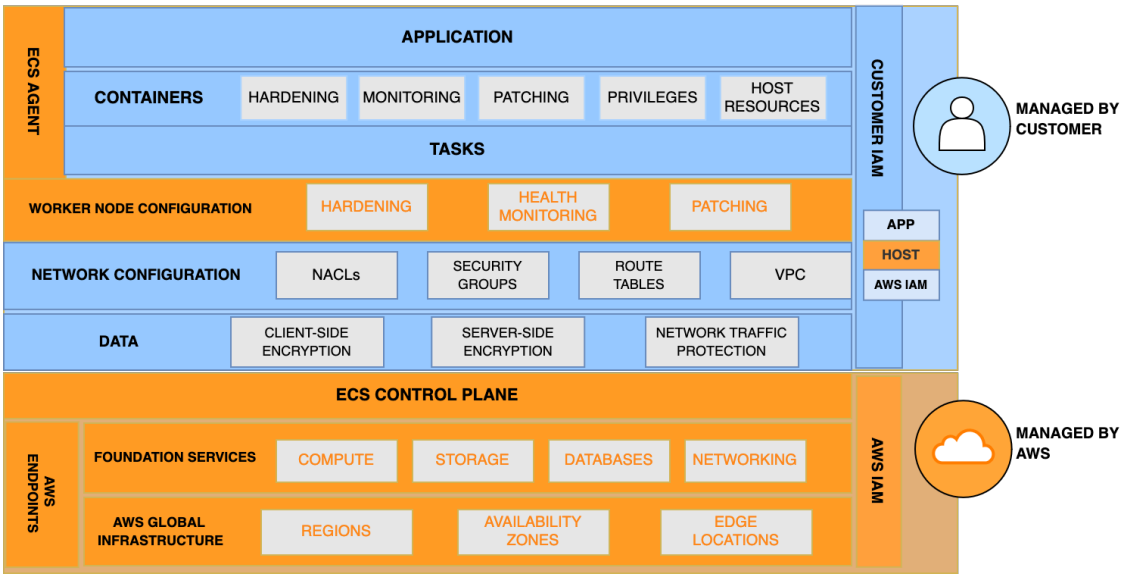
お客様は次のリソースを管理する責任があります。

- VPC、NACL、セキュリティグループ、ルートテーブルなどのネットワーク設定
- クライアントとサービスストレージの暗号化。詳細については、「Amazon ECS タスクのストレージオプション」を参照してください。
- コンテナイメージ 詳細については、「Amazon ECS タスクおよびコンテナのセキュリティのベストプラクティス」を参照してください。
- タスクロールを使用したアプリケーションの IAM アクセス許可。詳細については、「Amazon ECS タスクの IAM ロール」を参照してください。
- アプリケーションレベルの設定とモニタリング
- コンテナのセキュリティモニタリング
- タスクとサービスの定義
- 共存ワークロードのセキュリティ上の考慮事項。Fargate と異なり、Amazon ECS マネージドインスタンスにタスク分離はありません。コンテナは、同じインスタンス上の他のタスクから、以前実行されたタスクによって残されたデータなどの認証情報、環境変数、一時ファイルにアクセスする可能性があります。
- 有効化された場合の特権コンテナ設定と拡張 Linux 機能 (CAP_NET_ADMIN、CAP_BPF、CAP_PERFMON など)
- Amazon ECS API による管理オペレーション (SSH または SSM 経由のインスタンスへの直接アクセスは利用できません)

Amazon ECS マネージドインスタンスのセキュリティにおける AWS とお客様の責任については、「Amazon ECS マネージドインスタンスのセキュリティ上の考慮事項」を参照してください。

次の図は、Amazon ECS マネージドインスタンスの責任共有モデルを示しています。どのセキュリティ責任が AWS によって管理され、どのセキュリティ責任がお客様のものかを示しています。

AWS Shared Responsibility Model for Amazon ECS with Managed Instances



Amazon ECS のネットワークセキュリティのベストプラクティス

ネットワークセキュリティは、いくつかのサブトピックを含む幅広いトピックです。これには、転送中の暗号化、ネットワークのセグメンテーションと分離、ファイアウォール、トラフィックルーティング、オブザーバビリティなどが含まれます。

転送中の暗号化

ネットワークトラフィックを暗号化することで、データがネットワーク経由で送信されるときに、権限のないユーザーにデータを傍受されて読み取られることを防ぎます。Amazon ECS では、次のいずれかの方法でネットワーク暗号化を実装できます。

- Nitro インスタンスを使用する:

  デフォルトでは、C5n、G4、I3en、M5dn、M5n、P3dn、R5dn、R5n の Nitro インスタンスタイプ間のトラフィックは自動的に暗号化されます。トラフィックがトランジットゲートウェイ、ロードバランサー、または同様の仲介を経由する場合、トラフィックは暗号化されません。

  - 転送中の暗号化
  - 2019年の新発表
  - 「リ・インフォース 2019」からの会話
- Application Load Balancer で Server Name Indication (SNI) を使用する

Application Load Balancer (ALB) と Network Load Balancer (NLB) は、Server Name Indication (SNI) をサポートします。SNI を使用すると、複数の安全なアプリケーションを 1 つのリスナーのバックグラウンドに配置できます。そのため、各リスナーには独自の TLS 証明書があります。AWS Certificate Manager (ACM) を使用してロードバランサーの証明書をプロビジョニングし、リスナーの証明書リストに追加することをお勧めします。AWS ロードバランサーは、SNI を使用するスマート証明書の選択アルゴリズムを使用します。クライアントから提供されたホスト名と一致する証明書がリスト内にひとつだけある場合、ロードバランサーはこの証明書を選択します。クライアントが提供するホスト名と一致する証明書がリスト内に複数ある場合、ロードバランサーはクライアントがサポートできる証明書を選択します。例として、自己署名証明書や ACM を通じて生成された証明書などがあげられます。

- Application Load Balancer で SNI を使用する
- Network Load Balancer で SNI を使用する

- TLS 証明書を使用したエンドツーエンド暗号化:

  これには、タスクとともに TLS 証明書をデプロイすることが含まれます。自己署名証明書でも、信頼できる認証局からの証明書でもかまいません。証明書のシークレットを参照することで証明書を取得できます。それ以外の場合は、ACM に証明書署名リクエスト (CSR) を発行し、その結果得られたシークレットを共有ボリュームにマウントするコンテナを実行することもできます。

  - Amazon ECS でNetwork Load Balancer を使用してコンテナまでトランスポートレイヤーのセキュリティを維持する (パート 1)
  - コンテナまで Transport Layer Security (TLS) を維持する (パート 2): AWS Private Certificate Authority を使用する

タスクネットワーク

以下の推奨事項は Amazon ECS の仕組みに関する考慮したものです。Amazon ECS はオーバーレイネットワークを使用しません。代わりに、タスクは異なるネットワークモードで動作するように設定されます。たとえば、`bridge` モードを使用するように設定されたタスクは、各ホストで実行される Docker ネットワークからルーティングできない IP アドレスを取得します。`awsvpc` ネットワークモードを使用するように設定されたタスクは、ホストのサブネットから IP アドレスを取得します。`host` ネットワークを使用して構成されたタスクはホストのネットワークインターフェースを使用します。`awsvpc` が推奨ネットワークモードです。これは、タスクにセキュリティグループを割り当てるのに使用できる唯一のモードであるため推奨されます。また、Amazon ECS の AWS Fargate タスクで使用できる唯一のモードでもあります。

タスクのセキュリティグループ

awsvpc ネットワークモードを使用するようにタスクを設定することをお勧めします。このモードを使用するようにタスクを設定すると、Amazon ECS エージェントは自動的に Elastic Network Interface (ENI) をプロビジョニングしてタスクにアタッチします。ENI がプロビジョニングされると、タスクは AWS セキュリティグループに登録されます。セキュリティグループは、インバウンドトラフィックとアウトバウンドトラフィックをコントロールするための仮想ファイアウォールとして機能します。

タスクまたはサービスでカスタムファイアウォールを使用する場合は、Amazon ECS エージェント管理エンドポイント (「`ecs-a-*.region.amazonaws.com`」)、テレメトリエンドポイント (「`ecs-t-*.region.amazonaws.com`」)、および Service Connect Envoy 管理エンドポイント (「`ecs-sc.region.api.aws`」) のトラフィックを許可するアウトバウンドルールを追加します。

AWS PrivateLink および Amazon ECS

AWS PrivateLink は Amazon ECS を含むさまざまな AWS サービスのプライベートエンドポイントを作成できるようにするネットワーク技術です。エンドポイントは、Amazon VPC に Internet Gateway (IGW) が接続されておらず、インターネットへの代替ルートもないサンドボックス環境で必要となります。AWS PrivateLink を使用することで、Amazon ECS サービスへの呼び出しが Amazon VPC 内にとどまり、インターネットを経由しないことが保証されます。Amazon ECS およびその他の関連サービスの AWS PrivateLink エンドポイントを作成する方法については、「Amazon ECS インターフェイス Amazon VPC エンドポイント」を参照してください。

> **Important**
>
> AWS Fargate タスクには Amazon ECS の AWS PrivateLink エンドポイントは必要ありません。

Amazon ECR と Amazon ECS はどちらもエンドポイントポリシーをサポートしています。これらのポリシーにより、サービスの API へのアクセスを絞り込むことができます。たとえば、特定の AWS アカウントのレジストリにのみイメージをプッシュすることを許可する Amazon ECR のエンドポイントポリシーを作成できます。このようなポリシーは、コンテナイメージを通じてデータが流出するのを防ぎつつ、ユーザーが許可された Amazon ECR レジストリにプッシュできるようにするために使用できます。詳細については、「VPC エンドポイントポリシーの使用」を参照してください。

以下のポリシーでは、アカウントのすべての AWS プリンシパルが Amazon ECR リポジトリに対してのみ、すべてのアクションを実行できるようにします。

```
{
  "Statement": [
    {
      "Sid": "LimitECRAccess",
      "Principal": "*",
      "Action": "*",
      "Effect": "Allow",
      "Resource": "arn:aws:ecr:region:account_id:repository/*"
    }
  ]
}
```

新しい PrincipalOrgID プロパティを使用する条件を設定することで、これをさらに強化できます。これにより、ユーザーの AWS Organizations の一部ではない IAM プリンシパルによるイメージのプッシュやプルを防ぐことができます。詳細については、「aws:PrincipalOrgID」を参照してください。

com.amazonaws.*region*.ecr.dkr と com.amazonaws.*region*.ecr.api エンドポイントの両方に同じポリシーを適用することをお勧めします。

コンテナエージェントの設定

Amazon ECS コンテナエージェント設定ファイルには、ネットワークセキュリティに関連する複数の環境変数が含まれています。ECS_AWSVPC_BLOCK_IMDS と ECS_ENABLE_TASK_IAM_ROLE_NETWORK_HOST は、Amazon EC2 メタデータへのタスクのアクセスをブロックするために使用されます。HTTP_PROXY は、HTTP プロキシを経由してインターネットに接続するようにエージェントを設定するために使用されます。エージェントと Docker ランタイムをプロキシ経由でルーティングするように設定する方法については、「HTTP プロキシ設定」を参照してください。

> **Important**
>
> AWS Fargate を使用する場合は、これらの設定を使用することはできません。

ネットワークセキュリティに関する推奨事項

Amazon VPC、ロードバランサー、ネットワークを設定するときは、次のことを行うことをお勧めします。

Amazon ECS で必要に応じてネットワーク暗号化を使用する

必要に応じてネットワーク暗号化を使用してください。PCI DSS などの特定のコンプライアンスプログラムでは、データにカード所有者のデータが含まれている場合、転送中のデータを暗号化する必要があります。ワークロードに同様の要件がある場合は、ネットワーク暗号化を設定してください。

最新のブラウザでは、安全でないサイトに接続するとユーザーに警告が表示されます。公開されているロードバランサーがサービスのフロントエンドで使用されている場合は、TLS/SSL を使用してクライアントのブラウザーからロードバランサーへのトラフィックを暗号化し、必要に応じてバックエンドで再暗号化します。

`awsvpc` ネットワークモードとセキュリティグループを使用して、Amazon ECS のタスクと他のリソース間のトラフィックを制御する

タスク間のトラフィック、またはタスクと他のネットワークリソース間のトラフィックを制御する必要がある場合は、`awsvpc` ネットワークモードとセキュリティグループを使用してください。サービスが ALB の内部にある場合は、セキュリティグループを使用して、ALB と同じセキュリティグループを使用する他のネットワークリソースからのインバウンドトラフィックのみを許可します。アプリケーションが NLB の背後にある場合は、Amazon VPC CIDR 範囲からのインバウンドトラフィックと NLB に割り当てられた静的 IP アドレスのみを許可するようにタスクのセキュリティグループを設定します。

セキュリティグループは、タスクと Amazon VPC 内の他のリソース (Amazon RDS データベースなど) との間のトラフィックを制御するためにも使用する必要があります。

ネットワークトラフィックを厳密に分離する必要がある場合に、別の Amazon VPC に Amazon ECS クラスターを作成する

ネットワークトラフィックを厳密に分離する必要がある場合は、別の Amazon VPC にクラスターを作成してください。セキュリティ要件が厳しいワークロードを、その要件を満たす必要のないワークロードを含むクラスターで実行することは避けてください。ネットワークの厳密な分離が必須の場合は、別の Amazon VPC にクラスターを作成し、Amazon VPC エンドポイントを使用して他の Amazon VPC にサービスを選択的に公開します。詳細については、「VPC エンドポイント」を参照してください。

Amazon ECS で必要に応じて AWS PrivateLink エンドポイントを設定する

AWS PrivateLink エンドポイントは、必要に応じて設定してください。セキュリティポリシーにより Amazon VPC にインターネットゲートウェイ (IGW) をアタッチできない場合は、Amazon

ECS や Amazon ECR、AWS Secrets Manager、Amazon CloudWatch などの他のサービスの AWS PrivateLink エンドポイントを設定します。

Amazon VPC フローログを使用して、Amazon ECS の長時間実行されるタスクとの間のトラフィックを分析する

長時間実行されるタスクとの間のトラフィックを分析するには、Amazon VPC フローログを使用してください。`awsvpc` ネットワークモードを使用するタスクには独自の ENI が割り当てられます。これにより、Amazon VPC フローログを使用して個々のタスクに出入りするトラフィックをモニタリングできます。Amazon VPC フローログ (v3) の最新アップデートでは、VPC ID、サブネット ID、インスタンス ID などのトラフィックメタデータがログに追加されました。このメタデータを使用して、調査を絞り込むことができます。詳細については、「Amazon VPC フローログ」を参照してください。

> **Note**
>
> コンテナは一時的な性質を持つため、フローログは、異なるコンテナの間やコンテナとその他のネットワークリソースの間のトラフィックパターンを分析するうえで必ずしも効果的な方法であるとは限りません。

Amazon ECS タスクおよびコンテナのセキュリティのベストプラクティス

コンテナイメージは、攻撃に対する防御の最前線と見なしてください。安全性が低く、構築が不十分なイメージがあると、攻撃者はコンテナの境界を抜け出し、ホストにアクセスできるようになる可能性があります。これが発生するリスクを軽減するには、次のことを行う必要があります。

タスクとコンテナを設定するときは、次のことを行うことをお勧めします。

最小限のイメージを作成するか、distroless のイメージを使用する

まず、コンテナイメージから不要なバイナリをすべて削除します。Amazon ECR Public Gallery にある見慣れないイメージを使用している場合は、そのイメージを調べてコンテナの各レイヤーの内容を参照してください。これには Dive などのアプリケーションを使用できます。

あるいは、アプリケーションとそのランタイム依存関係のみを含む distroless イメージを使用することもできます。パッケージマネージャーやシェルは含まれません。Distroless イメージは、「スキャ

ナーの信号対雑音比を向上させ、必要なものだけを調達する負担を軽減する」ことができます。詳細については、GitHub の distroless に関するドキュメントを参照してください。

Docker には、scratch と呼ばれる、予約済みの最小限のイメージからイメージを作成するメカニズムがあります。詳細については、Docker ドキュメントの「scratch を使用した単純な親イメージの作成」を参照してください。Go などの言語では、静的にリンクされたバイナリを作成して Dockerfile で参照できます。以下の例で、その方法を示します。

```
############################
# STEP 1 build executable binary
############################
FROM golang:alpine AS builder
# Install git.
# Git is required for fetching the dependencies.
RUN apk update && apk add --no-cache git
WORKDIR $GOPATH/src/mypackage/myapp/
COPY . .
# Fetch dependencies.
# Using go get.
RUN go get -d -v
# Build the binary.
RUN go build -o /go/bin/hello
############################
# STEP 2 build a small image
############################
FROM scratch
# Copy our static executable.
COPY --from=builder /go/bin/hello /go/bin/hello
# Run the hello binary.
ENTRYPOINT ["/go/bin/hello"]
This creates a container image that consists of your application and nothing else,
 making it extremely secure.
```

前の例も、マルチステージビルドの例です。これらのタイプのビルドは、コンテナレジストリにプッシュされる最終イメージのサイズを最小限に抑えることができるため、セキュリティの観点からは魅力的です。コンテナイメージからビルドツールやその他の無関係なバイナリを省くことで、イメージのアタックサーフェスが減り、セキュリティ対策を強化することができます。マルチステージビルドの詳細については、Docker ドキュメントの「Multi-stage builds」を参照してください。

イメージをスキャンして脆弱性がないか調べる

対応する仮想マシンと同様に、コンテナイメージには脆弱性のあるバイナリやアプリケーションライブラリが含まれていたり、時間の経過とともに脆弱性が生じたりする可能性があります。エクスプロイトから保護する最善の方法は、イメージスキャナーでイメージを定期的にスキャンすることです。

Amazon ECR には、共通脆弱性識別子 (CVE) データベースを使用する 2 つのバージョンの基本スキャンが用意されています。

- AWS ネイティブ基本スキャン – AWS ネイティブテクノロジーを使用してください。これは現在一般提供中であり、推奨されるアプローチです。この改善された基本スキャンは、さまざまな人気のオペレーティングシステムにおいてより優れたスキャン結果と脆弱性検出を提供するよう設計されています。これにより、お客様はコンテナイメージのセキュリティをさらに強化できます。すべての新しい顧客レジストリは、デフォルトでこの改善されたバージョンにオプトインされます。
- Clair 基本スキャン – オープンソースの Clair プロジェクトを使用する旧バージョンの基本スキャン。Clair の詳細については、GitHub の Clair を参照してください。

AWS ネイティブスキャンと Clair 基本スキャンは、2024 年 9 月以降に新たに追加されたリージョンを除き、「リージョン別 AWS サービス」にリストされているすべてのリージョンでサポートされています。Clair のサポートは非推奨であるため、Clair は追加された新しいリージョンではサポートされておらず、また 2025 年 10 月 1 日をもってすべてのリージョンでのサポートを停止します。

Amazon ECR では、アップストリームのディストリビューションソースからの CVE の重要度が使用されます (使用可能な場合)。それ以外の場合は、共通脆弱性評価システム (CVSS) のスコアが使用されます。CVSS スコアは、NVD 脆弱性の重大度評価を取得するために使用できます。詳細については、「NVD 脆弱性の重大度」を参照してください。

また、Amazon ECR コンソール内から、あるいは DescribeImageScanFindings API を呼び出して、スキャンの結果を確認することもできます。脆弱性が `HIGH` または `CRITICAL` のイメージは、削除するか再構築する必要があります。デプロイされたイメージに脆弱性が生じた場合は、できるだけ早く交換する必要があります。

Docker Desktop Edge バージョン 2.3.6.0 以降では、ローカルイメージをスキャンできます。スキャンはアプリケーションセキュリティサービスの Snyk によって行われます。脆弱性が発見されると、Snyk は Dockerfile 内の脆弱性のあるレイヤーと依存関係を識別します。また、脆弱性が少なく、よりスリムなベースイメージを使用したり、特定のパッケージを新しいバージョンにアップグレードしたりするなど、安全な代替手段も推奨します。Docker スキャンを使用すれば、開発者はイメージをレジストリにプッシュする前に、潜在的なセキュリティ問題を解決できます。

- Amazon ECR と AWS Security Hub CSPM を使用してイメージコンプライアンスを自動化するでは、AWS Security Hub CSPM の Amazon ECR から脆弱性情報を検出し、脆弱なイメージへのアクセスをブロックすることで修復を自動化する方法について説明します。

イメージから特別な権限を削除する

アクセス権フラグsetuid と setgid は、実行ファイルの所有者またはグループの権限で実行できるようにします。これらのアクセス権限を持つバイナリは権限昇格に利用される可能性があるため、イメージからすべてのバイナリを削除してください。悪意のある目的に使用される可能性のある、nc や curl などのシェルやユーティリティをすべて削除することを検討してください。次のコマンドを使用すると setuid および setgid アクセス権のあるファイルを見つけることができます。

```
find / -perm /6000 -type f -exec ls -ld {} \;
```

これらのファイルから、これらの特別な権限を削除するには、以下のディレクティブをコンテナイメージに追加します。

```
RUN find / -xdev -perm /6000 -type f -exec chmod a-s {} \; || true
```

キュレーションされたイメージのセットを作成する

開発者が独自のイメージを作成できるようにするのではなく、組織内のさまざまなアプリケーションスタック用に、検証済みのイメージセットを作成することができます。そうすることで、開発者は Dockerfile の作成方法を学ぶ必要がなくなり、コードの記述に集中できます。変更がコードベースにマージされると、CI/CD パイプラインがアセットを自動的にコンパイルし、アーティファクトリポジトリに保存できます。最後に、アーティファクトを適切なイメージにコピーしてから、Amazon ECR などの Docker レジストリにプッシュします。少なくとも、開発者が独自の Dockerfile を作成できるようなベースイメージのセットを作成する必要があります。Docker Hub からイメージをプルすることは避けてください。イメージに含まれている内容が常に明らかであるとは限りません。イメージが脆弱性を抱えている可能性もあります。

アプリケーションパッケージとライブラリをスキャンして脆弱性がないか調べる

オープンソースライブラリの使用は今や一般的になっています。オペレーティングシステムや OS パッケージと同様に、これらのライブラリには脆弱性が潜んでいる可能性があります。開発ライフサイクルの一環として、重大な脆弱性が見つかった場合は、これらのライブラリをスキャンして更新する必要があります。

Docker Desktop は Snyk を使用してローカルスキャンを実行します。また、オープンソースライブラリの脆弱性や潜在的なライセンス問題の検出にも使用できます。開発者のワークフローに直接統合できるため、オープンソースライブラリがもたらすリスクを軽減できます。詳細については、以下の各トピックを参照してください。

- オープンソースのアプリケーションセキュリティツールには、アプリケーションの脆弱性を検出するためのツールのリストが含まれています。

静的コード分析を行う

コンテナイメージを構築する前に、静的コード分析を実行する必要があります。これはソースコードに対して実行され、コーディングエラーや、フォールトインジェクションなどの悪意のある攻撃者に悪用される可能性のあるコードの特定に使用されます。Amazon Inspector を使用できます。詳細については、「Amazon Inspector ユーザーガイド」の「Amazon Inspector による Amazon ECR コンテナイメージのスキャン」を参照してください。

非ルートユーザーとしてコンテナを実行する

コンテナを実行するときは、非ルートユーザーとして実行してください。デフォルトでは、Dockerfile に USER ディレクティブが含まれていない限り、コンテナは `root` ユーザーとして実行されます。Docker によって割り当てられるデフォルトの Linux 機能では、`root` として実行できるアクションが制限されますが、制限されるのはごくわずかです。例えば、`root` として実行しているコンテナは、デバイスにアクセスすることがまだできません。

CI/CD パイプラインの一部として、Dockerfiles をリントして USER ディレクティブを探し、見つからない場合はビルドを失敗させる必要があります。詳細については、以下の各トピックを参照してください。

- Dockerfile-lint は RedHat のオープンソースツールで、ファイルがベストプラクティスに準拠しているかどうかを確認するために使用できます。

- Hadolint も、ベストプラクティスに準拠した Docker イメージを構築するためのツールです。

読み取り専用のルートファイルシステムを使用する

読み取り専用のルートファイルシステムを使用してください。コンテナのルートファイルシステムは、デフォルトでは書き込み可能です。コンテナに RO (読み取り専用) ルートファイルシステムを設定すると、データを保存できる場所を明示的に定義する必要性が生じます。これにより、権限が具体的に付与されない限りコンテナのファイルシステムに書き込みができなくなるため、アタックサーフェスが減少します。

> **Note**
>
> ルートファイルシステムが読み取り専用であると、そのファイルシステムへの書き込みが可能であることを想定している特定の OS パッケージで問題が発生する可能性があります。読み取り専用のルートファイルシステムを使用する予定がある場合は、事前に十分にテストしてください。

CPU とメモリの制限を設定してタスクを設定する (Amazon EC2)

以下のリスクを最小限に抑えるには、CPU とメモリの制限を設定してタスクを設定する必要があります。タスクのリソース制限は、タスク内のすべてのコンテナで予約できる CPU とメモリの量の上限を設定します。制限が設定されていない場合、タスクはホストの CPU とメモリにアクセスできます。これにより、共有ホストにデプロイされたタスクが他のタスクのシステムリソースを枯渇させるという問題が発生する可能性があります。

> **Note**
>
> Amazon ECS on AWS Fargate Tasks では CPU とメモリの制限を指定する必要があります。これらの値は請求目的で使用されるためです。Amazon ECS Fargate では、1 つのタスクがすべてのシステムリソースを占有しても問題にはなりません。各タスクは専有インスタンスで実行されるからです。メモリ制限を指定しない場合、Amazon ECS は各コンテナに最低 4MB を割り当てます。同様に、タスクに CPU 制限が設定されていない場合、Amazon ECS コンテナエージェントはタスクに最低 2 つの CPU を割り当てます。

Amazon ECR でイミュータブルタグを使用する

Amazon ECR では、イミュータブルタグを使用してイメージを設定することができるので、これを使用するようにしてください。これにより、変更または更新されたバージョンの画像が、同一のタグで画像リポジトリにプッシュされるのを防ぐことができます。また、攻撃者が侵害されたバージョンのイメージを同じタグが付いたイメージにプッシュすることを防ぐことができます。イミュータブルタグを使用すれば、変更するたびに異なるタグが付いた新しいイメージを効果的にプッシュできます。

コンテナを特権として実行することは避けてください (Amazon EC2)

コンテナを特権として実行することは避けてください。バックグラウンドでは、`privileged` として実行されるコンテナは、ホスト上で拡張された権限でされます。つまり、ホスト上で `root` に割り当てられた Linux 機能がすべてコンテナに継承されるということです。その使用は厳しく制限するか禁止してください。Amazon ECS コンテナエージェント環境変数 `ECS_DISABLE_PRIVILEGED` を `true` に設定して、`privileged` が必要がない場合はコンテナが `privileged` として特定のホストで実行されないようにすることをお勧めします。あるいは、AWS Lambda を使用してタスク定義をスキャンして `privileged` パラメータを使用することもできます。

> **Note**
>
> AWS Fargate 上の Amazon ECS では、コンテナを `privileged` として実行することはサポートされていません。

不要な Linux 機能をコンテナから削除する

Docker コンテナに割り当てられているデフォルトの Linux 機能のリストを以下に示します。各機能の詳細については、「Linux 機能の概要」を参照してください。

```
CAP_CHOWN, CAP_DAC_OVERRIDE, CAP_FOWNER, CAP_FSETID, CAP_KILL,
CAP_SETGID, CAP_SETUID, CAP_SETPCAP, CAP_NET_BIND_SERVICE,
CAP_NET_RAW, CAP_SYS_CHROOT, CAP_MKNOD, CAP_AUDIT_WRITE,
CAP_SETFCAP
```

コンテナが上記の Docker カーネル機能のすべてを必要としない場合は、それらをコンテナから削除することを検討してください。各 Docker カーネル機能の詳細については、「KernelCapabilities」を参照してください。次の操作を実行することで、どの機能が使われているかを確認できます。

- OS パッケージ libcap-ng をインストールし、pscap ユーティリティを実行して各プロセスが使用している機能を一覧表示します。
- capsh を使用して、プロセスが使用している機能を解読することもできます。

カスタマーマネージドキー (CMK) を使用して Amazon ECR にプッシュされるイメージを暗号化する

Amazon ECR にプッシュされるイメージの暗号化には、カスタマーマネージドキー (CMK) を使用してください。Amazon ECR にプッシュされたイメージは、保存時に AWS Key Management Service (AWS KMS) 管理キーで自動的に暗号化されます。独自のキーを使用したい場合は、Amazon ECR ではカスタマーマネージドキー (CMK) による AWS KMS 暗号化がサポートされています。CMK によるサーバー側の暗号化を有効にする前に、「保管データ暗号化」に関するドキュメントに記載されている考慮事項を確認してください。

Amazon ECS のチュートリアル

以下のチュートリアルでは、Amazon ECS の使用時に一般的なタスクを実行する方法を示します。

Amazon ECS の開始方法については、以下のチュートリアルのいずれかを参照してください。

チュートリアルの概要	詳細情報
Fargate での Amazon ECS の開始方法	Fargate 用の Amazon ECS Linux タスクを作成する方法について説明します。
Fargate で Windows コンテナの使用を開始する。	Fargate 用の Amazon ECS Windows タスクを作成する方法について説明します。
EC2 の Windows コンテナの使用を開始する。	EC2 用の Amazon ECS Windows タスクを作成する方法について説明します。

以下のチュートリアルのいずれかを使用すると、AWS CLI を使用して Amazon ECS にタスクをデプロイできます。

チュートリアルの概要	詳細情報
Fargate 用の Linux タスクを作成する。	AWS CLI を使用して、Fargate 用の Amazon ECS Linux タスクを作成する
Fargate 用の Windows タスクを作成する。	AWS CLI を使用して、Fargate 用の Amazon ECS Windows タスクを作成する
EC2 用の Linux タスクを作成する。	AWS CLI を使用して、EC2 用の Amazon ECS タスクを作成する

以下のチュートリアルのいずれかを使用すると、モニタリングとログ記録の詳細を確認できます。

チュートリアルの概要	詳細情報
タスクイベントをリッスンし、CloudWatch Logs ログストリームに書き込むシンプルな Lambda 関数を設定します。	CloudWatch Events イベントをリッスンするように Amazon ECS を設定する
重要なコンテナのいずれかが終了したことにより、タスクの実行が停止したタスクイベントのみをキャプチャする Amazon EventBridge イベントルールを設定します。	Amazon ECS タスク停止イベントに関する Amazon Simple Notification Service アラートの送信
もともと 1 つのコンテキストに属していたが、複数のレコードまたはログ行に分割されたログメッセージを連結します。	複数行またはスタックトレースの Amazon ECS ログメッセージの連結
Amazon ECS で実行されている Windows インスタンスに Fluent Bit コンテナをデプロイし、Windows タスクによって生成されたログを Amazon CloudWatch にストリーミングして集中型ロギングを実行します。	Amazon ECS Windows コンテナに Fluent Bit をデプロイする

Amazon ECS のグループ管理サービスアカウントで Active Directory 認証を使用する方法の詳細については、以下のチュートリアルのいずれかを参照してください。

チュートリアルの概要	詳細情報
EC2 上の Linux コンテナで、グループ管理サービスアカウントを使用します。	Amazon EC2 の Linux コンテナで gMSA を使用する
EC2 上の Windows コンテナで、グループ管理サービスアカウントを使用します。	Amazon ECS の EC2 Windows コンテナで gMSA を使用する方法について説明します。
Fargate 上の Linux コンテナで、グループ管理サービスアカウントを使用します。	Fargate で Linux コンテナに gMSA を使用する
ドメインレスグループの Managed Service Account を使用して、Active Directory にアクセスするための認証情報を持つ Windows コンテナを実行するタスクを作成します。	AWS CLI を使用するドメインレス gMSA で Amazon ECS Windows コンテナを使用する

AWS CLI を使用して、Fargate 用の Amazon ECS Linux タスクを作成する

次のステップでは、AWS CLI を使って Amazon ECS でクラスターのセットアップ、タスク定義の登録、Linux タスクの実行、その他の一般的なシナリオを実行するために役立ちます。AWS CLI の最新バージョンを使用する。最新のバージョンにアップグレードする方法については、「AWS CLI の最新バージョンをインストールまたは更新」を参照してください。

> **Note**
>
> デュアルスタックサービスエンドポイントを使用することで、IPv4 と IPv6 の両方を介して AWS CLI、SDK、および Amazon ECS API から Amazon ECS とやり取りできます。詳細については、「Amazon ECS デュアルスタックエンドポイントの使用」を参照してください。

トピック

- 前提条件
- ステップ 1: クラスターを作成する
- ステップ 2: Linux タスク定義を登録
- ステップ 3: タスク定義をリスト表示する
- ステップ 4: サービスを作成する
- ステップ 5: サービスをリスト表示する
- ステップ 6: 実行中のサービスを記述する
- ステップ 7: テスト
- ステップ 8: クリーンアップ

前提条件

このチュートリアルでは、以下の前提条件をすでに満たしているものとします。

- AWS CLI の最新バージョンがインストールされ、設定されていること。AWS CLI のインストールまたはアップグレードについては、「AWS CLI の最新バージョンをインストールまたは更新」を参照してください。
- 「Amazon ECS を使用するようにセットアップする」のステップを完了していること。
- IAM ユーザーに AmazonECS_FullAccess IAM ポリシー例で指定されている必要なアクセス権限があること。
- VPC およびセキュリティグループが使用できるように作成されていること。このチュートリアルは、Amazon ECR パブリックにホストされているコンテナ画像を使用するため、タスクではインターネットアクセスが必要になります。タスクでインターネットへのルートを設定するには、以下のいずれかのオプションを使用します。
  - Elastic IP アドレスが割り当てられた NAT ゲートウェイでプライベートサブネットを使用する。
  - パブリックサブネットを使用してタスクにパブリック IP アドレスを割り当てる。

  詳細については、「the section called “仮想プライベートクラウドを作成する”」を参照してください。

  セキュリティグループとルールの詳細については、「Amazon Virtual Private Cloud ユーザーガイド」の「VPC のデフォルト セキュリティ グループ」および「ルールの例」を参照してください。

- プライベートサブネットを使用してこのチュートリアルを実行すると、Amazon ECS Exec を使用してコンテナを直接操作し、デプロイをテストできます。ECS Exec を使用するには、タスク IAM ロールを作成する必要があります。タスク IAM ロールおよびその他の前提条件の詳細については、「ECS Exec を使用して Amazon ECS コンテナをモニタリングする」を参照してください。
- (任意) AWS CloudShell は、お客様が独自の EC2 インスタンスを作成する必要なく、コマンドラインを提供するツールです。詳細については、AWS CloudShell ユーザーガイド の「What is AWS CloudShell? (とは ?)」 を参照してください。

ステップ 1: クラスターを作成する

デフォルトでは、アカウントは `default` クラスターを受け取ります。

> **Note**
>
> お客様に提供される `default` クラスターを使用する利点は、後続のコマンドで `--cluster` *`cluster_name`* オプションを指定する必要がないことです。デフォルト以外の独自のクラスターを作成する場合は、そのクラスターで使用する予定のコマンドごとに `--cluster` *`cluster_name`* を指定する必要があります。

以下のコマンドを使用して、一意の名前を付けた独自のクラスターを作成します。

```
aws ecs create-cluster --cluster-name fargate-cluster
```

出力:

```
{
    "cluster": {
        "status": "ACTIVE",
        "defaultCapacityProviderStrategy": [],
        "statistics": [],
        "capacityProviders": [],
        "tags": [],
        "clusterName": "fargate-cluster",
        "settings": [
            {
                "name": "containerInsights",
                "value": "disabled"
            }
```

```
        ],
        "registeredContainerInstancesCount": 0,
        "pendingTasksCount": 0,
        "runningTasksCount": 0,
        "activeServicesCount": 0,
        "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/fargate-cluster"
    }
}
```

ステップ 2: Linux タスク定義を登録

ECS クラスターでタスクを実行する前に、タスク定義を登録する必要があります。タスク定義とは、1 つにグループ化されたコンテナのリストです。以下の例は、Docker Hub でホストされている httpd コンテナイメージを使用して PHP ウェブアプリケーションを作成するシンプルなタスク定義です。使用できるタスク定義パラメータの詳細については、「Amazon ECSの タスク定義」を参照してください。このチュートリアルでは、プライベート サブネットにタスクをデプロイし、そのデプロイメントをテストする場合にのみ taskRoleArn が必要になります。taskRoleArn を、前提条件 で説明した ECS Exec を使用するために作成した IAM タスク ロールに置き換えます。

```
{
       "family": "sample-fargate",
       "networkMode": "awsvpc",
       "taskRoleArn": "arn:aws:iam::aws_account_id:role/execCommandRole",
       "containerDefinitions": [
           {
               "name": "fargate-app",
               "image": "public.ecr.aws/docker/library/httpd:latest",
               "portMappings": [
                   {
                       "containerPort": 80,
                       "hostPort": 80,
                       "protocol": "tcp"
                   }
               ],
               "essential": true,
               "entryPoint": [
                   "sh",
                   "-c"
               ],
               "command": [
                   "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample
App</title> <style>body {margin-top: 40px; background-color: #333;} </style> </
```

```
head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1>
 <h2>Congratulations!</h2> <p>Your application is now running on a container in Amazon
 ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html && httpd-
foreground\""
                ]
            }
        ],
        "requiresCompatibilities": [
            "FARGATE"
        ],
        "cpu": "256",
        "memory": "512"
}
```

タスク定義の JSON をファイルとして保存し、`--cli-input-json file://path_to_file.json` オプションで渡します。

コンテナの定義に JSON ファイルを使用するには。

```
aws ecs register-task-definition --cli-input-json file://$HOME/tasks/fargate-task.json
```

register-task-definition コマンドによって、タスク定義の登録が完了した後、その定義の説明が返されます。

ステップ 3: タスク定義をリスト表示する

list-task-definitions コマンドを使用して、いつでもアカウントのタスク定義をリスト表示できます。このコマンドの出力は、run-task または start-task をコールするときに一緒に使用する、`family` 値および `revision` 値を表示します。

```
aws ecs list-task-definitions
```

出力:

```
{
    "taskDefinitionArns": [
        "arn:aws:ecs:region:aws_account_id:task-definition/sample-fargate:1"
    ]
}
```

ステップ 4: サービスを作成する

アカウントのタスクを登録したら、クラスターに登録されたタスク用のサービスを作成することができます。この例では、sample-fargate:1 タスク定義の 1 つのインスタンスを使用して、クラスターで実行されるサービスを作成します。このタスクにはインターネットへのルートが必要であり、そのために 2 つの方法のいずれかを使用します。1 つの方法は、パブリックサブネットに Elastic IP アドレスを持つ NAT ゲートウェイで構成されたプライベートサブネットを使用することです。もう 1 つの方法は、パブリックサブネットを使用してパブリック IP アドレスをタスクに割り当てることです。次のセクションでは、両方の例を示します。

プライベートサブネットを使用した例。Amazon ECS Exec を使用するには　enable-execute-command　オプションが必要です。

```
aws ecs create-service --cluster fargate-cluster --service-name fargate-service --
task-definition sample-fargate:1 --desired-count 1 --launch-type "FARGATE" --network-
configuration "awsvpcConfiguration={subnets=[subnet-abcd1234],securityGroups=[sg-
abcd1234]}" --enable-execute-command
```

パブリックサブネットを使用した例。

```
aws ecs create-service --cluster fargate-cluster --service-name fargate-service --
task-definition sample-fargate:1 --desired-count 1 --launch-type "FARGATE" --network-
configuration "awsvpcConfiguration={subnets=[subnet-abcd1234],securityGroups=[sg-
abcd1234],assignPublicIp=ENABLED}"
```

create-service コマンドによって、タスク定義の登録が完了した後、その定義の説明が返されます。

ステップ 5: サービスをリスト表示する

クラスターのサービスをリスト表示します。前のセクションで作成したサービスが表示されます。このコマンドでサービス名または完全 ARN を取得できます。これを後で説明するサービスに使用します。

```
aws ecs list-services --cluster fargate-cluster
```

出力:

```
{
    "serviceArns": [
        "arn:aws:ecs:region:aws_account_id:service/fargate-cluster/fargate-service"
```

```
    ]
}
```

ステップ 6: 実行中のサービスを記述する

先に取得したサービス名を使用してタスクを記述し、サービスに関する詳細情報を取得します。

```
aws ecs describe-services --cluster fargate-cluster --services fargate-service
```

このコマンドが正常に実行されると、サービスの障害とサービスの説明が返されます。例えば、services セクションでは、タスクの実行中または保留中のステータスなど、デプロイに関する情報が見つかります。また、タスク定義、ネットワーク設定、タイムスタンプ付きのイベントに関する情報も見つかります。障害セクションでは、呼び出しに関連する障害がある場合にその情報が見つかります。トラブルシューティングについては、「サービスイベントメッセージ」を参照してください。サービスの説明取得の詳細については、「サービスの説明取得」を参照してください。

```
{
    "services": [
        {
            "networkConfiguration": {
                "awsvpcConfiguration": {
                    "subnets": [
                        "subnet-abcd1234"
                    ],
                    "securityGroups": [
                        "sg-abcd1234"
                    ],
                    "assignPublicIp": "ENABLED"
                }
            },
            "launchType": "FARGATE",
            "enableECSManagedTags": false,
            "loadBalancers": [],
            "deploymentController": {
                "type": "ECS"
            },
            "desiredCount": 1,
            "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/fargate-cluster",
            "serviceArn": "arn:aws:ecs:region:aws_account_id:service/fargate-cluster/
fargate-service",
            "deploymentConfiguration": {
                "maximumPercent": 200,
```

```
                "minimumHealthyPercent": 100
            },
            "createdAt": 1692283199.771,
            "schedulingStrategy": "REPLICA",
            "placementConstraints": [],
            "deployments": [
                {
                    "status": "PRIMARY",
                    "networkConfiguration": {
                        "awsvpcConfiguration": {
                            "subnets": [
                                "subnet-abcd1234"
                            ],
                            "securityGroups": [
                                "sg-abcd1234"
                            ],
                            "assignPublicIp": "ENABLED"
                        }
                    },
                    "pendingCount": 0,
                    "launchType": "FARGATE",
                    "createdAt": 1692283199.771,
                    "desiredCount": 1,
                    "taskDefinition": "arn:aws:ecs:region:aws_account_id:task-
definition/sample-fargate:1",
                    "updatedAt": 1692283199.771,
                    "platformVersion": "1.4.0",
                    "id": "ecs-svc/9223370526043414679",
                    "runningCount": 0
                }
            ],
            "serviceName": "fargate-service",
            "events": [
                {
                    "message": "(service fargate-service) has started 2 tasks: (task
 53c0de40-ea3b-489f-a352-623bf1235f08) (task d0aec985-901b-488f-9fb4-61b991b332a3).",
                    "id": "92b8443e-67fb-4886-880c-07e73383ea83",
                    "createdAt": 1510811841.408
                },
                {
                    "message": "(service fargate-service) has started 2 tasks: (task
 b4911bee-7203-4113-99d4-e89ba457c626) (task cc5853e3-6e2d-4678-8312-74f8a7d76474).",
                    "id": "d85c6ec6-a693-43b3-904a-a997e1fc844d",
                    "createdAt": 1510811601.938
```

```
                },
                {
                    "message": "(service fargate-service) has started 2 tasks: (task
 cba86182-52bf-42d7-9df8-b744699e6cfc) (task f4c1ad74-a5c6-4620-90cf-2aff118df5fc).",
                    "id": "095703e1-0ca3-4379-a7c8-c0f1b8b95ace",
                    "createdAt": 1510811364.691
                }
            ],
            "runningCount": 0,
            "status": "ACTIVE",
            "serviceRegistries": [],
            "pendingCount": 0,
            "createdBy": "arn:aws:iam::aws_account_id:user/user_name",
            "platformVersion": "LATEST",
            "placementStrategy": [],
            "propagateTags": "NONE",
            "roleArn": "arn:aws:iam::aws_account_id:role/aws-service-role/
ecs.amazonaws.com/AWSServiceRoleForECS",
            "taskDefinition": "arn:aws:ecs:region:aws_account_id:task-definition/
sample-fargate:1"
        }
    ],
    "failures": []
}
```

ステップ 7: テスト

パブリックサブネットを使用してデプロイされたタスクのテスト

タスクの Elastic Network Interface (ENI) を取得できるようにサービス内のタスクを記述します。

まず、タスク ARN を取得します。

```
aws ecs list-tasks --cluster fargate-cluster --service fargate-service
```

出力にはタスク ARN が含まれます。

```
{
    "taskArns": [
        "arn:aws:ecs:us-east-1:123456789012:task/fargate-service/EXAMPLE
    ]
}
```

タスクを記述して ENI ID を見つけます。tasks パラメータにはタスク ARN を使用します。

```
aws ecs describe-tasks --cluster fargate-cluster --tasks arn:aws:ecs:us-
east-1:123456789012:task/service/EXAMPLE
```

添付情報は出力にリストされます。

```
{
    "tasks": [
        {
            "attachments": [
                {
                    "id": "d9e7735a-16aa-4128-bc7a-b2d5115029e9",
                    "type": "ElasticNetworkInterface",
                    "status": "ATTACHED",
                    "details": [
                        {
                            "name": "subnetId",
                            "value": "subnetabcd1234"
                        },
                        {
                            "name": "networkInterfaceId",
                            "value": "eni-0fa40520aeEXAMPLE"
                        },
                    ]
                }
…
}
```

パブリック IP アドレスを取得するには ENI を記述します。

```
aws ec2 describe-network-interfaces --network-interface-id  eni-0fa40520aeEXAMPLE
```

パブリック IP アドレスが出力に表示されます。

```
{
    "NetworkInterfaces": [
        {
            "Association": {
                "IpOwnerId": "amazon",
                "PublicDnsName": "ec2-34-229-42-222.compute-1.amazonaws.com",
                "PublicIp": "198.51.100.2"
```

```
            },
…
}
```

ウェブブラウザにパブリック IP アドレスを入力すると、Amazon ECS のサンプルアプリケーションを表示するウェブページが確認できます。

プライベートサブネットを使用してデプロイされたタスクのテスト

タスクを説明し、managedAgents を見つけて、ExecuteCommandAgent が実行されていることを確認します。後で使用するために privateIPv4Address をメモしておきます。

```
aws ecs describe-tasks --cluster fargate-cluster --tasks arn:aws:ecs:us-
east-1:123456789012:task/fargate-service/EXAMPLE
```

マネージドエージェントの情報が出力にリストされます。

```
{
    "tasks": [
       {
           "attachments": [
               {
                   "id": "d9e7735a-16aa-4128-bc7a-b2d5115029e9",
                   "type": "ElasticNetworkInterface",
                   "status": "ATTACHED",
                   "details": [
                       {
                           "name": "subnetId",
                           "value": "subnetabcd1234"
                       },
                       {
                           "name": "networkInterfaceId",
                           "value": "eni-0fa40520aeEXAMPLE"
                       },
                       {
                           "name": "privateIPv4Address",
                           "value": "10.0.143.156"
                       }
                   ]
               }
           ],
    ...
```

```
    "containers": [
        {
        ...
       "managedAgents": [
                        {
                            "lastStartedAt": "2023-08-01T16:10:13.002000+00:00",
                            "name": "ExecuteCommandAgent",
                            "lastStatus": "RUNNING"
                        }
                ],
        ...
    }
```

ExecuteCommandAgent が実行されていることを確認した後、次のコマンドを実行して、タスク内のコンテナ上で対話的なシェルを実行できます。

```
aws ecs execute-command --cluster fargate-cluster \
    --task  arn:aws:ecs:us-east-1:123456789012:task/fargate-service/EXAMPLE  \
    --container  fargate-app \
    --interactive \
    --command "/bin/sh"
```

対話的なシェルの実行後、次のコマンドを実行して cURL をインストールします。

```
apt update
```

```
apt install curl
```

cURL をインストール後、先ほど取得したプライベート IP アドレスを使用して次のコマンドを実行します。

```
curl 10.0.143.156
```

Amazon ECS サンプルアプリケーションの Web ページと同等の HTML が表示されます。

```
<html>
    <head>
     <title>Amazon ECS Sample App</title>
     <style>body {margin-top: 40px; background-color: #333;} </style>
    </head>
      <body>
```

```
      <div style=color:white;text-align:center>
      <h1>Amazon ECS Sample App</h1>
      <h2>Congratulations!</h2> <p>Your application is now running on a container in
 Amazon ECS.</p>
      </div>
      </body>
</html>
```

ステップ 8: クリーンアップ

このチュートリアルが終了したら、未使用のリソースに対する料金が発生しないように、それに関連付けられたリソースをクリーンアップする必要があります。

サービスを削除します。

```
aws ecs delete-service --cluster fargate-cluster --service fargate-service --force
```

クラスターを削除します。

```
aws ecs delete-cluster --cluster fargate-cluster
```

AWS CLI を使用して、Fargate 用の Amazon ECS Windows タスクを作成する

次のステップでは、AWS CLI を使って Amazon ECS でクラスターのセットアップ、タスク定義の登録、Windows タスクの実行、その他の一般的なシナリオを実行するために役立ちます。最新バージョンの AWS CLI を使用していることを確認してください。最新のバージョンにアップグレードする方法については、「AWS CLI の最新バージョンをインストールまたは更新」を参照してください。

> **Note**
>
> デュアルスタックサービスエンドポイントを使用することで、IPv4 と IPv6 の両方を介して AWS CLI、SDK、および Amazon ECS API から Amazon ECS とやり取りできます。詳細については、「Amazon ECS デュアルスタックエンドポイントの使用」を参照してください。

トピック

- 前提条件
- ステップ 1: クラスターを作成する
- ステップ 2: Windows タスク定義を登録する
- ステップ 3: タスク定義をリスト表示する
- ステップ 4: サービスを作成する
- ステップ 5: サービスをリスト表示する
- ステップ 6: 実行中のサービスを記述する
- ステップ 7: クリーンアップする

前提条件

このチュートリアルでは、以下の前提条件をすでに満たしているものとします。

- AWS CLI の最新バージョンがインストールされ、設定されていること。AWS CLI のインストールまたはアップグレードについては、「AWS CLI の最新バージョンをインストールまたは更新」を参照してください。
- 「Amazon ECS を使用するようにセットアップする」のステップを完了していること。
- IAM ユーザーに AmazonECS_FullAccess IAM ポリシー例で指定されている必要なアクセス権限があること。
- VPC およびセキュリティグループが使用できるように作成されていること。このチュートリアルでは、Docker Hub でホストされているコンテナイメージを使用するため、タスクではインターネットにアクセスできる必要があります。タスクでインターネットへのルートを設定するには、以下のいずれかのオプションを使用します。
  - Elastic IP アドレスが割り当てられた NAT ゲートウェイでプライベートサブネットを使用する。
  - パブリックサブネットを使用してタスクにパブリック IP アドレスを割り当てる。

  詳細については、「the section called “仮想プライベートクラウドを作成する”」を参照してください。

  セキュリティグループとルールの詳細については、「Amazon Virtual Private Cloud ユーザーガイド」の「VPC のデフォルトセキュリティグループ」および「ルールの例」を参照してください。
- (任意) AWS CloudShell は、お客様が独自の EC2 インスタンスを作成する必要なく、コマンドラインを提供するツールです。詳細については、AWS CloudShell ユーザーガイド の「What is AWS CloudShell? (とは ?)」 を参照してください。

ステップ 1: クラスターを作成する

デフォルトでは、アカウントは default クラスターを受け取ります。

> **Note**
>
> お客様に提供される default クラスターを使用する利点は、後続のコマンドで --cluster *cluster_name* オプションを指定する必要がないことです。デフォルト以外の独自のクラスターを作成する場合は、そのクラスターで使用する予定のコマンドごとに --cluster *cluster_name* を指定する必要があります。

以下のコマンドを使用して、一意の名前を付けた独自のクラスターを作成します。

```
aws ecs create-cluster --cluster-name fargate-cluster
```

出力:

```
{
    "cluster": {
        "status": "ACTIVE",
        "statistics": [],
        "clusterName": "fargate-cluster",
        "registeredContainerInstancesCount": 0,
        "pendingTasksCount": 0,
        "runningTasksCount": 0,
        "activeServicesCount": 0,
        "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/fargate-cluster"
    }
}
```

ステップ 2: Windows タスク定義を登録する

Amazon ECS クラスターに Windows タスクを実行する前に、タスク定義を登録する必要があります。タスク定義とは、1 つにグループ化されたコンテナのリストです。次の例では、ウェブアプリを作成する単純なタスク定義です。使用できるタスク定義パラメータの詳細については、「Amazon ECSの タスク定義」を参照してください。

```
{
```

```
    "containerDefinitions": [
        {
            "command": ["New-Item -Path C:\\inetpub\\wwwroot\\index.html -Type file
 -Value '<html> <head> <title>Amazon ECS Sample App</title> <style>body {margin-top:
 40px; background-color: #333;} </style> </head><body> <div style=color:white;text-
align:center> <h1>Amazon ECS Sample App</h1> <h2>Congratulations!</h2> <p>Your
 application is now running on a container in Amazon ECS.</p>'; C:\\ServiceMonitor.exe
 w3svc"],
            "entryPoint": [
                "powershell",
                "-Command"
            ],
            "essential": true,
            "cpu": 2048,
            "memory": 4096,
            "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-
ltsc2019",
            "name": "sample_windows_app",
            "portMappings": [
                {
                    "hostPort": 80,
                    "containerPort": 80,
                    "protocol": "tcp"
                }
            ]
        }
    ],
    "memory": "4096",
    "cpu": "2048",
    "networkMode": "awsvpc",
    "family": "windows-simple-iis-2019-core",
    "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
    "runtimePlatform": {"operatingSystemFamily": "WINDOWS_SERVER_2019_CORE"},
    "requiresCompatibilities": ["FARGATE"]
}
```

前述の JSON の例は次の 2 通りの方法で AWS CLI に渡すことができます。1 番目はタスク定義 JSON をファイルとして保存し、`--cli-input-json file://path_to_file.json` オプションで渡すことができます。

コンテナの定義に JSON ファイルを使用するには。

```
aws ecs register-task-definition --cli-input-json file://$HOME/tasks/fargate-task.json
```

register-task-definition コマンドによって、タスク定義の登録が完了した後、その定義の説明が返されます。

ステップ 3: タスク定義をリスト表示する

list-task-definitions コマンドを使用して、いつでもアカウントのタスク定義をリスト表示できます。このコマンドの出力は、run-task または start-task をコールするときに一緒に使用する、`family` 値および `revision` 値を表示します。

```
aws ecs list-task-definitions
```

出力:

```
{
    "taskDefinitionArns": [
        "arn:aws:ecs:region:aws_account_id:task-definition/sample-fargate-windows:1"
    ]
}
```

ステップ 4: サービスを作成する

アカウントのタスクを登録したら、クラスターに登録されたタスク用のサービスを作成することができます。この例では、`sample-fargate:1` タスク定義の 1 つのインスタンスを使用して、クラスターで実行されるサービスを作成します。このタスクにはインターネットへのルートが必要であり、そのために 2 つの方法のいずれかを使用します。1 つの方法は、パブリックサブネットに Elastic IP アドレスを持つ NAT ゲートウェイで構成されたプライベートサブネットを使用することです。もう 1 つの方法は、パブリックサブネットを使用してパブリック IP アドレスをタスクに割り当てることです。次のセクションでは、両方の例を示します。

プライベートサブネットを使用した例。

```
aws ecs create-service --cluster fargate-cluster --service-name fargate-service
 --task-definition sample-fargate-windows:1 --desired-count 1 --launch-type
 "FARGATE" --network-configuration "awsvpcConfiguration={subnets=[subnet-
abcd1234],securityGroups=[sg-abcd1234]}"
```

パブリックサブネットを使用した例。

```
aws ecs create-service --cluster fargate-cluster --service-name fargate-service
 --task-definition sample-fargate-windows:1 --desired-count 1 --launch-type
```

```
 "FARGATE" --network-configuration "awsvpcConfiguration={subnets=[subnet-
abcd1234],securityGroups=[sg-abcd1234],assignPublicIp=ENABLED}"
```

create-service コマンドによって、タスク定義の登録が完了した後、その定義の説明が返されます。

ステップ 5: サービスをリスト表示する

クラスターのサービスをリスト表示します。前のセクションで作成したサービスが表示されます。このコマンドでサービス名または完全 ARN を取得できます。これを後で説明するサービスに使用します。

```
aws ecs list-services --cluster fargate-cluster
```

出力:

```
{
    "serviceArns": [
        "arn:aws:ecs:region:aws_account_id:service/fargate-cluster/fargate-service"
    ]
}
```

ステップ 6: 実行中のサービスを記述する

先に取得したサービス名を使用してタスクを記述し、サービスに関する詳細情報を取得します。

```
aws ecs describe-services --cluster fargate-cluster --services fargate-service
```

このコマンドが正常に実行されると、サービスの障害とサービスの説明が返されます。例えば、サービスセクションでは、タスクの実行中または保留中のステータスなど、デプロイに関する情報が見つかります。また、タスク定義、ネットワーク設定、タイムスタンプ付きのイベントに関する情報も見つかります。障害セクションでは、呼び出しに関連する障害がある場合にその情報が見つかります。トラブルシューティングについては、「サービスイベントメッセージ」を参照してください。サービスの説明取得の詳細については、「サービスの説明取得」を参照してください。

```
{
    "services": [
        {
            "status": "ACTIVE",
            "taskDefinition": "arn:aws:ecs:region:aws_account_id:task-definition/
sample-fargate-windows:1",
```

```
            "pendingCount": 2,
            "launchType": "FARGATE",
            "loadBalancers": [],
            "roleArn": "arn:aws:iam::aws_account_id:role/aws-service-role/
ecs.amazonaws.com/AWSServiceRoleForECS",
            "placementConstraints": [],
            "createdAt": 1510811361.128,
            "desiredCount": 2,
            "networkConfiguration": {
                "awsvpcConfiguration": {
                    "subnets": [
                        "subnet-abcd1234"
                    ],
                    "securityGroups": [
                        "sg-abcd1234"
                    ],
                    "assignPublicIp": "DISABLED"
                }
            },
            "platformVersion": "LATEST",
            "serviceName": "fargate-service",
            "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/fargate-cluster",
            "serviceArn": "arn:aws:ecs:region:aws_account_id:service/fargate-cluster/
fargate-service",
            "deploymentConfiguration": {
                "maximumPercent": 200,
                "minimumHealthyPercent": 100
            },
            "deployments": [
                {
                    "status": "PRIMARY",
                    "networkConfiguration": {
                        "awsvpcConfiguration": {
                            "subnets": [
                                "subnet-abcd1234"
                            ],
                            "securityGroups": [
                                "sg-abcd1234"
                            ],
                            "assignPublicIp": "DISABLED"
                        }
                    },
                    "pendingCount": 2,
                    "launchType": "FARGATE",
```

```
                    "createdAt": 1510811361.128,
                    "desiredCount": 2,
                    "taskDefinition": "arn:aws:ecs:region:aws_account_id:task-
definition/sample-fargate-windows:1",
                    "updatedAt": 1510811361.128,
                    "platformVersion": "0.0.1",
                    "id": "ecs-svc/9223370526043414679",
                    "runningCount": 0
                }
            ],
            "events": [
                {
                    "message": "(service fargate-service) has started 2 tasks: (task
 53c0de40-ea3b-489f-a352-623bf1235f08) (task d0aec985-901b-488f-9fb4-61b991b332a3).",
                    "id": "92b8443e-67fb-4886-880c-07e73383ea83",
                    "createdAt": 1510811841.408
                },
                {
                    "message": "(service fargate-service) has started 2 tasks: (task
 b4911bee-7203-4113-99d4-e89ba457c626) (task cc5853e3-6e2d-4678-8312-74f8a7d76474).",
                    "id": "d85c6ec6-a693-43b3-904a-a997e1fc844d",
                    "createdAt": 1510811601.938
                },
                {
                    "message": "(service fargate-service) has started 2 tasks: (task
 cba86182-52bf-42d7-9df8-b744699e6cfc) (task f4c1ad74-a5c6-4620-90cf-2aff118df5fc).",
                    "id": "095703e1-0ca3-4379-a7c8-c0f1b8b95ace",
                    "createdAt": 1510811364.691
                }
            ],
            "runningCount": 0,
            "placementStrategy": []
        }
    ],
    "failures": []
}
```

ステップ 7: クリーンアップする

このチュートリアルが終了したら、未使用のリソースに対する料金が発生しないように、それに関連付けられたリソースをクリーンアップする必要があります。

サービスを削除します。

```
aws ecs delete-service --cluster fargate-cluster --service fargate-service --force
```

クラスターを削除します。

```
aws ecs delete-cluster --cluster fargate-cluster
```

AWS CLI を使用して、EC2 用の Amazon ECS タスクを作成する

次のステップでは、AWS CLI を使って Amazon ECS でクラスターのセットアップ、タスク定義の登録、タスクの実行、その他の一般的なシナリオを実行するために役立ちます。AWS CLI の最新バージョンを使用する。最新のバージョンにアップグレードする方法については、「AWS CLI の最新バージョンをインストールまたは更新」を参照してください。

> **Note**
>
> デュアルスタックサービスエンドポイントを使用することで、IPv4 と IPv6 の両方を介して AWS CLI、SDK、および Amazon ECS API から Amazon ECS とやり取りできます。詳細については、「Amazon ECS デュアルスタックエンドポイントの使用」を参照してください。

トピック

- 前提条件
- クラスターを作成する
- Amazon ECS AMI 向けコンテナインスタンスの起動
- コンテナインスタンスを一覧表示する
- コンテナインスタンスを説明する
- タスク定義を登録する
- タスク定義をリスト表示する
- サービスを作成する
- リストサービス
- サービスを記述する
- 実行中のタスクを記述する
- ウェブサーバーをテストする。
- リソースをクリーンアップする

前提条件

このチュートリアルでは、以下の前提条件が完了済みであることを前提としています。

- AWS CLI の最新バージョンがインストールされ、設定されていること。AWS CLI のインストールまたはアップグレードについては、「AWS CLI の最新バージョンをインストールまたは更新」を参照してください。
- 「Amazon ECS を使用するようにセットアップする」のステップを完了していること。
- IAM ユーザーに AmazonECS_FullAccess IAM ポリシー例で指定されている必要なアクセス権限があること。
- 使用するコンテナインスタンス IAM ロールが作成されています。詳細については、「Amazon ECS コンテナインスタンスの IAM ロール」を参照してください。
- 使用する VPC が作成されています。詳細については、「the section called “仮想プライベートクラウドを作成する”」を参照してください。
- (任意) AWS CloudShell は、お客様が独自の EC2 インスタンスを作成する必要なく、コマンドラインを提供するツールです。詳細については、AWS CloudShell ユーザーガイド の「What is AWS CloudShell? (とは ?)」 を参照してください。

クラスターを作成する

デフォルトでは、初めてコンテナインスタンスを起動すると、アカウントが `default` クラスターを受け取ります。

> **Note**
>
> お客様に提供される `default` クラスターを使用する利点は、後続のコマンドで `--cluster` *`cluster_name`* オプションを指定する必要がないことです。デフォルト以外の独自のクラスターを作成する場合は、そのクラスターで使用する予定のコマンドごとに `--cluster` *`cluster_name`* を指定する必要があります。

以下のコマンドを使用して、一意の名前を付けた独自のクラスターを作成します。

```
aws ecs create-cluster --cluster-name MyCluster
```

出力:

```
{
    "cluster": {
        "clusterName": "MyCluster",
        "status": "ACTIVE",
        "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/MyCluster"
    }
}
```

Amazon ECS AMI 向けコンテナインスタンスの起動

コンテナインスタンスは、Amazon ECS コンテナエージェントを実行していて、クラスターに登録されている EC2 インスタンスです。このセクションでは、ECS 最適化 AMI を使用して EC2 インスタンスを起動します。

AWS CLI でコンテナインスタンスを起動するには

1. 次のコマンドを使用して、AWS リージョン の最新の ECS 最適化 Amazon Linux 2 AMI ID を取得します。このコマンドは AWS Systems Manager Parameter Store を使用して最新の ECS 最適化 AMI ID を取得します。AMI には、Amazon ECS コンテナエージェントと Docker ランタイムがプリインストールされています。

   ```
   aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/
   recommended --query 'Parameters[0].Value' --output text | jq -r '.image_id'
   ```

   出力:

   ```
   ami-abcd1234
   ```

2. コンテナインスタンスを管理するための SSH アクセスとウェブサーバーの HTTP アクセスを許可するセキュリティグループを作成します。

   ```
   aws ec2 create-security-group --group-name ecs-tutorial-sg --description "ECS
    tutorial security group"
   ```

   出力:

   ```
   {
       "GroupId": "sg-abcd1234"
   }
   ```

3. 次のコマンドを実行して、セキュリティグループにインバウンドルールを追加します。

```
aws ec2 authorize-security-group-ingress --group-id sg-abcd1234 --protocol tcp --
port 80 --cidr 0.0.0.0/0
```

出力:

```
{
    "Return": true,
    "SecurityGroupRules": [
        {
            "SecurityGroupRuleId": "sgr-efgh5678",
            "GroupId": "sg-abcd1234",
            "GroupOwnerId": "123456789012",
            "IsEgress": false,
            "IpProtocol": "tcp",
            "FromPort": 80,
            "ToPort": 80,
            "CidrIpv4": "0.0.0.0/0"
        }
    ]
}
```

セキュリティグループが、指定された IP 範囲からの SSH アクセスと、任意の場所からの HTTP アクセスを許可するようになりました。本番環境では、特定の IP アドレスへの SSH アクセスを制限し、必要に応じて HTTP アクセスの制限を検討する必要があります。

4. コンテナインスタンスへの SSH アクセス用の EC2 キーペアを作成します。

```
aws ec2 create-key-pair --key-name ecs-tutorial-key --query 'KeyMaterial' --output
 text > ecs-tutorial-key.pem
chmod 400 ecs-tutorial-key.pem
```

プライベートキーは、SSH アクセスに適切なアクセス許可を持つローカルマシンに保存されます。

5. ECS 最適化 AMI を使用して EC2 インスタンスを起動し、クラスターに参加するように設定します。

```
aws ec2 run-instances --image-id ami-abcd1234 --instance-type t3.micro --key-
name ecs-tutorial-key --security-group-ids sg-abcd1234 --iam-instance-profile
 Name=ecsInstanceRole --user-data '#!/bin/bash
echo ECS_CLUSTER=MyCluster >> /etc/ecs/ecs.config'
{
    "Instances": [
        {
            "InstanceId": "i-abcd1234",
            "ImageId": "ami-abcd1234",
            "State": {
                "Code": 0,
                "Name": "pending"
            },
            "PrivateDnsName": "",
            "PublicDnsName": "",
            "StateReason": {
                "Code": "pending",
                "Message": "pending"
            },
            "InstanceType": "t3.micro",
            "KeyName": "ecs-tutorial-key",
            "LaunchTime": "2025-01-13T10:30:00.000Z"
        }
    ]
}
```

ユーザーデータスクリプトは、インスタンスを MyCluster に登録するように Amazon ECS エージェントを設定します。インスタンスは、エージェントに必要なアクセス許可を提供する ecsInstanceRole IAM ロールを使用します。

コンテナインスタンスを一覧表示する

コンテナインスタンスを起動してから数分以内に、Amazon ECS エージェントがそのインスタンスを MyCluster クラスターに登録します。次のコマンドを実行して、クラスターのコンテナインスタンスをリスト表示できます。

```
aws ecs list-container-instances --cluster MyCluster
```

出力:

```
{
    "containerInstanceArns": [
        "arn:aws:ecs:us-east-1:aws_account_id:container-instance/container_instance_ID"
    ]
}
```

コンテナインスタンスを説明する

コンテナインスタンスの ARN または ID を取得した後、describe-container-instances コマンドを使用して、登録済みおよび残りの CPU やメモリリソースなど、インスタンスの変数情報を取得できます。

```
aws ecs describe-container-instances --cluster MyCluster --container-
instances container_instance_ID
```

出力:

```
{
    "failures": [],
    "containerInstances": [
        {
            "status": "ACTIVE",
            "registeredResources": [
                {
                    "integerValue": 1024,
                    "longValue": 0,
                    "type": "INTEGER",
                    "name": "CPU",
                    "doubleValue": 0.0
                },
                {
                    "integerValue": 995,
                    "longValue": 0,
                    "type": "INTEGER",
                    "name": "MEMORY",
                    "doubleValue": 0.0
                },
                {
                    "name": "PORTS",
                    "longValue": 0,
                    "doubleValue": 0.0,
                    "stringSetValue": [
```

```
                        "22",
                        "2376",
                        "2375",
                        "51678"
                    ],
                    "type": "STRINGSET",
                    "integerValue": 0
                },
                {
                    "name": "PORTS_UDP",
                    "longValue": 0,
                    "doubleValue": 0.0,
                    "stringSetValue": [],
                    "type": "STRINGSET",
                    "integerValue": 0
                }
            ],
            "ec2InstanceId": "instance_id",
            "agentConnected": true,
            "containerInstanceArn": "arn:aws:ecs:us-west-2:aws_account_id:container-
instance/container_instance_ID",
            "pendingTasksCount": 0,
            "remainingResources": [
                {
                    "integerValue": 1024,
                    "longValue": 0,
                    "type": "INTEGER",
                    "name": "CPU",
                    "doubleValue": 0.0
                },
                {
                    "integerValue": 995,
                    "longValue": 0,
                    "type": "INTEGER",
                    "name": "MEMORY",
                    "doubleValue": 0.0
                },
                {
                    "name": "PORTS",
                    "longValue": 0,
                    "doubleValue": 0.0,
                    "stringSetValue": [
                        "22",
                        "2376",
```

```
                    "2375",
                    "51678"
                ],
                "type": "STRINGSET",
                "integerValue": 0
            },
            {
                "name": "PORTS_UDP",
                "longValue": 0,
                "doubleValue": 0.0,
                "stringSetValue": [],
                "type": "STRINGSET",
                "integerValue": 0
            }
        ],
        "runningTasksCount": 0,
        "attributes": [
            {
                "name": "com.amazonaws.ecs.capability.privileged-container"
            },
            {
                "name": "com.amazonaws.ecs.capability.docker-remote-api.1.17"
            },
            {
                "name": "com.amazonaws.ecs.capability.docker-remote-api.1.18"
            },
            {
                "name": "com.amazonaws.ecs.capability.docker-remote-api.1.19"
            },
            {
                "name": "com.amazonaws.ecs.capability.logging-driver.json-file"
            },
            {
                "name": "com.amazonaws.ecs.capability.logging-driver.syslog"
            }
        ],
        "versionInfo": {
            "agentVersion": "1.5.0",
            "agentHash": "b197edd",
            "dockerVersion": "DockerVersion: 1.7.1"
        }
    }
]
```

```
}
```

Amazon EC2 コンソールまたは aws ec2 describe-instances --instance-id ***instance_id*** コマンドを使用して、インスタンスのモニタリングに使用できる Amazon EC2 インスタンス ID も検索することもできます。

タスク定義を登録する

Amazon ECS クラスターにタスクを実行する前に、タスク定義を登録する必要があります。タスク定義とは、1 つにグループ化されたコンテナのリストです。次の例は、nginx イメージを使用するシンプルなタスク定義です。使用できるタスク定義パラメータの詳細については、「Amazon ECSのタスク定義」を参照してください。

```
{
    "family": "nginx-task",
    "containerDefinitions": [
        {
            "name": "nginx",
            "image": "public.ecr.aws/ecs-sample-image/amazon-ecs-sample:latest",
            "cpu": 256,
            "memory": 512,
            "essential": true,
            "portMappings": [
                {
                    "containerPort": 80,
                    "hostPort": 80,
                    "protocol": "tcp"
                }
            ]
        }
    ],
    "requiresCompatibilities": ["EC2"],
    "networkMode": "bridge"
}
```

前述の JSON の例は次の 2 通りの方法で AWS CLI に渡すことができます。1 番目はタスク定義 JSON をファイルとして保存し、--cli-input-json file://***path_to_file.json*** オプションで渡すことができます。または、JSON の引用符をエスケープして JSON コンテナの定義をコマンドラインで渡すことができます。コマンドラインにコンテナ定義を渡す方法を選択した場合、追加で複数のバージョンのタスクを相互に関連付けたままにする --family パラメータが必要です。

コンテナの定義に JSON ファイルを使用するには。

```
aws ecs register-task-definition --cli-input-json file://$HOME/tasks/nginx.json
```

register-task-definition は登録を完了させた後、タスク定義の記述を返します。

```
{
    "taskDefinition": {
        "taskDefinitionArn": "arn:aws:ecs:us-east-1:123456789012:task-definition/nginx-
task:1",
        "family": "nginx-task",
        "revision": 1,
        "status": "ACTIVE",
        "containerDefinitions": [
            {
                "name": "nginx",
                "image": "public.ecr.aws/docker/library/nginx:latest",
                "cpu": 256,
                "memory": 512,
                "essential": true,
                "portMappings": [
                    {
                        "containerPort": 80,
                        "hostPort": 80,
                        "protocol": "tcp"
                    }
                ],
                "environment": [],
                "mountPoints": [],
                "volumesFrom": []
            }
        ],
        "volumes": [],
        "networkMode": "bridge",
        "compatibilities": [
            "EC2"
        ],
        "requiresCompatibilities": [
            "EC2"
        ]
    }
}
```

タスク定義をリスト表示する

list-task-definitions コマンドを使用して、いつでもアカウントのタスク定義をリスト表示できます。このコマンドの出力は、 または create-service をコールするときに一緒に使用する、family 値および revision 値を表示します。

```
aws ecs list-task-definitions
```

出力:

```
{
    "taskDefinitionArns": [
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/sleep360:1",
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/sleep360:2",
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/nginx-task:1",
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/wordpress:3",
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/wordpress:4",
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/wordpress:5",
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/wordpress:6"
    ]
}
```

サービスを作成する

アカウントのタスクを登録し、クラスターに登録されているコンテナインスタンスを起動した後、登録したタスク定義を使用して、必要な数のタスクを同時に実行および維持する Amazon ECS サービスを作成できます。この例では、MyCluster クラスター　に nginx:1 タスク定義のインスタンスを 1 つ配置します。

```
aws ecs create-service --cluster MyCluster --service-name nginx-service --task-
definition nginx-task:1 --desired-count 1
```

出力:

```
{
    "service": {
        "serviceArn": "arn:aws:ecs:us-east-1:aws_account_id:service/MyCluster/nginx-
service",
        "serviceName": "nginx-service",
        "clusterArn": "arn:aws:ecs:us-east-1:aws_account_id:cluster/MyCluster",
```

```
        "taskDefinition": "arn:aws:ecs:us-east-1:aws_account_id:task-definition/nginx-
task:1",
        "desiredCount": 1,
        "runningCount": 0,
        "pendingCount": 0,
        "launchType": "EC2",
        "status": "ACTIVE",
        "createdAt": "2025-01-13T10:45:00.000Z"
    }
}
```

リストサービス

クラスターのサービスをリスト表示します。前のセクションで作成したサービスが表示されます。このコマンドから返されたサービス ID または完全 ARN をメモし、これを後でサービスを記述するために使用できます。

```
aws ecs list-services --cluster MyCluster
```

出力:

```
{
    "taskArns": [
        "arn:aws:ecs:us-east-1:aws_account_id:task/task_ID"
    ]
}
```

サービスを記述する

次のコマンドを使用してサービスを記述し、サービスに関する詳細情報を取得します。

```
aws ecs describe-services --cluster MyCluster --services nginx-service
```

出力:

```
{
    "services": [
        {
            "serviceArn": "arn:aws:ecs:us-east-1:aws_account_id:service/MyCluster/
nginx-service",
            "serviceName": "nginx-service",
```

```
            "clusterArn": "arn:aws:ecs:us-east-1:aws_account_id:cluster/MyCluster",
            "taskDefinition": "arn:aws:ecs:us-east-1:aws_account_id:task-definition/
nginx-task:1",
            "desiredCount": 1,
            "runningCount": 1,
            "pendingCount": 0,
            "launchType": "EC2",
            "status": "ACTIVE",
            "createdAt": "2025-01-13T10:45:00.000Z",
            "events": [
                {
                    "id": "abcd1234-5678-90ab-cdef-1234567890ab",
                    "createdAt": "2025-01-13T10:45:30.000Z",
                    "message": "(service nginx-service) has started 1 tasks: (task
 abcd1234-5678-90ab-cdef-1234567890ab)."
                }
            ]
        }
    ]
}
```

実行中のタスクを記述する

サービスを記述したら、次のコマンドを実行して、サービスの一部として実行されているタスクに関する詳細情報を取得します。

```
aws ecs list-tasks --cluster MyCluster --service-name nginx-service
```

出力:

```
{
    "tasks": [
        {
            "taskArn": "arn:aws:ecs:us-east-1:aws_account_id:task/MyCluster/
abcd1234-5678-90ab-cdef-1234567890ab",
            "clusterArn": "arn:aws:ecs:us-east-1:aws_account_id:cluster/MyCluster",
            "taskDefinitionArn": "arn:aws:ecs:us-east-1:aws_account_id:task-definition/
nginx-task:1",
            "containerInstanceArn": "arn:aws:ecs:us-east-1:aws_account_id:container-
instance/MyCluster/abcd1234-5678-90ab-cdef-1234567890ab",
            "lastStatus": "RUNNING",
            "desiredStatus": "RUNNING",
```

```
            "containers": [
                {
                    "containerArn": "arn:aws:ecs:us-east-1:aws_account_id:container/
MyCluster/abcd1234-5678-90ab-cdef-1234567890ab/abcd1234-5678-90ab-cdef-1234567890ab",
                    "taskArn": "arn:aws:ecs:us-east-1:aws_account_id:task/MyCluster/
abcd1234-5678-90ab-cdef-1234567890ab",
                    "name": "nginx",
                    "lastStatus": "RUNNING",
                    "networkBindings": [
                        {
                            "bindIP": "0.0.0.0",
                            "containerPort": 80,
                            "hostPort": 80,
                            "protocol": "tcp"
                        }
                    ]
                }
            ],
            "createdAt": "2025-01-13T10:45:00.000Z",
            "startedAt": "2025-01-13T10:45:30.000Z"
        }
    ]
}
```

ウェブサーバーをテストする。

ウェブサーバーをテストするには

1. 次のコマンドを実行して、コンテナインスタンスのパブリック IP アドレスを取得します。

   ```
   aws ec2 describe-instances --instance-ids i-abcd1234 --query
    'Reservations[0].Instances[0].PublicIpAddress' --output text
   ```

   出力:

   ```
   203.0.113.25
   ```

2. IP アドレスを取得したら、IP アドレスを使用して次の curl コマンドを実行します。

   ```
   curl http://203.0.113.25
   ```

出力:

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you can see this page, the nginx web server is successfully installed and
 working.</p>
...
</body>
</html>
```

nginx ウェルカムページは、サービスが正常に実行され、インターネットからアクセス可能であることを示します。

リソースをクリーンアップする

料金の発生を回避するには、このチュートリアルで作成したリソースをクリーンアップします。

リソースをクリーンアップするには

1. 必要なタスクがゼロになるようにサービスを更新し、サービスを削除します。

```
aws ecs update-service --cluster MyCluster --service nginx-service --desired-count
 0
{
    "service": {
        "serviceArn": "arn:aws:ecs:us-east-1:123456789012:service/MyCluster/nginx-
service",
        "serviceName": "nginx-service",
        "desiredCount": 0,
        "runningCount": 1,
        "pendingCount": 0,
        "status": "ACTIVE"
    }
}
```

2. 実行中のタスクが停止するのを待ってから、サービスを削除します。

```
aws ecs delete-service --cluster MyCluster --service nginx-service
{
    "service": {
        "serviceArn": "arn:aws:ecs:us-east-1:123456789012:service/MyCluster/nginx-
service",
        "serviceName": "nginx-service",
        "status": "DRAINING"
    }
}
```

3. 作成したコンテナインスタンスを終了します。

```
aws ec2 terminate-instances --instance-ids i-abcd1234
{
    "TerminatingInstances": [
        {
            "InstanceId": "i-abcd1234",
            "CurrentState": {
                "Code": 32,
                "Name": "shutting-down"
            },
            "PreviousState": {
                "Code": 16,
                "Name": "running"
            }
        }
    ]
}
```

4. 作成したセキュリティグループとキーペアをクリーンアップします。

```
aws ec2 delete-security-group --group-id sg-abcd1234
aws ec2 delete-key-pair --key-name ecs-tutorial-key
rm ecs-tutorial-key.pem
```

5. Amazon ECS クラスターを削除します。

```
aws ecs delete-cluster --cluster MyCluster
{
    "cluster": {
        "clusterArn": "arn:aws:ecs:us-east-1:123456789012:cluster/MyCluster",
```

```
        "clusterName": "MyCluster",
        "status": "INACTIVE"
    }
}
```

CloudWatch Events イベントをリッスンするように Amazon ECS を設定する

タスクイベントをリッスンし、それを CloudWatch Logs ログストリームに書き込むシンプルな Lambda 関数を設定する方法について説明します。

前提条件: テストクラスターを設定する

イベントをキャプチャする実行中のクラスターがなければ、the section called “Fargate ワークロード用のクラスターを作成する” のステップに従ってクラスターを作成します。このチュートリアルの最後に、このクラスターでタスクを実行して Lambda 関数が正しく設定されていることをテストします。

ステップ 1: Lambda 関数を作成する

この手順では、Amazon ECS イベントストリームメッセージのターゲットとなるシンプルな Lambda 関数を作成します。

1. AWS Lambda コンソールの https://console.aws.amazon.com/lambda/ を開いてください。
2. [関数の作成] を選択してください。
3. [Author from scratch] 画面で、次の操作を行います。
   a. [名前] に値を入力します。
   b. [Runtime] (ランタイム) で、Python のバージョン (Python 3.9 など) を選択します。
   c. [Role] で、[Create a new role with basic Lambda permissions] を選択します。
4. [Create function] を選択します。
5. [Function code] (関数コード) セクションで、以下の例に一致するようにサンプルコードを編集します。

```
import json
```

```
def lambda_handler(event, context):
    if event["source"] != "aws.ecs":
       raise ValueError("Function only supports input from events with a source
 type of: aws.ecs")

    print('Here is the event:')
    print(json.dumps(event))
```

これは、Amazon ECS から送信されたイベントを出力するシンプルな Python 3.9 関数です。すべてが正しく設定されると、このチュートリアルの最後に、この Lambda 関数に関連付けられている CloudWatch Logs ログストリームにイベントの詳細が表示されます。

6. [保存] を選択します。

ステップ 2: イベントルールを登録する

次に、Amazon ECS クラスターから送信されるタスクイベントをキャプチャする CloudWatch Events イベントルールを作成します。このルールでは、それが定義されているアカウント内のすべてのクラスターから送信されるすべてのイベントがキャプチャされます。タスクメッセージ自体内に、イベントソースに関する情報 (イベントソースがあるクラスターの情報など) が含まれており、この情報を使用してプログラムでイベントをフィルタしてソートできます。

> **Note**
>
> AWS マネジメントコンソール を使用してイベントルールを作成すると、コンソールが CloudWatch Eventsに Lambda 関数を呼び出す権限を付与するために必要な IAM 権限を自動的に追加します。AWS CLI を使用してイベントルールを作成する場合は、この権限を明示的に付与する必要があります。詳細については、「Amazon EventBridge ユーザーガイド」の「Amazon EventBridge イベント」と「Amazon EventBridge のイベントパターン」を参照してください。

Lambda 関数にイベントをルートするには

1. CloudWatch コンソールの https://console.aws.amazon.com/cloudwatch/ を開いてください。
2. ナビゲーションペインで、[Events]、[Rules]、[Create rule] の順に選択します。
3. [Event Source] で、イベントソースとして [ECS] を選択します。デフォルトでは、ルールはすべての Amazon ECS グループのすべての Amazon ECS イベントに適用されます。または、特定のイベントや特定の Amazon ECS グループを選択することもできます。

4. [ターゲット] に [ターゲットの追加] を選択し、[ターゲットの種類] に [Lambda 関数] を選択したら、Lambda 関数を選択します。
5. [詳細の設定] を選択します。
6. [Rule definition] で、ルールの名前と説明を入力し、[Create rule] を選択します。

ステップ 3: タスク定義を作成する

タスク定義を作成します。

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで、[タスク定義] を選択します。
3. [Create new Task Definition] (新しいタスク定義の作成)、[Create new revision with JSON] (JSON で新しいリビジョンを作成) の順に選択します。
4. 以下のタスク定義の例をコピーしてボックスに貼り付け、[Save (保存)] を選択します。

```
{
   "containerDefinitions": [
      {
         "command": [
            "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample App</
title> <style>body {margin-top: 40px; background-color: #333;} </style> </
head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</
h1> <h2>Congratulations!</h2> <p>Your application is now running on a container in
 Amazon ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html &&
 httpd-foreground\""
         ],
         "entryPoint": [
            "sh",
            "-c"
         ],
         "essential": true,
         "image": "public.ecr.aws/docker/library/httpd:2.4",
         "logConfiguration": {
            "logDriver": "awslogs",
            "options": {
               "awslogs-group" : "/ecs/fargate-task-definition",
               "awslogs-region": "us-east-1",
               "awslogs-stream-prefix": "ecs"
            }
         },
```

```
         "name": "sample-fargate-app",
         "portMappings": [
            {
               "containerPort": 80,
               "hostPort": 80,
               "protocol": "tcp"
            }
         ]
      }
   ],
   "cpu": "256",
   "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
   "family": "fargate-task-definition",
   "memory": "512",
   "networkMode": "awsvpc",
   "runtimePlatform": {
        "operatingSystemFamily": "LINUX"
    },
   "requiresCompatibilities": [
       "FARGATE"
    ]
}
```

5. [作成] を選択します。

ステップ 4: ルールをテストする

最後に、Amazon ECS クラスターから送信されるタスクイベントをキャプチャする CloudWatch Events イベントルールを作成します。このルールでは、それが定義されているアカウント内のすべてのクラスターから送信されるすべてのイベントがキャプチャされます。タスクメッセージ自体内に、イベントソースに関する情報 (イベントソースがあるクラスターの情報など) が含まれており、この情報を使用してプログラムでイベントをフィルタしてソートできます。

ルールをテストするには

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. [Task definitions] (タスク定義) を選択します。
3. [console-sample-app-static] を選択し、[Deploy] (デプロイ)、[Run new task] (新しいタスクの実行) の順に選択します。
4. [Cluster] (クラスター) で [default] (デフォルト) を選択し、[Deploy] (デプロイ) を選択します。

5. CloudWatch コンソールの https://console.aws.amazon.com/cloudwatch/ を開いてください。
6. ナビゲーションペインで、ログ を選択して Lambda 関数 (例えば、/aws/lambda/*my-function##*)] のロググループを選択します。
7. イベントデータを表示するログストリームを選択します。

Amazon ECS タスク停止イベントに関する Amazon Simple Notification Service アラートの送信

重要なコンテナのいずれかが終了したことにより、タスクの実行が停止したタスクイベントのみをキャプチャする Amazon EventBridge イベントルールを設定します。イベントは、特定の `stoppedReason` プロパティを持つタスクイベントのみが、指定された Amazon SNS トピックに送信します。

前提条件: テストクラスターを設定する

イベントをキャプチャする実行中のクラスターがなければ、「AWS Fargate の Linux コンテナによるコンソールの使用開始」のステップに従ってクラスターを作成します。このチュートリアルの最後に、このクラスターでタスクを実行し、 Amazon SNS トピックと EventBridge ルールが正しく設定されていることをテストします。

前提条件: Amazon SNS 用のアクセス許可の設定

EventBridge による Amazon SNS トピックへの公開を許可するには、aws sns get-topic-attributes コマンドと aws sns set-topic-attributes コマンドを使用します。

アクセス許可を追加する方法については、「Amazon Simple Notification Service デベロッパーガイド」の「Amazon SNS のアクセス許可」を参照してください。

以下のアクセス許可を追加します。

```
{
  "Sid": "PublishEventsToMyTopic",
  "Effect": "Allow",
  "Principal": {
     "Service": "events.amazonaws.com"
  },
  "Action": "sns: Publish",
  "Resource": "arn:aws:sns:region:account-id:TaskStoppedAlert",
```

```
}
```

ステップ 1: Amazon SNS トピックを作成してサブスクライブする

このチュートリアルでは、新しいイベントルールのイベントターゲットとして使用する Amazon SNS トピックを設定します。

Amazon SNS トピックの作成とサブスクライブの方法については、「Amazon Simple Notification Service デベロッパーガイド」の「Amazon SNS の開始方法」を参照し、さらに、次の表から選択するオプションを決定します。

オプション	値	
タイプ	標準	
名前	TaskStoppedAlert	
プロトコル	E メール	
Endpoint	現在アクセスできる E メールアドレス	

ステップ 2: イベントルールを登録する

次に、コンテナが停止されたタスクについてのみ、タスク停止時のイベントをキャプチャするイベントルールを登録します。

Amazon SNS トピックを作成してサブスクライブする方法については、「Amazon EventBridge デベロッパーガイド」の「Amazon EventBridge でルールを作成する」を参照し、次の表を使用して、選択するオプションを決定します。

オプション	値	
ルールタイプ	イベントパターンを持つルール	

オプション	値
イベントソース	AWS イベントまたは EventBridge パートナーイベント
イベントパターン	カスタムパターン (JSON エディター)

イベントパターン	<pre>{
	  "source":[
	  "aws.ecs"
	  ],
	  "detail-type":[
	  "ECS Task State Change"
	  ],
	  "detail":{
	  "lastStatus":[
	  "STOPPED"
	  ],
	  "stoppedReason":[
	  "Essential container in task exited"
	  ]
	  }
	}</pre>

対象タイプ	AWS のサービス
ターゲット	SNS トピック
Topic	TaskStoppedAlert (ステップ 1 で作成したトピック)

ステップ 3: ルールをテストする

開始直後に終了するタスクを実行して、ルールが機能していることを確認します。イベントルールが正しく設定されていれば、数分以内にイベントテキストが記載されたメールメッセージが届きます。

ルールの要件を満たす既存のタスク定義がある場合は、それを使用してタスクを実行します。そうでない場合は、次の手順に従って Fargate タスク定義を登録し、その定義を使用してタスクを実行します。

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで、タスク定義 を選択します。
3. [Create new task definition] (新しいタスク定義の作成)、[Create new task definition with JSON] (JSON で新しいタスク定義を作成) の順に選択します。
4. JSON エディタボックスで、JSON ファイルを編集し、以下をエディタにコピーします。

```
{
   "containerDefinitions":[
      {
         "command":[
            "sh",
            "-c",
            "sleep 5"
         ],
         "essential":true,
         "image":"public.ecr.aws/amazonlinux/amazonlinux:latest",
         "name":"test-sleep"
      }
   ],
   "cpu":"256",
   "executionRoleArn":"arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
   "family":"fargate-task-definition",
   "memory":"512",
   "networkMode":"awsvpc",
   "requiresCompatibilities":[
      "FARGATE"
   ]
}
```

5. [作成] を選択します。

コンソールからタスクを実行するには

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. [クラスター] ページで、前提条件の下で作成したクラスターを選択します。
3. Tasksタブで、Run new taskを選択します。

4. Application type(アプリケーションの種類)で、Task(タスク)を選択します。
5. [タスク定義] で、[fargate-task-definition] を選択します。
6. [Desired tasks] (必要なタスク) で、起動するタスクの数を入力します。
7. [作成] を選択します。

複数行またはスタックトレースの Amazon ECS ログメッセージの連結

Fluent Bit バージョン 2.22.0 以降の AWS では、複数行フィルターが含まれています。複数行フィルターは、もともと 1 つのコンテキストに属していても、複数のレコードまたはログ行に分割されたログメッセージを連結するのに役立ちます。複数行フィルターの詳細については、「Fluent Bit documentation」(Fluent Bit ドキュメント) を参照してください。

以下は、分割されたログメッセージの一般的な例です。

- スタックトレース
- 複数の行にログを出力するアプリケーション。
- 指定されたランタイムの最大バッファサイズを超過した場合に、ログメッセージが分割されます GitHub の例、「FireLens Example: Concatenate Partial/Split Container Logs」(FireLens の例:部分/分割コンテナログの連結) に従い、コンテナランタイムによって分割されたログメッセージを連結することができます。

必要な IAM 許可

コンテナエージェントが Amazon ECR からコンテナイメージをプルし、コンテナがログを CloudWatch Logs にルーティングするために必要なIAM 許可を持っていることを確認する必要があります。

これらの許可には、次のロールが必要です。

- タスク IAM ロール
- タスク実行 IAM ロールです。

次のアクセス許可が必要です。

- `logs:CreateLogStream`

- logs:CreateLogGroup
- logs:PutLogEvents

複数行ログの設定を使用する場合を決定する

CloudWatch Logs コンソールで、デフォルトのログ設定で表示されるログス二ペットの例を次に示します。log で始まる行を見て、複数行フィルタが必要かどうかを判断できます。コンテキストが同じ場合は、複数行ログ設定を使用できます。この例の場合、コンテキストは「com.myproject.model.MyProject」です。

```
2022-09-20T15:47:56:595-05-00                              {"container_id":
 "82ba37cada1d44d389b03e78caf74faa-EXAMPLE", "container_name": "example-
app", "source=": "stdout", "log": ": "     at com.myproject.modele.
(MyProject.badMethod.java:22)",
    {
      "container_id":  "82ba37cada1d44d389b03e78caf74faa-EXAMPLE",
      "container_name: ": "example-app",
      "source": "stdout",
      "log": ": "     at com.myproject.model.MyProject.badMethod(MyProject.java:22)",
      "ecs_cluster": "default",
      "ecs_task_arn": "arn:aws:region:123456789012:task/default/
b23c940d29ed4714971cba72cEXAMPLE",
      "ecs_task_definition": "firelense-example-multiline:3"
     }
```

```
2022-09-20T15:47:56:595-05-00                              {"container_id":
 "82ba37cada1d44d389b03e78caf74faa-EXAMPLE", "container_name": "example-app", "stdout",
 "log": ": "     at com.myproject.modele.(MyProject.oneMoreMethod.java:18)",
    {
      "container_id":  "82ba37cada1d44d389b03e78caf74faa-EXAMPLE",
      "container_name: ": "example-app",
      "source": "stdout",
      "log": ": "     at
 com.myproject.model.MyProject.oneMoreMethod(MyProject.java:18)",
      "ecs_cluster": "default",
      "ecs_task_arn": "arn:aws:region:123456789012:task/default/
b23c940d29ed4714971cba72cEXAMPLE,
      "ecs_task_definition": "firelense-example-multiline:3"
     }
```

複数行のログ設定を使用すると、出力は例のようになります。

```
2022-09-20T15:47:56:595-05-00                              {"container_id":
 "82ba37cada1d44d389b03e78caf74faa-EXAMPLE", "container_name": "example-app",
 "stdout",...
    {
      "container_id":  "82ba37cada1d44d389b03e78caf74faa-EXAMPLE",
      "container_name: ": "example-app",
      "source": "stdout",
      "log:    "September 20, 2022 06:41:48 Exception in thread \"main\"
 java.lang.RuntimeException: Something has gone wrong, aborting!\n
    at com.myproject.module.MyProject.badMethod(MyProject.java:22)\n    at
    at com.myproject.model.MyProject.oneMoreMethod(MyProject.java:18)
 com.myproject.module.MyProject.main(MyProject.java:6)",
      "ecs_cluster": "default",
      "ecs_task_arn": "arn:aws:region:123456789012:task/default/
b23c940d29ed4714971cba72cEXAMPLE",
      "ecs_task_definition": "firelense-example-multiline:2"
     }
```

オプションを解析して連結する

ログを解析し、改行によって分割された行を連結するには、以下の 2 つのオプションのいずれかを使用します。

- 同じメッセージに属する行を解析および連結するルールを含む独自のパーサーファイルを作成します。
- Fluent Bit 組み込みパーサーを使用します。Fluent Bit 組み込みパーサーでサポートされている言語のリストについては、「Fluent Bit ドキュメント」を参照してください。

次のチュートリアルでは、各ユースケースの手順を説明します。この手順では、複数行を連結し、ログを Amazon CloudWatch に送信する方法について説明します。ログに別の宛先を指定できます。

例:作成したパーサーを使用する

このチュートリアルでは、次の手順を実行します。

1. Fluent Bit コンテナのイメージをビルドしてアップロードします。

2. 実行され、失敗し、複数行スタックトレースを生成するデモ複数行アプリケーションのイメージをビルドしてアップロードします。
3. タスク定義を作成して、タスクを起動する
4. ログを表示して、複数の行にまたがるメッセージが連結されて表示されることを確認します。

Fluent Bit コンテナのイメージをビルドしてアップロードします。

このイメージには、正規表現を指定するパーサーファイルと、パーサーファイルを参照する設定ファイルが含まれます。

1. FluentBitDockerImage という名前のフォルダを作成します。
2. フォルダ内で、ログを解析し、同じメッセージに属する行を連結するためのルールを含むパーサーファイルを作成します。

   a. パーサーファイルに次のコンテンツを貼り付けます。

```
[MULTILINE_PARSER]
    name          multiline-regex-test
    type          regex
    flush_timeout 1000
    #
    # Regex rules for multiline parsing
    # ---------------------------------
    #
    # configuration hints:
    #
    #  - first state always has the name: start_state
    #  - every field in the rule must be inside double quotes
    #
    # rules |   state name  | regex pattern                  | next state
    # ------|---------------|--------------------------------------------
    rule      "start_state"   "/(Dec \d+ \d+\:\d+\:\d+)(.*)/"  "cont"
    rule      "cont"          "/^\s+at.*/"                     "cont"
```

   正規表現パターンをカスタマイズするときは、正規表現エディタを使用して式をテストすることをお勧めします。

   b. parsers_multiline.conf という名前でファイルを保存します。
3. FluentBitDockerImage フォルダで、前のステップで作成したパーサーファイルを参照するカスタム設定ファイルを作成します。

カスタム設定ファイルの詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「カスタム設定ファイルの指定」を参照してください。

a. ファイルに次のコンテンツを貼り付けます。

```
[SERVICE]
    flush                 1
    log_level             info
    parsers_file          /parsers_multiline.conf

[FILTER]
    name                  multiline
    match                 *
    multiline.key_content log
    multiline.parser      multiline-regex-test
```

> **Note**
>
> パーサーの絶対パスを使用する必要があります。

b. `extra.conf` という名前でファイルを保存します。

4. `FluentBitDockerImage` フォルダで、Fluent Bit イメージと、作成したパーサーファイルと設定ファイルを使用して Dockerfile を作成します。

a. ファイルに次のコンテンツを貼り付けます。

```
FROM public.ecr.aws/aws-observability/aws-for-fluent-bit:latest

ADD parsers_multiline.conf /parsers_multiline.conf
ADD extra.conf /extra.conf
```

b. `Dockerfile` という名前でファイルを保存します。

5. Dockerfile を使用して、パーサーとカスタム設定ファイルが含まれたカスタム Fluent Bit イメージをビルドします。

> **Note**
>
> このファイルパスが FireLens によって使用される /fluent-bit/etc/fluent-bit.conf の場合を除いて、Dockerイメージの任意の場所にパーサーファイルと設定ファイルを配置できます。

a. イメージを構築します: `docker build -t fluent-bit-multiline-image.`

どこ: `fluent-bit-multiline-image` この例のイメージの名前です。

b. 次を実行して、イメージが正しく作成されたことを確認します: `docker images –filter reference=fluent-bit-multiline-image`

成功すると、出力にイメージと `latest` タグが表示されます。

6. カスタムの Fluent Bit イメージを Amazon Elastic Container Registry にアップロードします。

a. イメージを保存する Amazon ECR リポジトリを作成します: `aws ecr create-repository --repository-name fluent-bit-multiline-repo --region us-east-1`

どこ: `fluent-bit-multiline-repo` は、リポジトリの名前です。`us-east-1` はこの例のリージョンです。

出力には、新しいリポジトリの詳細が表示されます。

b. 前のステップの `docker tag` *`fluent-bit-multiline-image repositoryUri`* の値で `repositoryUri` イメージにタグを付けます。

例:`docker tag fluent-bit-multiline-image xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/fluent-bit-multiline-repo`

c. Docker イメージを実行して、正しく実行されたことを確認します: `docker images –filter reference=`*`repositoryUri`*

出力では、リポジトリ名が fluent-bit-multiline-repo から `repositoryUri` に変わります。

d. `aws ecr get-login-password` コマンドを実行し、認証先のレジストリ ID を指定して Amazon ECR を認証します: `aws ecr get-login-password | docker login --username AWS --password-stdin` *`registry ID`*`.dkr.ecr.`*`region`*`.amazonaws.com`

例:`ecr get-login-password | docker login --username AWS --password-stdin xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com`

正常にログインメッセージが表示されます。

e. Amazon ECR にイメージをプッシュします: `docker push registry ID.dkr.ecr.region.amazonaws.com/repository name`

例:`docker push xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/fluent-bit-multiline-repo`

デモ複数行アプリケーションのイメージをビルドしてアップロードする

このイメージには、アプリケーションを実行する Python スクリプトファイルと、サンプルログファイルが含まれます。

タスクを実行すると、アプリケーションは実行をシミュレートし、失敗してスタックトレースを作成します。

1. `multiline-app` という名前のフォルダを作成します: `mkdir multiline-app`
2. Python スクリプトファイルを作成します。

   a. `multiline-app` フォルダで、ファイルを作成して、`main.py` という名前を付けます。

   b. ファイルに次のコンテンツを貼り付けます。

```
import os
import time
file1 = open('/test.log', 'r')
Lines = file1.readlines()

count = 0

for i in range(10):
    print("app running normally...")
    time.sleep(1)

# Strips the newline character
for line in Lines:
    count += 1
    print(line.rstrip())
print(count)
```

```
print("app terminated.")
```

c. main.py ファイルを保存します。

3. サンプルのログファイルを作成します。

a. multiline-app フォルダで、ファイルを作成して、test.log という名前を付けます。

b. ファイルに次のコンテンツを貼り付けます。

```
single line...
Dec 14 06:41:08 Exception in thread "main" java.lang.RuntimeException:
 Something has gone wrong, aborting!
    at com.myproject.module.MyProject.badMethod(MyProject.java:22)
    at com.myproject.module.MyProject.oneMoreMethod(MyProject.java:18)
    at com.myproject.module.MyProject.anotherMethod(MyProject.java:14)
    at com.myproject.module.MyProject.someMethod(MyProject.java:10)
    at com.myproject.module.MyProject.main(MyProject.java:6)
another line...
```

c. test.log ファイルを保存します。

4. multiline-app フォルダで Dockerfile を作成します。

a. ファイルに次のコンテンツを貼り付けます。

```
FROM public.ecr.aws/amazonlinux/amazonlinux:latest
ADD test.log /test.log

RUN yum upgrade -y && yum install -y python3

WORKDIR /usr/local/bin

COPY main.py .

CMD ["python3", "main.py"]
```

b. Dockerfile ファイルを保存します。

5. Dockerfile を使用して、イメージをビルドします。

a. イメージを構築します: docker build -t multiline-app-image

どこ: multiline-app-image この例のイメージの名前です。

b. 次を実行して、イメージが正しく作成されたことを確認します: `docker images --filter reference=multiline-app-image`

成功すると、出力にイメージと `latest` タグが表示されます。

6. イメージを Amazon Elastic コンテナレジストリにアップロードします。

a. イメージを保存する Amazon ECR リポジトリを作成します: `aws ecr create-repository --repository-name multiline-app-repo --region us-east-1`

どこ: `multiline-app-repo` は、リポジトリの名前です。`us-east-1` はこの例のリージョンです。

出力には、新しいリポジトリの詳細が表示されます。次のステップで必要になるため、`repositoryUri` の値を書きとめておきます。

b. 前のステップの `docker tag` *`multiline-app-image repositoryUri`* の値で `repositoryUri` イメージにタグを付けます。

例:`docker tag multiline-app-image xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/multiline-app-repo`

c. Docker イメージを実行して、正しく実行されたことを確認します: `docker images --filter reference=`*`repositoryUri`*

出力では、リポジトリ名が `multiline-app-repo` から `repositoryUri` に変更されます。

d. Amazon ECR にイメージをプッシュします: `docker push` *`aws_account_id`*`.dkr.ecr.`*`region`*`.amazonaws.com/`*`repository name`*

例:`docker push` *`xxxxxxxxxxxx`*`.dkr.ecr.`*`us-east-1`*`.amazonaws.com/multiline-app-repo`

タスク定義を作成して、タスクを実行する

1. ファイル名 `multiline-task-definition.json` でタスク定義ファイルを作成します。
2. `multiline-task-definition.json` ファイルに次のコンテンツを貼り付けます。

```
{
    "family": "firelens-example-multiline",
    "taskRoleArn": "task role ARN,
```

```
    "executionRoleArn": "execution role ARN",
    "containerDefinitions": [
        {
            "essential": true,
            "image": "aws_account_id.dkr.ecr.us-east-1.amazonaws.com/fluent-bit-
multiline-image:latest",
            "name": "log_router",
            "firelensConfiguration": {
                "type": "fluentbit",
                "options": {
                    "config-file-type": "file",
                    "config-file-value": "/extra.conf"
                }
            },
            "memoryReservation": 50
        },
        {
            "essential": true,
            "image": "aws_account_id.dkr.ecr.us-east-1.amazonaws.com/multiline-app-
image:latest",
            "name": "app",
            "logConfiguration": {
                "logDriver": "awsfirelens",
                "options": {
                    "Name": "cloudwatch_logs",
                    "region": "us-east-1",
                    "log_group_name": "multiline-test/application",
                    "auto_create_group": "true",
                    "log_stream_prefix": "multiline-"
                }
            },
            "memoryReservation": 100
        }
    ],
    "requiresCompatibilities": ["FARGATE"],
    "networkMode": "awsvpc",
    "cpu": "256",
    "memory": "512"
}
```

multiline-task-definition.json タスク定義で以下を置き換えます:

a. *task role ARN*

タスクロールの ARN を検索するには、IAM コンソールに移動します。ロールを選択し、作成した ecs-task-role-for-firelens タスクロールを検索します。ロールを選択し、概要セクションに表示される ARN をコピーします。

b. *execution role ARN*

実行ロールの ARN を検索するには、IAM コンソールに移動します。ロールを選択し、ecsTaskExecutionRole ロールを検索します。ロールを選択し、概要セクションに表示される ARN をコピーします。

c. *aws_account_id*

aws_account_id を検索するには、AWS マネジメントコンソール にログインします。ユーザー名を選択し、アカウント ID をコピーします。

d. *us-east-1*

必要に応じてリージョンを置換します。

3. タスク定義ファイルを登録します: `aws ecs register-task-definition --cli-input-json file://multiline-task-definition.json --region region`
4. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
5. ナビゲーションペインで、[タスク定義] を選択し、前のタスク定義の最初の行でタスク定義をこのファミリに登録したため、firelens-example-multiline ファミリを選択します。
6. 最新バージョンを選択します。
7. [デプロイ]、[タスクを実行] を選択します。
8. [タスクを実行] ページの [クラスター] でクラスターを選択し、[ネットワーク] の [サブネット] で、タスクに使用できるサブネットを選択します。
9. [作成] を選択します。

Amazon CloudWatch の複数行のログメッセージが連結されて表示されることを確認する

1. CloudWatch コンソールの https://console.aws.amazon.com/cloudwatch/ を開いてください。
2. ナビゲーションペインで、[Logs (ログ)] を展開して、[Log groups (ロググループ)] を選択します。
3. multiline-test/applicatio ロググループを選択します。
4. ログを選択します。メッセージを表示します。パーサーファイル内のルールに一致する行は連結され、1 つのメッセージとして表示されます。

次のログスニペットは、単一の Java スタックトレースイベントで連結された行を示しています。

```
{
    "container_id": "xxxxxx",
    "container_name": "app",
    "source": "stdout",
    "log": "Dec 14 06:41:08 Exception in thread \"main\"
 java.lang.RuntimeException: Something has gone wrong, aborting!\n
 at com.myproject.module.MyProject.badMethod(MyProject.java:22)\n    at
 com.myproject.module.MyProject.oneMoreMethod(MyProject.java:18)\n
 at com.myproject.module.MyProject.anotherMethod(MyProject.java:14)\n
 at com.myproject.module.MyProject.someMethod(MyProject.java:10)\n    at
 com.myproject.module.MyProject.main(MyProject.java:6)",
    "ecs_cluster": "default",
    "ecs_task_arn": "arn:aws:ecs:us-east-1:xxxxxxxxxxxx:task/default/xxxxxx",
    "ecs_task_definition": "firelens-example-multiline:2"
}
```

次のログスニペットは、複数行のログメッセージを連結するように設定されていない Amazon ECS コンテナを実行する場合に、同じメッセージが 1 行でどのように表示されるかを示しています。

```
{
    "log": "Dec 14 06:41:08 Exception in thread \"main\"
 java.lang.RuntimeException: Something has gone wrong, aborting!",
    "container_id": "xxxxxx-xxxxxx",
    "container_name": "app",
    "source": "stdout",
    "ecs_cluster": "default",
    "ecs_task_arn": "arn:aws:ecs:us-east-1:xxxxxxxxxxxx:task/default/xxxxxx",
    "ecs_task_definition": "firelens-example-multiline:3"
}
```

例: Fluent Bit 組み込みパーサーを使用します。

このチュートリアルでは、次の手順を実行します。

1. Fluent Bit コンテナのイメージをビルドしてアップロードします。

2. 実行され、失敗し、複数行スタックトレースを生成するデモ複数行アプリケーションのイメージをビルドしてアップロードします。
3. タスク定義を作成して、タスクを起動する
4. ログを表示して、複数の行にまたがるメッセージが連結されて表示されることを確認します。

Fluent Bit コンテナのイメージをビルドしてアップロードします。

このイメージには、Fluent Bit パーサーを参照する設定ファイルが含まれています。

1. `FluentBitDockerImage` という名前のフォルダを作成します。
2. `FluentBitDockerImage` フォルダで、Fluent Bit 組み込みパーサーファイルを参照するカスタム設定ファイルを作成します。

   カスタム設定ファイルの詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「カスタム設定ファイルの指定」を参照してください。

   a. ファイルに次のコンテンツを貼り付けます。

   ```
   [FILTER]
       name                  multiline
       match                 *
       multiline.key_content log
       multiline.parser      go
   ```

   b. `extra.conf` という名前でファイルを保存します。
3. `FluentBitDockerImage` フォルダで、Fluent Bit イメージと、作成したパーサーファイルと設定ファイルを使用して Dockerfile を作成します。

   a. ファイルに次のコンテンツを貼り付けます。

   ```
   FROM public.ecr.aws/aws-observability/aws-for-fluent-bit:latest
   ADD extra.conf /extra.conf
   ```

   b. `Dockerfile` という名前でファイルを保存します。
4. Dockerfile を使用して、カスタム設定ファイルを含めてカスタム Fluent Bit イメージをビルドします。

> **Note**
>
> このファイルパスが FireLens によって使用される /fluent-bit/etc/fluent-bit.conf の場合を除いて、Docker イメージの任意の場所に設定ファイルを配置できます。

a. イメージを構築します: `docker build -t fluent-bit-multiline-image.`

どこ: `fluent-bit-multiline-image` この例のイメージの名前です。

b. 次を実行して、イメージが正しく作成されたことを確認します: `docker images –filter reference=fluent-bit-multiline-image`

成功すると、出力にイメージと `latest` タグが表示されます。

5. カスタムの Fluent Bit イメージを Amazon Elastic Container Registry にアップロードします。

a. イメージを保存する Amazon ECR リポジトリを作成します: `aws ecr create-repository --repository-name fluent-bit-multiline-repo --region us-east-1`

どこ: `fluent-bit-multiline-repo` は、リポジトリの名前です。`us-east-1` はこの例のリージョンです。

出力には、新しいリポジトリの詳細が表示されます。

b. 前のステップの `docker tag` *`fluent-bit-multiline-image repositoryUri`* の値で `repositoryUri` イメージにタグを付けます。

例:`docker tag fluent-bit-multiline-image xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/fluent-bit-multiline-repo`

c. Docker イメージを実行して、正しく実行されたことを確認します: `docker images –filter reference=`*`repositoryUri`*

出力では、リポジトリ名が fluent-bit-multiline-repo から `repositoryUri` に変わります。

d. `aws ecr get-login-password` コマンドを実行し、認証先のレジストリ ID を指定して Amazon ECR を認証します: `aws ecr get-login-password | docker login --username AWS --password-stdin` *`registry ID`*`.dkr.ecr.`*`region`*`.amazonaws.com`

例:`ecr get-login-password | docker login --username AWS --password-stdin xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com`

正常にログインメッセージが表示されます。

e. Amazon ECR にイメージをプッシュします: `docker push registry ID.dkr.ecr.region.amazonaws.com/repository name`

例:`docker push xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/fluent-bit-multiline-repo`

デモ複数行アプリケーションのイメージをビルドしてアップロードする

このイメージには、アプリケーションを実行する Python スクリプトファイルと、サンプルログファイルが含まれます。

1. `multiline-app` という名前のフォルダを作成します: `mkdir multiline-app`
2. Python スクリプトファイルを作成します。
   a. `multiline-app` フォルダで、ファイルを作成して、`main.py` という名前を付けます。
   b. ファイルに次のコンテンツを貼り付けます。

```
import os
import time
file1 = open('/test.log', 'r')
Lines = file1.readlines()

count = 0

for i in range(10):
    print("app running normally...")
    time.sleep(1)

# Strips the newline character
for line in Lines:
    count += 1
    print(line.rstrip())
print(count)
print("app terminated.")
```

c. main.py ファイルを保存します。

3. サンプルのログファイルを作成します。

a. multiline-app フォルダで、ファイルを作成して、test.log という名前を付けます。

b. ファイルに次のコンテンツを貼り付けます。

```
panic: my panic

goroutine 4 [running]:
panic(0x45cb40, 0x47ad70)
  /usr/local/go/src/runtime/panic.go:542 +0x46c fp=0xc42003f7b8 sp=0xc42003f710
 pc=0x422f7c
main.main.func1(0xc420024120)
  foo.go:6 +0x39 fp=0xc42003f7d8 sp=0xc42003f7b8 pc=0x451339
runtime.goexit()
  /usr/local/go/src/runtime/asm_amd64.s:2337 +0x1 fp=0xc42003f7e0
 sp=0xc42003f7d8 pc=0x44b4d1
created by main.main
  foo.go:5 +0x58

goroutine 1 [chan receive]:
runtime.gopark(0x4739b8, 0xc420024178, 0x46fcd7, 0xc, 0xc420028e17, 0x3)
  /usr/local/go/src/runtime/proc.go:280 +0x12c fp=0xc420053e30 sp=0xc420053e00
 pc=0x42503c
runtime.goparkunlock(0xc420024178, 0x46fcd7, 0xc, 0x1000f010040c217, 0x3)
  /usr/local/go/src/runtime/proc.go:286 +0x5e fp=0xc420053e70 sp=0xc420053e30
 pc=0x42512e
runtime.chanrecv(0xc420024120, 0x0, 0xc420053f01, 0x4512d8)
  /usr/local/go/src/runtime/chan.go:506 +0x304 fp=0xc420053f20 sp=0xc420053e70
 pc=0x4046b4
runtime.chanrecv1(0xc420024120, 0x0)
  /usr/local/go/src/runtime/chan.go:388 +0x2b fp=0xc420053f50 sp=0xc420053f20
 pc=0x40439b
main.main()
  foo.go:9 +0x6f fp=0xc420053f80 sp=0xc420053f50 pc=0x4512ef
runtime.main()
  /usr/local/go/src/runtime/proc.go:185 +0x20d fp=0xc420053fe0 sp=0xc420053f80
 pc=0x424bad
runtime.goexit()
  /usr/local/go/src/runtime/asm_amd64.s:2337 +0x1 fp=0xc420053fe8
 sp=0xc420053fe0 pc=0x44b4d1
```

```
goroutine 2 [force gc (idle)]:
runtime.gopark(0x4739b8, 0x4ad720, 0x47001e, 0xf, 0x14, 0x1)
  /usr/local/go/src/runtime/proc.go:280 +0x12c fp=0xc42003e768 sp=0xc42003e738
 pc=0x42503c
runtime.goparkunlock(0x4ad720, 0x47001e, 0xf, 0xc420000114, 0x1)
  /usr/local/go/src/runtime/proc.go:286 +0x5e fp=0xc42003e7a8 sp=0xc42003e768
 pc=0x42512e
runtime.forcegchelper()
  /usr/local/go/src/runtime/proc.go:238 +0xcc fp=0xc42003e7e0 sp=0xc42003e7a8
 pc=0x424e5c
runtime.goexit()
  /usr/local/go/src/runtime/asm_amd64.s:2337 +0x1 fp=0xc42003e7e8
 sp=0xc42003e7e0 pc=0x44b4d1
created by runtime.init.4
  /usr/local/go/src/runtime/proc.go:227 +0x35

goroutine 3 [GC sweep wait]:
runtime.gopark(0x4739b8, 0x4ad7e0, 0x46fdd2, 0xd, 0x419914, 0x1)
  /usr/local/go/src/runtime/proc.go:280 +0x12c fp=0xc42003ef60 sp=0xc42003ef30
 pc=0x42503c
runtime.goparkunlock(0x4ad7e0, 0x46fdd2, 0xd, 0x14, 0x1)
  /usr/local/go/src/runtime/proc.go:286 +0x5e fp=0xc42003efa0 sp=0xc42003ef60
 pc=0x42512e
runtime.bgsweep(0xc42001e150)
  /usr/local/go/src/runtime/mgcsweep.go:52 +0xa3 fp=0xc42003efd8
 sp=0xc42003efa0 pc=0x419973
runtime.goexit()
  /usr/local/go/src/runtime/asm_amd64.s:2337 +0x1 fp=0xc42003efe0
 sp=0xc42003efd8 pc=0x44b4d1
created by runtime.gcenable
  /usr/local/go/src/runtime/mgc.go:216 +0x58
one more line, no multiline
```

c. test.log ファイルを保存します。

4. multiline-app フォルダで Dockerfile を作成します。

a. ファイルに次のコンテンツを貼り付けます。

```
FROM public.ecr.aws/amazonlinux/amazonlinux:latest
ADD test.log /test.log

RUN yum upgrade -y && yum install -y python3
```

```
WORKDIR /usr/local/bin

COPY main.py .

CMD ["python3", "main.py"]
```

b. Dockerfile ファイルを保存します。

5. Dockerfile を使用して、イメージをビルドします。

a. イメージを構築します: `docker build -t multiline-app-image`

どこ: `multiline-app-image` この例のイメージの名前です。

b. 次を実行して、イメージが正しく作成されたことを確認します: `docker images –filter reference=multiline-app-image`

成功すると、出力にイメージと `latest` タグが表示されます。

6. イメージを Amazon Elastic コンテナレジストリにアップロードします。

a. イメージを保存する Amazon ECR リポジトリを作成します: `aws ecr create-repository --repository-name multiline-app-repo --region us-east-1`

どこ: `multiline-app-repo` は、リポジトリの名前です。`us-east-1` はこの例のリージョンです。

出力には、新しいリポジトリの詳細が表示されます。次のステップで必要になるため、`repositoryUri` の値を書きとめておきます。

b. 前のステップの `docker tag` *`multiline-app-image repositoryUri`* の値で `repositoryUri` イメージにタグを付けます。

例:`docker tag multiline-app-image xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/multiline-app-repo`

c. Docker イメージを実行して、正しく実行されたことを確認します: `docker images –filter reference=`*`repositoryUri`*

出力では、リポジトリ名が `multiline-app-repo` から `repositoryUri` に変更されます。

d. Amazon ECR にイメージをプッシュします: `docker push` *`aws_account_id`*`.dkr.ecr.`*`region`*`.amazonaws.com/`*`repository name`*

例:docker push *xxxxxxxxxxxx*.dkr.ecr.*us-east-1*.amazonaws.com/multiline-app-repo

タスク定義を作成して、タスクを実行する

1. ファイル名 multiline-task-definition.json でタスク定義ファイルを作成します。
2. multiline-task-definition.json ファイルに次のコンテンツを貼り付けます。

```
{
    "family": "firelens-example-multiline",
    "taskRoleArn": "task role ARN,
    "executionRoleArn": "execution role ARN",
    "containerDefinitions": [
        {
            "essential": true,
            "image": "aws_account_id.dkr.ecr.us-east-1.amazonaws.com/fluent-bit-
multiline-image:latest",
            "name": "log_router",
            "firelensConfiguration": {
                "type": "fluentbit",
                "options": {
                    "config-file-type": "file",
                    "config-file-value": "/extra.conf"
                }
            },
            "memoryReservation": 50
        },
        {
            "essential": true,
            "image": "aws_account_id.dkr.ecr.us-east-1.amazonaws.com/multiline-app-
image:latest",
            "name": "app",
            "logConfiguration": {
                "logDriver": "awsfirelens",
                "options": {
                    "Name": "cloudwatch_logs",
                    "region": "us-east-1",
                    "log_group_name": "multiline-test/application",
                    "auto_create_group": "true",
                    "log_stream_prefix": "multiline-"
                }
```

```
            },
            "memoryReservation": 100
        }
    ],
    "requiresCompatibilities": ["FARGATE"],
    "networkMode": "awsvpc",
    "cpu": "256",
    "memory": "512"
}
```

multiline-task-definition.json タスク定義で以下を置き換えます:

a. *task role ARN*

タスクロールの ARN を検索するには、IAM コンソールに移動します。ロールを選択し、作成した ecs-task-role-for-firelens タスクロールを検索します。ロールを選択し、概要セクションに表示される ARN をコピーします。

b. *execution role ARN*

実行ロールの ARN を検索するには、IAM コンソールに移動します。ロールを選択し、ecsTaskExecutionRole ロールを検索します。ロールを選択し、概要セクションに表示される ARN をコピーします。

c. *aws_account_id*

aws_account_id を検索するには、AWS マネジメントコンソール にログインします。ユーザー名を選択し、アカウント ID をコピーします。

d. *us-east-1*

必要に応じてリージョンを置換します。

3. タスク定義ファイルを登録します: `aws ecs register-task-definition --cli-input-json file://multiline-task-definition.json --region us-east-1`
4. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
5. ナビゲーションペインで、[タスク定義] を選択し、前のタスク定義の最初の行でタスク定義をこのファミリに登録したため、firelens-example-multiline ファミリを選択します。
6. 最新バージョンを選択します。
7. [デプロイ]、[タスクを実行] を選択します。
8. [タスクを実行] ページの [クラスター] でクラスターを選択し、[ネットワーク] の [サブネット] で、タスクに使用できるサブネットを選択します。

9. [作成] を選択します。

Amazon CloudWatch の複数行のログメッセージが連結されて表示されることを確認する

1. CloudWatch コンソールの https://console.aws.amazon.com/cloudwatch/ を開いてください。
2. ナビゲーションペインで、[Logs (ログ)] を展開して、[Log groups (ロググループ)] を選択します。
3. `multiline-test/applicatio` ロググループを選択します。
4. ログを選択し、メッセージを表示します。パーサーファイル内のルールに一致する行は連結され、1 つのメッセージとして表示されます。

   次のログスニペットは、単一の Go スタックトレースイベントで連結された行を示しています。

```
{
    "log": "panic: my panic\n\ngoroutine 4 [running]:\npanic(0x45cb40,
 0x47ad70)\n  /usr/local/go/src/runtime/panic.go:542 +0x46c fp=0xc42003f7b8
 sp=0xc42003f710 pc=0x422f7c\nmain.main.func1(0xc420024120)\n  foo.go:6
 +0x39 fp=0xc42003f7d8 sp=0xc42003f7b8 pc=0x451339\nruntime.goexit()\n  /usr/
local/go/src/runtime/asm_amd64.s:2337 +0x1 fp=0xc42003f7e0 sp=0xc42003f7d8
 pc=0x44b4d1\ncreated by main.main\n  foo.go:5 +0x58\n\ngoroutine 1 [chan receive]:
\nruntime.gopark(0x4739b8, 0xc420024178, 0x46fcd7, 0xc, 0xc420028e17, 0x3)\n  /usr/
local/go/src/runtime/proc.go:280 +0x12c fp=0xc420053e30 sp=0xc420053e00 pc=0x42503c
\nruntime.goparkunlock(0xc420024178, 0x46fcd7, 0xc, 0x1000f010040c217, 0x3)\n
  /usr/local/go/src/runtime/proc.go:286 +0x5e fp=0xc420053e70 sp=0xc420053e30
 pc=0x42512e\nruntime.chanrecv(0xc420024120, 0x0, 0xc420053f01, 0x4512d8)\n
  /usr/local/go/src/runtime/chan.go:506 +0x304 fp=0xc420053f20 sp=0xc420053e70
 pc=0x4046b4\nruntime.chanrecv1(0xc420024120, 0x0)\n  /usr/local/go/src/runtime/
chan.go:388 +0x2b fp=0xc420053f50 sp=0xc420053f20 pc=0x40439b\nmain.main()\n
 foo.go:9 +0x6f fp=0xc420053f80 sp=0xc420053f50 pc=0x4512ef\nruntime.main()\n
  /usr/local/go/src/runtime/proc.go:185 +0x20d fp=0xc420053fe0 sp=0xc420053f80
 pc=0x424bad\nruntime.goexit()\n  /usr/local/go/src/runtime/asm_amd64.s:2337
 +0x1 fp=0xc420053fe8 sp=0xc420053fe0 pc=0x44b4d1\n\ngoroutine 2 [force gc
 (idle)]:\nruntime.gopark(0x4739b8, 0x4ad720, 0x47001e, 0xf, 0x14, 0x1)\n  /
usr/local/go/src/runtime/proc.go:280 +0x12c fp=0xc42003e768 sp=0xc42003e738
 pc=0x42503c\nruntime.goparkunlock(0x4ad720, 0x47001e, 0xf, 0xc420000114, 0x1)\n
  /usr/local/go/src/runtime/proc.go:286 +0x5e fp=0xc42003e7a8 sp=0xc42003e768
 pc=0x42512e\nruntime.forcegchelper()\n  /usr/local/go/src/runtime/proc.go:238
 +0xcc fp=0xc42003e7e0 sp=0xc42003e7a8 pc=0x424e5c\nruntime.goexit()\n  /usr/
local/go/src/runtime/asm_amd64.s:2337 +0x1 fp=0xc42003e7e8 sp=0xc42003e7e0
 pc=0x44b4d1\ncreated by runtime.init.4\n  /usr/local/go/src/runtime/proc.go:227
 +0x35\n\ngoroutine 3 [GC sweep wait]:\nruntime.gopark(0x4739b8, 0x4ad7e0,
```

```
 0x46fdd2, 0xd, 0x419914, 0x1)\n  /usr/local/go/src/runtime/proc.go:280 +0x12c
 fp=0xc42003ef60 sp=0xc42003ef30 pc=0x42503c\nruntime.goparkunlock(0x4ad7e0,
 0x46fdd2, 0xd, 0x14, 0x1)\n  /usr/local/go/src/runtime/proc.go:286 +0x5e
 fp=0xc42003efa0 sp=0xc42003ef60 pc=0x42512e\nruntime.bgsweep(0xc42001e150)\n
  /usr/local/go/src/runtime/mgcsweep.go:52 +0xa3 fp=0xc42003efd8 sp=0xc42003efa0
 pc=0x419973\nruntime.goexit()\n  /usr/local/go/src/runtime/asm_amd64.s:2337 +0x1
 fp=0xc42003efe0 sp=0xc42003efd8 pc=0x44b4d1\ncreated by runtime.gcenable\n  /usr/
local/go/src/runtime/mgc.go:216 +0x58",
    "container_id": "xxxxxx-xxxxxx",
    "container_name": "app",
    "source": "stdout",
    "ecs_cluster": "default",
    "ecs_task_arn": "arn:aws:ecs:us-east-1:xxxxxxxxxxxx:task/default/xxxxxx",
    "ecs_task_definition": "firelens-example-multiline:2"
}
```

次のログスニペットは、複数行のログメッセージを連結するように設定されていない ECS コンテナを実行する場合に、同じイベントがどのように表示されるかを示しています。ログフィールドには 1 行が含まれます。

```
{
    "log": "panic: my panic",
    "container_id": "xxxxxx-xxxxxx",
    "container_name": "app",
    "source": "stdout",
    "ecs_cluster": "default",
    "ecs_task_arn": "arn:aws:ecs:us-east-1:xxxxxxxxxxxx:task/default/xxxxxx",
    "ecs_task_definition": "firelens-example-multiline:3"
```

> **Note**
>
> ログが標準出力ではなくログファイルに移動する場合は、フィルターではなく Tail 入力プラグインで `multiline.parser` および `multiline.key_content` 設定パラメーターを指定することをお勧めします。

Amazon ECS Windows コンテナに Fluent Bit をデプロイする

Fluent Bit とは、さまざまなオペレーティングシステムでサポートされている高速で柔軟なログプロセッサおよびルーターです。Amazon CloudWatch Logs、Firehose Amazon S3、Amazon OpenSearch Service など、さまざまな AWS の宛先にログをルーティングするために使用できます。Fluent Bit は、Datadog、Splunk、カスタム HTTP サーバーなどの一般的なパートナーソリューションをサポートしています。Fluent Bit の詳細については、Fluent Bit のウェブサイトを参照してください。

AWS for Fluent Bit イメージは、高可用性を実現するためにほとんどのリージョンの Amazon ECR Public Gallery と Amazon ECR リポジトリの両方の Amazon ECR で利用できます。この詳細については、GitHub ウェブサイトの「aws-for-fluent-bit」を参照してください。

このチュートリアルでは、Amazon ECS で実行されている Windows インスタンスに Fluent Bit コンテナをデプロイし、Windows タスクによって生成されたログを Amazon CloudWatch にストリーミングして集中型ロギングを実現する方法について説明します。

このチュートリアルでは、次のようなアプローチを使用します。

- Fluent Bit は、デーモンスケジューリングストラテジーを備えたサービスとして実行されます。この戦略により、Fluent Bit の 1 つのインスタンスが常にクラスター内のコンテナインスタンス上で実行されるようになります。
  - 転送入力プラグインを使用して、ポート 24224 でリッスンします。
  - Docker ランタイムがこの公開ポートを使用して Fluent Bit にログを送信できるように、ポート 24224 をホストに公開します。
  - Fluent Bit がログレコードを指定された宛先に送信できるようにする設定があります。
- fluentd ロギングドライバーを使用して、その他すべての Amazon ECS タスクコンテナを起動します。詳細については、Docker ドキュメントのウェブサイトの「Fluentd ロギングドライバー」を参照してください。
  - Docker は、ホスト名前空間内のローカルホスト上の TCP ソケット 24224 に接続します。
  - Amazon ECS エージェントは、クラスター名、タスク定義ファミリー名、タスク定義リビジョン番号、タスク ARN、およびコンテナ名を含むラベルをコンテナに追加します。fluentd docker ロギングドライバーの labels オプションを使用して、同じ情報がログレコードに追加されます。詳細については、Docker ドキュメントのウェブサイトの「ラベル、labels-regex、env、env、env-regex」を参照してください。
  - fluentd ロギングドライバーの `async` オプションは `true` に設定されているため、Fluent Bit コンテナを再起動すると、docker は Fluent Bit コンテナが再起動されるまでログをバッファー処

理します。fluentd-buffer-limit オプションを設定することで、バッファー制限を増やすことができます。詳細については、Docker ドキュメントのウェブサイトの「fluentd-buffer-limit」を参照してください。

ワークフローは次のとおりです。

- Fluent Bit コンテナは、ホストに公開されているポート 24224 で起動してリッスンします。
- Fluent Bit では、タスク定義で指定されたタスク IAM ロール認証情報を使用します。
- 同じインスタンスで起動された他のタスクは、fluentd docker ロギングドライバーを使用してポート 24224 にある Fluent Bit コンテナに接続します。
- アプリケーションコンテナがログを生成すると、docker ランタイムはそれらのレコードにタグを付け、ラベルに指定されたメタデータを追加して、ホスト名前空間のポート 24224 に転送します。
- Fluent Bit はホスト名前空間に公開されているため、ポート 24224 でログレコードを受信します。
- Fluent Bit は内部処理を実行し、指定されたとおりにログをルーティングします。

このチュートリアルでは、以下を行うデフォルトの CloudWatch Fluent Bit 設定を使用します。

- 各クラスターとタスク定義ファミリーごとに新しいロググループを作成します。
- 新しいタスクが開始されるたびに、以前に生成されたロググループのタスクコンテナごとに新しいログストリームを作成します。各ストリームは、コンテナが属しているタスク ID でマークされます。
- クラスター名、タスク ARN、タスクコンテナ名、タスク定義ファミリー、タスク定義リビジョン番号などのを含むメタデータを各ログエントリに追加します。

  例えば、`container_1` と `container_2` のある `task_1` および `container_3` のある task_2 の場合、CloudWatch ログストリームは次のようになります。

  - `/aws/ecs/windows.ecs_task_1`

    `task-out.TASK_ID.container_1`

    `task-out.TASK_ID.container_2`
  - `/aws/ecs/windows.ecs_task_2`

    `task-out.TASK_ID.container_3`

> **Note**
>
> デュアルスタックサービスエンドポイントを使用することで、IPv4 と IPv6 の両方を介して AWS CLI、SDK、および Amazon ECS API から Amazon ECS とやり取りできます。詳細については、「Amazon ECS デュアルスタックエンドポイントの使用」を参照してください。

Steps

- 前提条件
- ステップ 1: IAM アクセスロールを作成する
- ステップ 2: Amazon ECS Windows コンテナインスタンスを作成する
- ステップ 3: Fluent Bit の設定
- ステップ 4: ログを CloudWatch にルーティングする Windows Fluent Bit タスク定義を登録する
- ステップ 5: デーモンスケジューリング戦略を使用して ecs-windows-fluent-bit タスク定義を Amazon ECS サービスとして実行する
- ステップ 6: ログを生成する Windows タスク定義を登録する
- ステップ 7: windows-app-task タスク定義を実行する
- ステップ 8: CloudWatch でログを確認する
- ステップ 9: クリーンアップする。

前提条件

このチュートリアルでは、以下の前提条件が完了済みであることを前提としています。

- AWS CLI の最新バージョンがインストールされ、設定されていること。詳細については、「AWS CLIの最新バージョンをインストールまたは更新」を参照してください。
- `aws-for-fluent-bit` コンテナイメージは、次の Windows オペレーティングシステムで使用できます。
  - Windows Server 2019 Core
  - Windows Server 2019 Full
  - Windows Server 2022 Core
  - Windows Server 2022 Full
- 「Amazon ECS を使用するようにセットアップする」のステップを完了していること。

- クラスターがあります。このチュートリアルでは、クラスター名は [FluentBit-Cluster] です。
- EC2 インスタンスを起動するパブリックサブネットを持つ VPC があります。デフォルトの VPC を使用できます。Amazon CloudWatch エンドポイントがサブネットに到達できるようにするプライベートサブネットを使用することもできます。Amazon CloudWatch エンドポイントの詳細については、「AWS 全般のリファレンス」の「Amazon CloudWatch のエンドポイントとクォータ」を参照してください。Amazon VPC ウィザードを使用して VPC を作成する方法の詳細については、「the section called “仮想プライベートクラウドを作成する”」を参照してください。

ステップ 1: IAM アクセスロールを作成する

Amazon ECS の IAM ロールを作成します。

1. 「ecsInstanceRole」という名前の Amazon ECS コンテナインスタンスロールを作成します。詳細については、「Amazon ECS コンテナインスタンス IAM ロール」を参照してください。
2. `fluentTaskRole` と名付けられた Fluent Bit タスク用の IAM ロールを作成します。詳細については、「the section called “タスク IAM ロール”」を参照してください。

   この IAM ロールで付与される IAM 許可は、タスクコンテナによって引き受けられます。Fluent Bit が CloudWatch にログを送信できるようにするには、タスク IAM ロールに次の権限をアタッチする必要があります。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "logs:CreateLogStream",
            "logs:CreateLogGroup",
            "logs:DescribeLogStreams",
            "logs:PutLogEvents"
        ],
        "Resource": "*"
    }
    ]
}
```

3. ロールへのポリシーの付与

   a. `fluent-bit-policy.json` という名前のファイルに前述の内容を保存します。

   b. `fluentTaskRole` IAM ロールにインラインポリシーをアタッチするには、次のコマンドを実行します。

```
aws iam put-role-policy --role-name fluentTaskRole --policy-name
 fluentTaskPolicy --policy-document file://fluent-bit-policy.json
```

ステップ 2: Amazon ECS Windows コンテナインスタンスを作成する

Amazon ECS Windows コンテナインスタンスを作成します。

Amazon ECS インスタンスを作成するには

1. `aws ssm get-parameters` コマンドを使用して、VPC をホストするリージョン用の AMI ID を取得します。詳細については、「Amazon ECS に最適化された AMI メタデータを取得する」を参照してください。
2. Amazon EC2 コンソールを使用して、インスタンスを起動します。

   a. Amazon EC2 コンソールの https://console.aws.amazon.com/ec2/ を開いてください。

   b. ナビゲーションバーから、使用するリージョンを選択します。

   c. EC2 ダッシュボードから、[Launch Instance] を選択します。

   d. [Name (名前)] に一意の名前を入力します。

   e. [Application and OS Images (Amazon Machine Image)] (アプリケーションおよび OS イメージ (Amazon マシンイメージ)) で、最初の手順で取得した AMI を選択します。

   f. [Instance type (インスタンスタイプ)] として [`t3.xlarge`] を選択します。

   g. [Key pair (login)] (キーペア (ログイン)) には、キーペアを選択します。

   h. [Network settings] (ネットワーク設定) にある [Security group] (セキュリティグループ) には、既存のセキュリティグループを選択することも、新しいセキュリティグループを作成することもできます。

   i. [Network settings] (ネットワーク設定) の [Auto-assign Public IP] (パブリック IP の自動割り当て) で、[Enable] (有効にする) を選択します。

   j. [Advanced details] (高度な詳細) で、[IAM instance profile] (IAM インスタンスプロファイル) として [ecsInstanceRole] を選択します。

k. 次のユーザーデータを使用して、Amazon ECS コンテナインスタンスを設定します。[Advanced Details] (詳細情報) にある [User data] (ユーザーデータ) フィールドに以下のスクリプトを貼り付け、*cluster_name* をクラスターの名前に置き換えます。

```
<powershell>
Import-Module ECSTools
Initialize-ECSAgent -Cluster cluster-name -EnableTaskENI -EnableTaskIAMRole -
LoggingDrivers '["awslogs","fluentd"]'
</powershell>
```

l. 準備ができたら、確認フィールドを選択してから、[Launch Instances] を選択します。

m. 確認ページは、インスタンスが起動中であることを通知します。[View Instances] (インスタンスを表示) を選択して確認ページを閉じ、コンソールに戻ります。

ステップ 3: Fluent Bit の設定

AWS に用意されている次のデフォルト設定を使用すると、すぐに開始できます。

- Amazon CloudWatch。「Fluent Bit 公式マニュアル」にある Amazon CloudWatch 用 Fluent Bit プラグインをベースにしています。

または、AWS で提供されている他のデフォルト設定を使用することもできます。詳細については、`aws-for-fluent-bit` Github ウェブサイトの「Windows イメージのエントリポイントのオーバーライド」を参照してください。

デフォルトの Amazon CloudWatch Fluent Bit 設定を次に示します。

以下の変数を置き換えます。

- *region* を、Amazon CloudWatch ログを送信したいリージョンで置き換えます。

```
[SERVICE]
    Flush               5
    Log_Level           info
    Daemon              off

[INPUT]
    Name                forward
    Listen              0.0.0.0
```

```
    Port                24224
    Buffer_Chunk_Size   1M
    Buffer_Max_Size     6M
    Tag_Prefix          ecs.

# Amazon ECS agent adds the following log keys as labels to the docker container.
# We would use fluentd logging driver to add these to log record while sending it to
 Fluent Bit.
[FILTER]
    Name                modify
    Match               ecs.*
    Rename              com.amazonaws.ecs.cluster ecs_cluster
    Rename              com.amazonaws.ecs.container-name ecs_container_name
    Rename              com.amazonaws.ecs.task-arn ecs_task_arn
    Rename              com.amazonaws.ecs.task-definition-family
 ecs_task_definition_family
    Rename              com.amazonaws.ecs.task-definition-version
 ecs_task_definition_version

[FILTER]
    Name                rewrite_tag
    Match               ecs.*
    Rule                $ecs_task_arn ^([a-z-:0-9]+)/([a-zA-Z0-9-_]+)/([a-z0-9]+)$
 out.$3.$ecs_container_name false
    Emitter_Name        re_emitted

[OUTPUT]
    Name                cloudwatch_logs
    Match               out.*
    region              region
    log_group_name      fallback-group
    log_group_template  /aws/ecs/$ecs_cluster.$ecs_task_definition_family
    log_stream_prefix   task-
    auto_create_group   On
```

Fluent Bit に入るすべてのログには、ユーザーが指定したタグが付いているか、指定しない場合は自動的に生成されます。タグを使用して、さまざまなログをさまざまな宛先にルーティングできます。追加情報については、「Fluent Bit 公式マニュアル」の「タグ」を参照してください。

前述の Fluent Bit 設定は以下のようなプロパティを持ちます。

- 転送入力プラグインは TCP ポート 24224 で受信トラフィックをリッスンします。

- ポートで受信した各ログエントリにはタグがあり、転送入力プラグインはこれを変更してレコードのプレフィックスに `ecs.` 文字列を付けます。
- Fluent Bit の内部パイプラインは、Match regex を使用してログエントリをルーティングし、フィルターを変更します。このフィルターは、ログレコード JSON のキーを Fluent Bit が使用できる形式に置き換えます。
- 変更されたログエントリは、その後 rewrite_tag フィルターによって使用されます。このフィルターは、ログレコードのタグを out.*TASK_ID*。*CONTAINER_NAME* 形式に変更します。
- 新しいタグは、CloudWatch 出力プラグインの `log_group_template` オプションおよび `log_stream_prefix` オプションを使用して前述のようにロググループとストリームを作成する出力 cloudwatch_logs プラグインにルーティングされます。追加情報については、「Fluent Bit 公式マニュアル」の「設定パラメータ」を参照してください。

ステップ 4: ログを CloudWatch にルーティングする Windows Fluent Bit タスク定義を登録する

ログを CloudWatch にルーティングする Windows Fluent Bit タスク定義を登録します。

> **Note**
>
> このタスク定義では、Fluent Bit コンテナポート 24224 をホストポート 24224 に公開します。外部からのアクセスを防ぐため、EC2 インスタンスのセキュリティグループでこのポートが開いていないことを確認してください。

タスク定義を登録するには

1. 次の内容で、`fluent-bit.json` という名前のファイルを作成します。

   以下の変数を置き換えます。

   - *task-iam-role*。タスク IAM ロールの Amazon リソースネーム (ARN) に置き換えます。
   - *region*。タスクが実行されるリージョンに置き換えます。

```
{
  "family": "ecs-windows-fluent-bit",
  "taskRoleArn": "task-iam-role",
  "containerDefinitions": [
```

```
    {
      "name": "fluent-bit",
      "image": "public.ecr.aws/aws-observability/aws-for-fluent-
bit:windowsservercore-latest",
      "cpu": 512,
      "portMappings": [
        {
          "hostPort": 24224,
          "containerPort": 24224,
          "protocol": "tcp"
        }
      ],
      "entryPoint": [
        "Powershell",
        "-Command"
      ],
      "command": [
        "C:\\entrypoint.ps1 -ConfigFile C:\\ecs_windows_forward_daemon\
\cloudwatch.conf"
      ],
      "environment": [
        {
          "name": "AWS_REGION",
          "value": "region"
        }
      ],
      "memory": 512,
      "essential": true,
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "/ecs/fluent-bit-logs",
          "awslogs-region": "region",
          "awslogs-stream-prefix": "flb",
          "awslogs-create-group": "true"
        }
      }
    }
  ],
  "memory": "512",
  "cpu": "512"
}
```

2. 次のコマンドを実行して、タスク定義を登録します。

```
aws ecs register-task-definition --cli-input-json file://fluent-bit.json --
region region
```

list-task-definitions コマンドを実行して、アカウントのタスク定義をリスト表示できます。出力には、run-task または start-task と一緒に使用できるファミリーとリビジョンの値が表示されます。

ステップ 5: デーモンスケジューリング戦略を使用して ecs-windows-fluent-bit タスク定義を Amazon ECS サービスとして実行する

アカウントのタスク定義を登録したら、クラスターでタスクを実行できます。このチュートリアルでは、FluentBit-cluster クラスターで ecs-windows-fluent-bit:1 タスク定義のインスタンスを 1 つ実行します。デーモンスケジューリング戦略を使用するサービスでタスクを実行することにより、Fluent Bit の単一インスタンスが常に各コンテナインスタンスで実行されます。

タスクを実行するには

1. 以下のコマンドを実行して、ecs-windows-fluent-bit:1 タスク定義 (前のステップで登録済み) をサービスとして起動します。

> **Note**
>
> このタスク定義は awslogs ロギングドライバーを使用するため、コンテナインスタンスには必要な権限が必要です。

以下の変数を置き換えます。

- *region*。サービスが実行されるリージョンに置き換えます。

```
aws ecs create-service \
    --cluster FluentBit-cluster \
    --service-name FluentBitForwardDaemonService \
    --task-definition ecs-windows-fluent-bit:1 \
    --launch-type EC2 \
    --scheduling-strategy DAEMON \
```

```
    --region region
```

2. 次のコマンドを実行して、タスクを一覧表示します。

   以下の変数を置き換えます。

   - *region*。サービスタスクが実行されるリージョンに置き換えます。

```
aws ecs list-tasks --cluster FluentBit-cluster --region region
```

ステップ 6: ログを生成する Windows タスク定義を登録する

ログを生成するタスク定義を登録します。このタスク定義では、1 秒ごとに増分数を stdout に書き込む Windows コンテナイメージをデプロイします。

タスク定義では、Fluent Bit プラグインがリッスンするポート 24224 に接続する fluentd ロギングドライバーを使用します。Amazon ECS エージェントは、クラスター名、タスク ARN、タスク定義ファミリー名、タスク定義リビジョン番号、タスクコンテナ名などを含むタグのある各 Amazon ECS コンテナにラベルを付けます。これらのキー値ラベルは Fluent Bit に渡されます。

> **Note**
>
> このタスクは、default ネットワークモードを使用します。ただし、タスクで awsvpc ネットワークモードを使用することもできます。

タスク定義を登録するには

1. 次の内容で、windows-app-task.json という名前のファイルを作成します。

```
{
  "family": "windows-app-task",
  "containerDefinitions": [
    {
      "name": "sample-container",
      "image": "mcr.microsoft.com/windows/servercore:ltsc2019",
      "cpu": 512,
      "memory": 512,
      "essential": true,
```

```
      "entryPoint": [
        "Powershell",
        "-Command"
      ],
      "command": [
        "$count=1;while(1) { Write-Host $count; sleep 1; $count=$count+1;}"
      ],
      "logConfiguration": {
        "logDriver": "fluentd",
        "options": {
          "fluentd-address": "localhost:24224",
          "tag": "{{ index .ContainerLabels \"com.amazonaws.ecs.task-definition-
family\" }}",
          "fluentd-async": "true",
          "labels": "com.amazonaws.ecs.cluster,com.amazonaws.ecs.container-
name,com.amazonaws.ecs.task-arn,com.amazonaws.ecs.task-definition-
family,com.amazonaws.ecs.task-definition-version"
        }
      }
    }
  ],
  "memory": "512",
  "cpu": "512"
}
```

2. 次のコマンドを実行して、タスク定義を登録します。

   以下の変数を置き換えます。

   - *region*。タスクが実行されるリージョンに置き換えます。

   ```
   aws ecs register-task-definition --cli-input-json file://windows-app-task.json --
   region region
   ```

   list-task-definitions コマンドを実行して、アカウントのタスク定義をリスト表示できます。出力には、run-task または start-task と一緒に使用できるファミリーとリビジョンの値が表示されます。

ステップ 7: **windows-app-task** タスク定義を実行する

windows-app-task タスク定義を登録したら、FluentBit-cluster クラスターで実行します。

タスクを実行するには

1. 前の手順で登録した windows-app-task:1 タスク定義を実行します。

   以下の変数を置き換えます。

   - *region*。タスクが実行されるリージョンに置き換えます。

```
aws ecs run-task --cluster FluentBit-cluster --task-definition windows-app-task:1
 --count 2 --region region
```

2. 次のコマンドを実行して、タスクを一覧表示します。

```
aws ecs list-tasks --cluster FluentBit-cluster
```

ステップ 8: CloudWatch でログを確認する

Fluent Bit の設定を確認するには、CloudWatch コンソールで次のロググループを確認します。

- /ecs/fluent-bit-logs - これは、コンテナインスタンスで実行されている Fluent Bit デーモンコンテナに対応するロググループです。
- /aws/ecs/FluentBit-cluster.windows-app-task - これは、FluentBit-cluster クラスター内の windows-app-task タスク定義ファミリーで起動されたすべてのタスクに対応するロググループです。

  task-out.*FIRST_TASK_ID*.sample-container - このログストリームには、sample-container タスクコンテナ内のタスクの最初のインスタンスによって生成されたすべてのログが含まれます。

  task-out.*SECOND_TASK_ID*.sample-container - このログストリームには、sample-container タスクコンテナ内のタスクの 2 番目のインスタンスによって生成されたすべてのログが含まれます。

task-out.*TASK_ID*.sample-container ログストリームには次のようなフィールドがあります。

```
{
    "source": "stdout",
```

```
    "ecs_task_arn": "arn:aws:ecs:region:0123456789012:task/FluentBit-
cluster/13EXAMPLE",
    "container_name": "/ecs-windows-app-task-1-sample-container-cEXAMPLE",
    "ecs_cluster": "FluentBit-cluster",
    "ecs_container_name": "sample-container",
    "ecs_task_definition_version": "1",
    "container_id": "61f5e6EXAMPLE",
    "log": "10",
    "ecs_task_definition_family": "windows-app-task"
}
```

Fluent Bit 設定を検証するには

1. CloudWatch コンソールの https://console.aws.amazon.com/cloudwatch/ を開いてください。
2. ナビゲーションペインで、[Log groups] (ロググループ) を選択します。Fluent Bit をコンテナにデプロイしたリージョンにいることを確認してください。

   AWS リージョンのロググループのリストは、以下のように表示されます。

   - `/ecs/fluent-bit-logs`
   - `/aws/ecs/FluentBit-cluster.windows-app-task`

   これらのロググループが表示される場合、Fluent Bit のセットアップは検証済みです。

ステップ 9: クリーンアップする。

このチュートリアルが終了したら、使用していないリソースに対する料金が発生しないように、チュートリアルに関連付けられたリソースをクリーンアップします。

チュートリアルリソースをクリーンアップするには

1. `windows-simple-task` タスクおよび `ecs-fluent-bit` タスクを停止します。詳細については、「the section called “タスクの停止”」を参照してください。
2. 次のコマンドを実行して、`/ecs/fluent-bit-logs` ロググループを削除します。ロググループの削除に関する詳細については、「AWS Command Line Interfaceリファレンス」の「delete-log-group」を参照してください。

   ```
   aws logs delete-log-group --log-group-name /ecs/fluent-bit-logs
   ```

```
aws logs delete-log-group --log-group-name /aws/ecs/FluentBit-cluster.windows-app-task
```

3. 次のコマンドを実行して、インスタンスを停止します。

```
aws ec2 terminate-instances --instance-ids instance-id
```

4. 次のコマンドを実行して、IAM ロールを削除します。

```
aws iam delete-role --role-name ecsInstanceRole
aws iam delete-role --role-name fluentTaskRole
```

5. 次のコマンドを実行して、Amazon ECS クラスターを削除します。

```
aws ecs delete-cluster --cluster FluentBit-cluster
```

Amazon EC2 の Linux コンテナで gMSA を使用する

Amazon ECS は、グループ管理サービスアカウント (gMSA) と呼ばれる特殊なサービスアカウントを使用して、EC2 上の Linux コンテナの Active Directory 認証をサポートします。

Linux Core アプリケーションなどの .NET ベースのネットワークアプリケーションは、Active Directory を使用して、ユーザーとサービス間の認証と認可の管理を円滑化することができます。この機能は、Active Directory と統合され、ドメインに参加しているサーバー上で実行されるアプリケーションを設計することで利用できます。しかし、Linux コンテナはドメインに参加できないため、gMSA を使用して実行する Linux コンテナを設定する必要があります。

gMSA で実行される Linux コンテナは、コンテナのホスト Amazon EC2 インスタンスで実行される `credentials-fetcher` デーモンに依存します。つまり、デーモンは Active Directory ドメインコントローラーからも認証情報を取得し、これらの gMSA 認証情報をコンテナインスタンスに転送します。サービスアカウントの詳細については、Microsoft Learn Web サイトの「Windows コンテナ向け gMSAs の作成」を参照してください。

考慮事項

Linux コンテナ向け gMSA を使用する前に、次の点を考慮してください。

- コンテナが EC2 で実行されている場合は、Windows コンテナおよびLinux コンテナ向け gMSA を使用できます。Fargate で Linux コンテナに gMSA を使用する方法については、Fargate で Linux コンテナに gMSA を使用する を参照してください。
- 前提条件を完全に満たすには、ドメインに参加している Windows コンピュータが必要になる場合があります。例えば、PowerShell を使用して Active Directory で gMSA を作成するには、ドメインに参加している Windows コンピュータが必要になる場合があります。RSAT Active Director PowerShell ツールは Windows でのみ使用できます。詳細については、「Active Directory 管理ツールのインストール」を参照してください。
- ドメインレス gMSAか各インスタンスを単一のドメインに結合するを選択しました。ドメインレス gMSA を使用すると、コンテナインスタンスはドメインに参加せず、インスタンス上の他のアプリケーションは認証情報を使用してドメインにアクセスできなくなり、異なるドメインに参加するタスクを同じインスタンス上で実行できます。

  次に、CredSpec のデータ ストレージを選択し、必要に応じて、ドメインレス gMSA の Active Directory ユーザー認証情報を選択します。

  Amazon ECS は Active Directory の認証情報仕様ファイル (CredSpec) を使用します。このファイルは、gMSA アカウントコンテキストをコンテナに伝達するために使用される gMSA メタデータを含む認証情報仕様ファイルを使用します。CredSpec ファイルを生成し、コンテナインスタンスのオペレーティングシステムに応じて、次の表にある CredSpec ストレージオプションのいずれかに保存します。ドメインレス方式を使用するには、CredSpec ファイル内のオプションのセクションで、コンテナインスタンスのオペレーティングシステムに固有の、次の表の domainless user credentials ストレージオプションのいずれかの認証情報を指定できます。

ストレージの場所	Linux	Windows
Amazon Simple Storage Service	CredSpec	CredSpec
AWS Secrets Manager	ドメインレスユーザー認証情報	ドメインレスユーザー認証情報
Amazon EC2 Systems Manager Parameter Store	CredSpec	CredSpec、ドメインレスユーザー認証情報
ローカルファイル	該当なし	CredSpec

前提条件

gMSA を使用する前に、Amazon ECS の Linux コンテナ機能のために必ず以下を完了してください。

- コンテナがアクセスするリソースを含む Active Directory ドメインを設定します。Amazon ECS は以下の設定をサポートしています。
  - Directory Service Active Directory。Directory Service は、Amazon EC2 でホストされる AWS マネージド Active Directory です。詳細については、「AWS Directory Service 管理ガイド」の「AWS Managed Microsoft AD の開始方法」を参照してください。
  - オンプレミスの Active Directory。Amazon ECS Linux コンテナインスタンスがドメインに参加できることを確認する必要があります。詳細については、「AWS Direct Connect」を参照してください。
- Active Directory に既存の gMSA アカウントがあります。詳細については、「Amazon EC2 の Linux コンテナで gMSA を使用する」を参照してください。
- Amazon ECS Linux コンテナインスタンスに `credentials-fetcher` デーモンをインストールして実行しています。また、Active Directory で認証するための初期資格情報セットを `credentials-fetcher` デーモンに追加しました。

  > **Note**
  >
  > `credentials-fetcher` デーモンは、Amazon Linux 2023 および Fedora 37 以降でのみ使用できます。デーモン は Amazon Linux 2 では使用できません。詳細については、GitHub の「aws/credentials-fetcher」を参照してください。

- Active Directory で認証するための `credentials-fetcher` デーモンの認定情報を設定します。認証情報は、gMSA アカウントにアクセスできる Active Directory セキュリティグループのメンバーである必要があります。インスタンスをドメインに参加させるか、ドメインレス gMSA を使用するかを決定します。 には複数のオプションがあります。
- 必須の IAM アクセス権限を追加しました。必要な権限は、初期認証情報と認証情報の仕様の保存に選択した方法によって異なります。
  - 初期認証情報にドメインレス gMSA を使用する場合は、タスク実行ロールに AWS Secrets Manager の IAM アクセス許可が必要です。
  - 認証情報の仕様を SSM Parameter Store に保存する場合は、タスク実行ロールで Amazon EC2 Systems Manager Parameter Store に対する IAM 許可が必要です。

- 認証情報の仕様を Amazon S3 で保存する場合は、タスク実行ロールで Amazon Simple Storage Service の IAM 許可が必要です。

Amazon ECS での gMSA 対応の Linux コンテナの設定

インフラストラクチャを準備する

次のステップは、考慮事項と 1 回実行される設定です。これらのステップを完了したら、コンテナインスタンスの作成を自動化して、この設定を再利用できます。

初期認証情報の提供方法を決定し、再利用可能な EC2 起動テンプレートで EC2 ユーザーデータを設定して、`credentials-fetcher` デーモンをインストールします。

1. インスタンスをドメインに参加させるか、ドメインレス gMSA を使用するかを決定します。
   - EC2 インスタンスを Active Directory ドメインに参加させる
     - ユーザーデータを使用してインスタンスを参加させる

       Active Directory ドメインを EC2 起動テンプレートの EC2 ユーザーデータに参加させるステップを追加します。複数の Amazon EC2 Auto Scaling グループが同じ起動テンプレートを使用できます。

       Fedora ドキュメントの「Active Directory または FreeIPA ドメインへの参加」のこれらのステップを使用できます。
   - ドメインレス gMSA 用に Active Directory ユーザーを作成する

     `credentials-fetcher` デーモンには、ドメインレス gMSA と呼ばれる機能があります。この機能にはドメインが必要ですが、EC2 インスタンスをドメインに参加させる必要はありません。ドメインレス gMSA を使用すると、コンテナインスタンスはドメインに参加せず、インスタンス上の他のアプリケーションは認証情報を使用してドメインにアクセスできなくなり、異なるドメインに参加するタスクを同じインスタンス上で実行できます。代わりに、CredSpec ファイルの AWS Secrets Manager にシークレットの名前を指定します。シークレットには、ユーザー名、パスワード、ログイン先のドメインが含まれている必要があります。

     この機能はサポートされており、Linux および Windows コンテナで使用できます。

この機能は gMSA support for non-domain-joined container hosts 機能に似ています。Windows の機能の詳細については、Microsoft Learn ウェブサイトの「gMSA アーキテクチャと改善」を参照してください。

a. Active Directory ドメインでユーザーを作成します。Active Directory のユーザーは、タスクで使用する gMSA サービスアカウントにアクセスするための許可を持っている必要があります。

b. Active Directory でユーザーを作成した後、AWS Secrets Manager でシークレットを作成します。詳細については、「AWS Secrets Manager シークレットを作成する」を参照してください。

c. ユーザーのユーザー名、パスワード、ドメインを、それぞれ `username`、`password`、`domainName` と呼ばれる JSON キーと値のペアに入力します。

```
{"username":"username","password":"passw0rd", "domainName":"example.com"}
```

d. サービスアカウントの CredSpec ファイルに設定を追加します。追加の `HostAccountConfig` には、Secrets Manager のシークレットの Amazon リソースネーム (ARN) が含まれています。

Windows では、`PluginGUID` は次のスニペットの例の GUID と一致する必要があります。Linux では、`PluginGUID` は無視されます。`MySecret` の例を、シークレットの Amazon リソースネーム (ARN) に置き換えます。

```
    "ActiveDirectoryConfig": {
        "HostAccountConfig": {
            "PortableCcgVersion": "1",
            "PluginGUID": "{859E1386-BDB4-49E8-85C7-3070B13920E1}",
            "PluginInput": {
                "CredentialArn": "arn:aws:secretsmanager:aws-
region:111122223333:secret:MySecret"
            }
        }
```

e. ドメインレス gMSA 機能には、タスク実行ロールの追加のアクセス許可が必要です。ステップ (オプション) ドメインレス gMSA シークレット に従います。

2. インスタンスを設定し、**credentials-fetcher** デーモンをインストールする

   EC2 起動テンプレートでユーザーデータスクリプトを使用して credentials-fetcher デーモンをインストールできます。次の例は、2 種類のユーザーデータ (cloud-config YAML または bash スクリプト) を示しています。これらの例は、Amazon Linux 2023 (AL2023) についてのものです。MyCluster を、これらのインスタンスが参加する Amazon ECS クラスターの名前に置き換えます。

   - **cloud-config** YAML

     ```
     Content-Type: text/cloud-config
     package_reboot_if_required: true
     packages:
       # prerequisites
       - dotnet
       - realmd
       - oddjob
       - oddjob-mkhomedir
       - sssd
       - adcli
       - krb5-workstation
       - samba-common-tools
       # https://github.com/aws/credentials-fetcher gMSA credentials management for
      containers
       - credentials-fetcher
     write_files:
     # configure the ECS Agent to join your cluster.
     # replace MyCluster with the name of your cluster.
     - path: /etc/ecs/ecs.config
       owner: root:root
       permissions: '0644'
       content: |
         ECS_CLUSTER=MyCluster
         ECS_GMSA_SUPPORTED=true
     runcmd:
     # start the credentials-fetcher daemon and if it succeeded, make it start after
      every reboot
     - "systemctl start credentials-fetcher"
     - "systemctl is-active credentials-fetcher && systemctl enable credentials-
     fetcher"
     ```

- bash スクリプト

  bash スクリプトに慣れていて、/etc/ecs/ecs.config に書き込む変数が複数ある場合は、次の heredoc 形式を使用します。この形式は cat で始まる行と EOF の間のすべてを設定ファイルに書き込みます。

```
#!/usr/bin/env bash
set -euxo pipefail

# prerequisites
timeout 30 dnf install -y dotnet realmd oddjob oddjob-mkhomedir sssd adcli
 krb5-workstation samba-common-tools
# install https://github.com/aws/credentials-fetcher gMSA credentials
 management for containers
timeout 30 dnf install -y credentials-fetcher

# start credentials-fetcher
systemctl start credentials-fetcher
systemctl is-active credentials-fetcher && systemctl enable credentials-fetcher

cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_GMSA_SUPPORTED=true
EOF
```

/etc/ecs/ecs.config で設定できる credentials-fetcher デーモンのオプションの設定変数があります。YAML ブロック内のユーザーデータに変数を設定するか、heredoc前の例と同様に変数を設定することをお勧めします。これにより、1 つのファイルを複数回編集した場合に発生する、部分的な設定に関する問題を回避できます。ECS エージェントの設定の詳細については、GitHub の「Amazon ECS コンテナエージェント」を参照してください。

- オプションで、credentials-fetcher デーモン設定を変更してソケットを別の場所に移動する場合は、この変数 CREDENTIALS_FETCHER_HOST を使用できます。

許可とシークレットの設定

各アプリケーションおよび各タスク定義のために、次のステップを 1 回実行します。最小特権を認めるというベストプラクティスに従い、ポリシーで使用されるアクセス許可を絞り込むことをお勧めします。これにより、各タスクは必要なシークレットのみを読み取れます。

1. (オプション) ドメインレス gMSA シークレット

   インスタンスがドメインに参加していないドメインレス方式を使用する場合は、次のステップに従います。

   次の権限をインラインポリシーとしてタスク実行 IAM ロールに追加する必要があります。これにより、`credentials-fetcher` デーモンが Secrets Manager シークレットにアクセスできるようになります。`MySecret` の例を、`Resource` リスト内のシークレットの Amazon リソースネーム (ARN) に置き換えます。

   JSON

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": [
                   "secretsmanager:GetSecretValue"
               ],
               "Resource": "arn:aws:secretsmanager:us-
   east-1:123456789012:secret:my-secret-AbCdEf"
           }
       ]
   }
   ```

   > **Note**
   >
   > 独自の KMS キーを使用してシークレットを暗号化する場合は、必要な許可をこのロールに追加し、このロールを AWS KMS キーポリシーに追加する必要があります。

2. SSM Parameter Store または S3 を使用して CredSpec を保存するかどうかを決定する

   Amazon ECS では、タスク定義の `credentialSpecs` フィールドでファイルパスを参照する次の方法がサポートされています。

   インスタンスを単一のドメインに参加させる場合は、文字列内の ARN の先頭にプレフィックス `credentialspec:` を使用します。ドメインレス gMSA を使用する場合は、次に `credentialspecdomainless:` を使用します。

CredSpec の詳細については、「認証情報仕様ファイル」を参照してください。

- Amazon S3 バケット

  認証情報の仕様を Amazon S3 バケットに追加します。次に、タスク定義の `credentialSpecs` フィールドで Amazon S3 バケットの Amazon リソースネーム (ARN) を参照します。

```
{
    "family": "",
    "executionRoleArn": "",
    "containerDefinitions": [
        {
            "name": "",
            ...
            "credentialSpecs": [
                "credentialspecdomainless:arn:aws:s3:::${BucketName}/${ObjectName}"
            ],
            ...
        }
    ],
    ...
}
```

タスクに S3 バケットへのアクセスを許可するには、次のアクセス許可をインラインポリシーとして Amazon ECS タスク実行 IAM ロールに追加します。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor",
            "Effect": "Allow",
            "Action": [
                "s3:Get*",
                "s3:List*"
            ],
            "Resource": [
```

```
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/{object}"
            ]
        }
    ]
}
```

- SSM パラメータストアパラメータ

  SSM Parameter Store パラメータに認証情報仕様を追加します。タスク定義の `credentialSpecs` フィールドで SSM パラメータストアパラメータの Amazon リソースネーム (ARN) を参照します。

```
{
    "family": "",
    "executionRoleArn": "",
    "containerDefinitions": [
        {
            "name": "",
            ...
            "credentialSpecs": [
                "credentialspecdomainless:arn:aws:ssm:aws-
region:111122223333:parameter/parameter_name"
            ],
            ...
        }
    ],
    ...
}
```

  タスクに SSM Parameter Store パラメータへのアクセスを許可するには、次のアクセス許可をインラインポリシーとして Amazon ECS タスク実行 IAM ロールに追加します。

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
```

```
                "ssm:GetParameters"
            ],
            "Resource": "arn:aws:ssm:us-east-1:123456789012:parameter/my-
parameter"
        }
    ]
}
```

認証情報仕様ファイル

Amazon ECS は Active Directory の認証情報仕様ファイル (CredSpec) を使用します。このファイルには、gMSA アカウントコンテキストを Linux コンテナに伝達するために使用される gMSA メタデータが含まれています。CredSpec を生成し、タスク定義の `credentialSpecs` フィールドで参照します。CredSpec ファイルにはシークレットは含まれていません。

次は、CredSpec ファイルの例です。

```
{
    "CmsPlugins": [
        "ActiveDirectory"
    ],
    "DomainJoinConfig": {
        "Sid": "S-1-5-21-2554468230-2647958158-2204241789",
        "MachineAccountName": "WebApp01",
        "Guid": "8665abd4-e947-4dd0-9a51-f8254943c90b",
        "DnsTreeName": "example.com",
        "DnsName": "example.com",
        "NetBiosName": "example"
    },
    "ActiveDirectoryConfig": {
        "GroupManagedServiceAccounts": [
            {
                "Name": "WebApp01",
                "Scope": "example.com"
            }
        ],
        "HostAccountConfig": {
            "PortableCcgVersion": "1",
            "PluginGUID": "{859E1386-BDB4-49E8-85C7-3070B13920E1}",
            "PluginInput": {
                "CredentialArn": "arn:aws:secretsmanager:aws-
region:111122223333:secret:MySecret"
```

```
            }
        }
    }
}
```

「CredSpec」の作成

CredSpec を作成するには、ドメインに参加している Windows コンピューター上の CredSpec PowerShell モジュールを使用します。Microsoft Learn Web サイトの「認証情報の仕様を作成する」のステップに従います。

Fargate で Linux コンテナに gMSA を使用する

Amazon ECS は、グループ管理サービスアカウント (gMSA) と呼ばれる特殊なサービスアカウントを使用して、Fargate 上の Linux コンテナの Active Directory 認証をサポートします。

Linux Core アプリケーションなどの .NET ベースのネットワークアプリケーションは、Active Directory を使用して、ユーザーとサービス間の認証と認可の管理を円滑化することができます。この機能は、Active Directory と統合され、ドメインに参加しているサーバー上で実行されるアプリケーションを設計することで利用できます。しかし、Linux コンテナはドメインに参加できないため、gMSA を使用して実行する Linux コンテナを設定する必要があります。

考慮事項

Fargate で Linux コンテナ向けに gMSA を使用する前に、次の点を考慮してください。

- プラットフォームバージョン 1.4 以降を実行している必要があります。
- 前提条件を完全に満たすには、ドメインに参加している Windows コンピュータが必要になる場合があります。例えば、PowerShell を使用して Active Directory で gMSA を作成するには、ドメインに参加している Windows コンピュータが必要になる場合があります。RSAT Active Director PowerShell ツールは Windows でのみ使用できます。詳細については、「Active Directory 管理ツールのインストール」を参照してください。
- ドメインレス gMSA を使用する必要があります。

  Amazon ECS は Active Directory の認証情報仕様ファイル (CredSpec) を使用します。このファイルは、gMSA アカウントコンテキストをコンテナに伝達するために使用される gMSA メタデータを含む認証情報仕様ファイルを使用します。CredSpec ファイルを生成し、Amazon S3 バケットに保存します。
- 1 つのタスクでサポートできるアクティブディレクトリは 1 つだけです。

前提条件

gMSA を使用する前に、Amazon ECS の Linux コンテナ機能のために必ず以下を完了してください。

- コンテナがアクセスするリソースを含む Active Directory ドメインを設定します。Amazon ECS は以下の設定をサポートしています。
  - Directory Service Active Directory。Directory Service は、Amazon EC2 でホストされる AWS マネージド Active Directory です。詳細については、「AWS Directory Service 管理ガイド」の「AWS Managed Microsoft AD の開始方法」を参照してください。
  - オンプレミスの Active Directory。Amazon ECS Linux コンテナインスタンスがドメインに参加できることを確認する必要があります。詳細については、「AWS Direct Connect」を参照してください。
- Active Directory に既存の gMSA アカウントがあり、その gMSA サービスアカウントにアクセスする権限を持つユーザーがいます。詳細については、「ドメインレス gMSA 用に Active Directory ユーザーを作成する」を参照してください。
- Amazon S3 バケットがある。詳細については、「Amazon S3 ユーザーガイド」の「バケットの作成」を参照してください。

Amazon ECS での gMSA 対応の Linux コンテナの設定

インフラストラクチャを準備する

次のステップは、考慮事項と 1 回実行される設定です。

- ドメインレス gMSA 用に Active Directory ユーザーを作成する

  ドメインレス gMSAを使用する場合、コンテナはドメインに接続されません。コンテナ上で実行される他のアプリケーションは、Active Directory の認証情報を使用してドメインにアクセスすることはできません。別のドメインを使用するタスクも同じコンテナ上で実行できます。CredSpec ファイルの AWS Secrets Manager にシークレットの名前を指定します。シークレットには、ユーザー名、パスワード、ログイン先のドメインが含まれている必要があります。

  この機能は gMSA support for non-domain-joined container hosts 機能に似ています。Windows の機能の詳細については、Microsoft Learn ウェブサイトの「gMSA アーキテクチャと改善」を参照してください。

a. Active Directory ドメインでユーザーを設定します。Active Directory のユーザーは、タスクで使用する gMSA サービスアカウントにアクセスするための許可を持っている必要があります。

b. Active Directory ドメイン名を解決できる VPC とサブネットがあります。DHCP オプションを使用して、Active Directory サービス名を指すドメイン名で VPC を設定します。VPC 向け DHCP オプション設定の詳細については、「Amazon Virtual Private Cloud ユーザーガイド」の「DHCP オプションセットの使用」を参照してください。

c. AWS Secrets Manager にシークレットを作成

d. 認証情報仕様ファイルを作成します。

許可とシークレットの設定

各アプリケーションおよび各タスク定義ごとに、次のステップを 1 回ずつ実行します。最小特権を認めるというベストプラクティスに従い、ポリシーで使用されるアクセス許可を絞り込むことをお勧めします。これにより、各タスクは必要なシークレットのみを読み取れます。

1. Active Directory ドメインでユーザーを作成します。Active Directory のユーザーは、タスクで使用する gMSA サービスアカウントにアクセスするための許可を持っている必要があります。
2. Active Directory ユーザーを作成した後、AWS Secrets Manager でシークレットを作成します。詳細については、「AWS Secrets Manager シークレットを作成する」を参照してください。
3. ユーザーのユーザー名、パスワード、ドメインを、それぞれ `username`、`password`、`domainName` と呼ばれる JSON キーと値のペアに入力します。

```
{"username":"username","password":"passw0rd", "domainName":"example.com"}
```

4. 次の権限をインラインポリシーとしてタスク実行 IAM ロールに追加する必要があります。これにより、`credentials-fetcher` デーモンが Secrets Manager シークレットにアクセスできるようになります。`MySecret` の例を、`Resource` リスト内のシークレットの Amazon リソースネーム (ARN) に置き換えます。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": [
            "arn:aws:secretsmanager:us-east-1:111122223333:secret:MySecret"
            ]
        }
    ]
}
```

> **Note**
>
> 独自の KMS キーを使用してシークレットを暗号化する場合は、必要な許可をこのロールに追加し、このロールを AWS KMS キーポリシーに追加する必要があります。

5. 認証情報の仕様を Amazon S3 バケットに追加します。次に、タスク定義の `credentialSpecs` フィールドで Amazon S3 バケットの Amazon リソースネーム (ARN) を参照します。

```
{
    "family": "",
    "executionRoleArn": "",
    "containerDefinitions": [
        {
            "name": "",
            ...
            "credentialSpecs": [
                "credentialspecdomainless:arn:aws:s3:::${BucketName}/${ObjectName}"
            ],
            ...
        }
    ],
    ...
}
```

タスクに S3 バケットへのアクセスを許可するには、次のアクセス許可をインラインポリシーとして Amazon ECS タスク実行 IAM ロールに追加します。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor",
            "Effect": "Allow",
            "Action": [
            "s3:GetObjectVersion",
            "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::{bucket_name}",
                "arn:aws:s3:::{bucket_name}/{object}"
            ]
        }
    ]
}
```

認証情報仕様ファイル

Amazon ECS は Active Directory の認証情報仕様ファイル (CredSpec) を使用します。このファイルには、gMSA アカウントコンテキストを Linux コンテナに伝達するために使用される gMSA メタデータが含まれています。CredSpec を生成し、タスク定義の `credentialSpecs` フィールドで参照します。CredSpec ファイルにはシークレットは含まれていません。

次は、CredSpec ファイルの例です。

```
{
    "CmsPlugins": [
        "ActiveDirectory"
    ],
    "DomainJoinConfig": {
        "Sid": "S-1-5-21-2554468230-2647958158-2204241789",
        "MachineAccountName": "WebApp01",
        "Guid": "8665abd4-e947-4dd0-9a51-f8254943c90b",
        "DnsTreeName": "example.com",
        "DnsName": "example.com",
        "NetBiosName": "example"
```

```
    },
    "ActiveDirectoryConfig": {
        "GroupManagedServiceAccounts": [
            {
                "Name": "WebApp01",
                "Scope": "example.com"
            }
        ],
        "HostAccountConfig": {
            "PortableCcgVersion": "1",
            "PluginGUID": "{859E1386-BDB4-49E8-85C7-3070B13920E1}",
            "PluginInput": {
                "CredentialArn": "arn:aws:secretsmanager:aws-
region:111122223333:secret:MySecret"
            }
        }
    }
}
```

CredSpec を作成して Amazon S3 にアップロードする

CredSpec を作成するには、ドメインに参加している Windows コンピューター上の CredSpec PowerShell モジュールを使用します。Microsoft Learn Web サイトの「認証情報の仕様を作成する」のステップに従います。

認証情報仕様ファイルを作成したら、Amazon S3 バケットにアップロードします。AWS CLI コマンドを実行しているコンピューターまたは環境に CredSpec ファイルをコピーします。

次の AWS CLI コマンドを実行し、Amazon S3 に CredSpec をアップロードします。*amzn-s3-demo-bucket* をお使いの Amazon S3 バケットの名前に置き換えます。ファイルは任意のバケットと場所にオブジェクトとして保存できますが、タスク実行ロールにアタッチするポリシーでそのバケットと場所へのアクセスを許可する必要があります。

PowerShell については、次のコマンドを実行します。

```
$ Write-S3Object -BucketName "amzn-s3-demo-bucket" -Key "ecs-domainless-gmsa-credspec"
 -File "gmsa-cred-spec.json"
```

次の AWS CLI コマンドでは、sh および互換性のあるシェルで使用されるバックスラッシュ継続文字を使用します。

```
$ aws s3 cp gmsa-cred-spec.json \
```

```
s3://amzn-s3-demo-bucket/ecs-domainless-gmsa-credspec
```

AWS CLI を使用するドメインレス gMSA で Amazon ECS Windows コンテナを使用する

次のチュートリアルでは、AWS CLI を使用して Active Directory にアクセスするための認証情報を持つ Windows コンテナを実行する Amazon ECS タスクを作成する方法を示します。ドメインレス gMSA を使用すると、コンテナインスタンスはドメインに参加せず、インスタンス上の他のアプリケーションは認証情報を使用してドメインにアクセスできなくなり、異なるドメインに参加するタスクを同じインスタンス上で実行できます。

トピック

- 前提条件
- ステップ 1: Active Directory ドメイン サービス (AD DS) で gMSA アカウントを作成して設定する
- ステップ 2: Secrets Manager に認証情報をアップロードする
- ステップ 3: CredSpec JSON を変更してドメインレス gMSA 情報を含める
- ステップ 4： Amazon S3 に CredSpec をアップロードする
- ステップ 5: (オプション) Amazon ECS クラスターを作成する
- ステップ 6: コンテナインスタンスの IAM ロールを作成する
- ステップ 7: カスタムのタスク実行ロールを作成する
- ステップ 8: Amazon ECS Exec のタスクロールを作成する
- ステップ 9: ドメインレス gMSA を使用するタスク定義を登録する
- ステップ 10: Windows コンテナインスタンスをクラスターに登録する
- ステップ 11: コンテナインスタンスを検証する
- ステップ 12: Windows タスクを実行する
- ステップ 13: コンテナに gMSA 認証情報があることを検証する
- ステップ 14: クリーンアップする
- Windows コンテナ用 Amazon ECS ドメインレス gMSA のデバッグ

前提条件

このチュートリアルでは、以下の前提条件が完了済みであることを前提としています。

- 「Amazon ECS を使用するようにセットアップする」のステップを完了していること。
- IAM ユーザーに AmazonECS_FullAccess IAM ポリシー例で指定されている必要なアクセス権限があること。
- AWS CLI の最新バージョンがインストールされ、設定されていること。AWS CLI のインストールまたはアップグレードの詳細については、「AWS Command Line Interface のインストール」を参照してください。

> **Note**
>
> デュアルスタックサービスエンドポイントを使用することで、IPv4 と IPv6 の両方を介して AWS CLI、SDK、および Amazon ECS API から Amazon ECS とやり取りできます。詳細については、「Amazon ECS デュアルスタックエンドポイントの使用」を参照してください。

- コンテナがアクセスするリソースを含む Active Directory ドメインを設定します。Amazon ECS は以下の設定をサポートしています。
  - Directory Service Active Directory。Directory Service は、Amazon EC2 でホストされる AWS マネージド Active Directory です。詳細については、「AWS Directory Service 管理ガイド」の「AWS Managed Microsoft AD の開始方法」を参照してください。
  - オンプレミスの Active Directory。Amazon ECS Linux コンテナインスタンスがドメインに参加できることを確認する必要があります。詳細については、「AWS Direct Connect」を参照してください。
- Active Directory ドメイン名を解決できる VPC とサブネットがあります。
- ドメインレス gMSAか各インスタンスを単一のドメインに結合するを選択しました。ドメインレス gMSA を使用すると、コンテナインスタンスはドメインに参加せず、インスタンス上の他のアプリケーションは認証情報を使用してドメインにアクセスできなくなり、異なるドメインに参加するタスクを同じインスタンス上で実行できます。

  次に、CredSpec のデータ ストレージを選択し、必要に応じて、ドメインレス gMSA の Active Directory ユーザー認証情報を選択します。

  Amazon ECS は Active Directory の認証情報仕様ファイル (CredSpec) を使用します。このファイルは、gMSA アカウントコンテキストをコンテナに伝達するために使用される gMSA メタデータを含む認証情報仕様ファイルを使用します。CredSpec ファイルを生成し、コンテナインスタンスのオペレーティングシステムに応じて、次の表にある CredSpec ストレージオプションのいずれかに保存します。ドメインレス方式を使用するには、CredSpec ファイル内のオプションのセクショ

ンで、コンテナインスタンスのオペレーティングシステムに固有の、次の表の domainless user credentials ストレージオプションのいずれかの認証情報を指定できます。

ストレージの場所	Linux	Windows
Amazon Simple Storage Service	CredSpec	CredSpec
AWS Secrets Manager	ドメインレスユーザー認証情報	ドメインレスユーザー認証情報
Amazon EC2 Systems Manager Parameter Store	CredSpec	CredSpec、ドメインレスユーザー認証情報
ローカルファイル	該当なし	CredSpec

- (任意) AWS CloudShell は、お客様が独自の EC2 インスタンスを作成する必要なく、コマンドラインを提供するツールです。詳細については、AWS CloudShell ユーザーガイド の「What is AWS CloudShell? (とは？)」 を参照してください。

ステップ 1: Active Directory ドメイン サービス (AD DS) で gMSA アカウントを作成して設定する

Active Directory ドメインの gMSA アカウントを作成して設定します。

> **Note**
>
> このステップでは、コンテナの ID を提供するグループ管理サービスアカウント (gMSA) と、ドメイン認証に使用される通常のユーザーアカウントの 2 つの個別のアカウントを作成します。これらのアカウントは目的が異なるため、異なる名前を付ける必要があります。

1. Key Distribution Service のルートキーを生成する

   > **Note**
   >
   > Directory Service を使用している場合は、このステップを省略できます。

KDS ルートキーと gMSA アクセス許可は、AWS マネージド Microsoft AD で構成されます。

ドメインに 1 つの gMSA Service Account をまだ作成していない場合は、まず Key Distribution Service (KDS) ルートキーを生成する必要があります。KDS は、gMSA パスワードの作成、ローテーション、承認されたホストへのパスワードのリリースを担います。`ccg.exe` が gMSA 認証情報を取得する必要がある場合は、KDS に連絡して現在のパスワードを取得します。

KDS ルート キーが既に作成されているかどうかを確認するには、`ActiveDirectory` PowerShell モジュールを使用して、ドメインコントローラー上でドメイン管理者権限で次の PowerShell コマンドレットを実行します。モジュールの詳細については、Microsoft Learn Web サイトの「ActiveDirectory モジュール」を参照してください。

```
PS C:\> Get-KdsRootKey
```

コマンドがキー ID を返す場合は、このステップの残りをスキップできます。それ以外の場合は、次のコマンドを実行して KDS ルートキーを作成します。

```
PS C:\> Add-KdsRootKey -EffectiveImmediately
```

コマンドへの引数 `EffectiveImmediately` は、キーがすぐに有効になることを暗示していますが、KDS ルートキーが複製され、すべてのドメインコントローラーで使用できるようになるまで 10 時間待つ必要があります。

2. gMSA アカウントを作成するには

   gMSA アカウントを作成して `ccg.exe` が gMSA パスワードを取得できるようにするには、ドメインにアクセスできる Windows サーバーまたはクライアントから次の PowerShell コマンドを実行します。`ExampleAccount` を gMSA アカウントに必要な名前に置き換え、`example-domain` を Active Directory のドメイン名に置き換えます (例えば、ドメインが `contoso.com` の場合は、`contoso` を使用します)。

   a.
   ```
   PS C:\> Install-WindowsFeature RSAT-AD-PowerShell
   ```

   b.
   ```
   PS C:\> New-ADGroup -Name "ExampleAccount Authorized Hosts" -SamAccountName
    "ExampleAccountHosts" -GroupScope DomainLocal
   ```

c.
```
PS C:\> New-ADServiceAccount -Name "ExampleAccount" -DnsHostName "example-
domain" -ServicePrincipalNames "host/ExampleAccount", "host/example-domain" -
PrincipalsAllowedToRetrieveManagedPassword "ExampleAccountHosts"
```

d. 有効期限のない永続的なパスワードでユーザーを作成します。これらの認証情報は AWS Secrets Manager に保存され、各タスクがドメインに参加するために使用されます。これは、上記で作成した gMSA アカウントとは別のユーザーアカウントです。ExampleServiceUser を、このサービスユーザーアカウントに必要な名前へと置き換えます。

```
PS C:\> New-ADUser -Name "ExampleServiceUser" -AccountPassword (ConvertTo-
SecureString -AsPlainText "Test123" -Force) -Enabled 1 -PasswordNeverExpires 1
```

e.
```
PS C:\> Add-ADGroupMember -Identity "ExampleAccountHosts" -Members
 "ExampleServiceUser"
```

f. Active Directory に CredSpec オブジェクトを作成するための PowerShell モジュールをインストールし、CredSpec JSON を出力します。

```
PS C:\> Install-PackageProvider -Name NuGet -Force
```

```
PS C:\> Install-Module CredentialSpec
```

g.
```
PS C:\> New-CredentialSpec -AccountName ExampleAccount
```

3. 前のコマンドの JSON 出力を、gmsa-cred-spec.json と呼ばれるファイルにコピーします。これは、CredSpec ファイルです。ステップ 3 の ステップ 3: CredSpec JSON を変更してドメインレス gMSA 情報を含める で使用されます。

ステップ 2: Secrets Manager に認証情報をアップロードする

Active Directory 認証情報を安全な認証情報ストレージシステムにコピーして、各タスクがそれを取得できるようにします。これはドメインレス gMSA メソッドです。ドメインレス gMSA を使用すると、コンテナインスタンスはドメインに参加せず、インスタンス上の他のアプリケーションは認証情報を使用してドメインにアクセスできなくなり、異なるドメインに参加するタスクを同じインスタンス上で実行できます。

このステップでは、AWS CLI を使用します。AWS CloudShell のこれらのコマンドは、デフォルトのシェルである bash で実行できます。

- AWS CLI コマンドを実行し、ユーザー名、パスワード、ドメイン名を環境に合わせて置き換えます。ユーザー名には、サービスユーザーアカウント名 (gMSA アカウント名ではない) を使用します。シークレットの ARN を保管して次のステップ ステップ 3: CredSpec JSON を変更してドメインレス gMSA 情報を含める で使用します。

  次のコマンドでは、sh と互換性のあるシェルで使用されるバックスラッシュ継続文字を使用します。このコマンドは PowerShell と互換性がありません。PowerShell で使用するには、コマンドを変更する必要があります。

```
$ aws secretsmanager create-secret \
--name gmsa-plugin-input \
--description "Amazon ECS - gMSA Portable Identity." \
--secret-string "{\"username\":\"ExampleServiceUser\",\"password\":\"Test123\",
\"domainName\":\"contoso.com\"}"
```

ステップ 3: CredSpec JSON を変更してドメインレス gMSA 情報を含める

CredSpec をいずれかのストレージオプションにアップロードする前に、前の手順で Secrets Manager のシークレットの ARN を含む情報を CredSpec に追加します。詳細については、Microsoft Learn Web サイトの「ドメイン結合されていないコンテナホストのユースケース向けの追加の認証情報仕様設定」を参照してください。

1. `ActiveDirectoryConfig` 内部の CredSpec ファイルに以下の情報を追加します。ARN を、前の手順の Secrets Manager のシークレットに置き換えます。

   `PluginGUID` 値は次のサンプルのスニペットの GUID と一致する必要があり、必須であることに注意してください。

```
"HostAccountConfig": {
      "PortableCcgVersion": "1",
      "PluginGUID": "{859E1386-BDB4-49E8-85C7-3070B13920E1}",
      "PluginInput": "{\"credentialArn\": \"arn:aws:secretsmanager:aws-
region:111122223333:secret:gmsa-plugin-input\"}"
    }
```

また、この形式 \"arn:aws:ssm:*aws-region*:111122223333:parameter/*gmsa-plugin-input*\" の ARN を使用して SSM Parameter Store でシークレットを使用できます。

2. CredSpec ファイルを修正すると、次の例のようになります。

```
{
  "CmsPlugins": [
    "ActiveDirectory"
  ],
  "DomainJoinConfig": {
    "Sid": "S-1-5-21-4066351383-705263209-1606769140",
    "MachineAccountName": "ExampleAccount",
    "Guid": "ac822f13-583e-49f7-aa7b-284f9a8c97b6",
    "DnsTreeName": "example-domain",
    "DnsName": "example-domain",
    "NetBiosName": "example-domain"
  },
  "ActiveDirectoryConfig": {
    "GroupManagedServiceAccounts": [
      {
        "Name": "ExampleAccount",
        "Scope": "example-domain"
      },
      {
        "Name": "ExampleAccount",
        "Scope": "example-domain"
      }
    ],
    "HostAccountConfig": {
      "PortableCcgVersion": "1",
      "PluginGUID": "{859E1386-BDB4-49E8-85C7-3070B13920E1}",
      "PluginInput": "{\"credentialArn\": \"arn:aws:secretsmanager:aws-region:111122223333:secret:gmsa-plugin-input\"}"
    }
  }
}
```

ステップ 4： Amazon S3 に CredSpec をアップロードする

このステップでは、AWS CLI を使用します。AWS CloudShell のこれらのコマンドは、デフォルトのシェルである bash で実行できます。

1. AWS CLI コマンドを実行しているコンピューターまたは環境に CredSpec ファイルをコピーします。
2. 次の AWS CLI コマンドを実行し、Amazon S3 に CredSpec をアップロードします。`MyBucket` をお使いの Amazon S3 バケットの名前に置き換えます。ファイルは任意のバケットと場所にオブジェクトとして保存できますが、タスク実行ロールにアタッチするポリシーでそのバケットと場所へのアクセスを許可する必要があります。

   次のコマンドでは、sh と互換性のあるシェルで使用されるバックスラッシュ継続文字を使用します。このコマンドは PowerShell と互換性がありません。PowerShell で使用するには、コマンドを変更する必要があります。

   ```
   $ aws s3 cp gmsa-cred-spec.json \
   s3://MyBucket/ecs-domainless-gmsa-credspec
   ```

ステップ 5: (オプション) Amazon ECS クラスターを作成する

デフォルトでは、アカウントには `default` という名前の Amazon ECS クラスターがあります。このクラスターは、AWS CLI、SDK、CloudFormation でデフォルトで使用されます。追加のクラスターを使用してタスクとインフラストラクチャをグループ化および整理し、一部の設定にデフォルトを割り当てることができます。

クラスターは、AWS マネジメントコンソール、AWS CLI、SDK、または CloudFormation のいずれかを使用して作成できます。クラスター内の設定と設定は gMSA に影響しません。

このステップでは、AWS CLI を使用します。AWS CloudShell のこれらのコマンドは、デフォルトのシェルである `bash` で実行できます。

```
$ aws ecs create-cluster --cluster-name windows-domainless-gmsa-cluster
```

> **Important**
>
> 独自のクラスターを作成することを選択する場合は、そのクラスターで使用する予定のコマンドごとに `--cluster clusterName` を指定する必要があります。

ステップ 6: コンテナインスタンスの IAM ロールを作成する

コンテナインスタンスは、Amazon EC2 インスタンスなどの ECS タスクでコンテナを実行するホストコンピュータです。各コンテナインスタンスは Amazon ECS クラスターに登録されます。Amazon EC2 インスタンスを起動してクラスターに登録する前に、使用するコンテナインスタンス用の IAM ロールを作成する必要があります。

コンテナインスタンスロールを作成するには、「Amazon ECS コンテナインスタンスの IAM ロール」を参照してください。デフォルトの`ecsInstanceRole` は、このチュートリアルを完了するための十分なアクセス許可を持っています。

ステップ 7: カスタムのタスク実行ロールを作成する

Amazon ECS では、各タスクの開始に必要なアクセス権限として、コンテナインスタンスロールの代わりに別の IAM ロールを使用できます。このロールはタスク実行ロールです。ECS がタスクを実行するために必要なアクセス許可 (最小特権アクセス許可とも呼ばれる) のみを持つタスク実行ロールを作成することをお勧めします。最小特権の原則の詳細については、「AWS Well-Architected Framework」の「SEC03-BP02 最小特権アクセスの付与」を参照してください。

1. タスク実行ロールを作成するには、「タスク実行 ロールの作成」を参照してください。デフォルトの権限では、コンテナインスタンスは Amazon Elastic Container Registry、`stdout` および `stderr` からコンテナイメージを取得でき、アプリケーションから Amazon CloudWatch Logs に記録されます。

   このチュートリアルではロールにカスタム権限が必要なため、ロールには `ecsTaskExecutionRole` とは異なる名前を付けられます。このチュートリアルでは、以降の手順で `ecsTaskExecutionRole` を使用します。
2. このロールにのみ存在するインラインポリシー、または再利用可能なポリシーのいずれかのカスタムポリシーを作成して、次の権限を追加します。最初のステートメントの `Resource` の ARN を Amazon S3 バケットと場所に置き換え、2 番目の `Resource` を Secrets Manager のシークレットの ARN に置き換えます。

   Secrets Manager でカスタムキーを使用してシークレットを暗号化する場合は、そのキーの `kms:Decrypt` も許可する必要があります。

   Secrets Manager の代わりに SSM Parameter Store を使用する場合は、`secretsmanager:GetSecretValue` の代わりにパラメータの `ssm:GetParameter` を許可する必要があります。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::MyBucket/ecs-domainless-gmsa-credspec/
gmsa-cred-spec.json"
        },
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": "arn:aws:secretsmanager:us-
east-1:111122223333:secret:gmsa-plugin-AbCdEf"
        }
    ]
}
```

ステップ 8: Amazon ECS Exec のタスクロールを作成する

このチュートリアルでは、Amazon ECS Exec を使用して、実行中のタスク内でコマンドを実行して機能を検証します。ECS Exec を使用するには、サービスまたはタスクが ECS Exec を有効にし、タスク ロール (タスク実行ロールではない) に ssmmessages アクセス許可が必要です。必要な IAM ポリシーについては、「ECS Exec のアクセス許可」を参照してください。

このステップでは、AWS CLI を使用します。AWS CloudShell のこれらのコマンドは、デフォルトのシェルである bash で実行できます。

AWS CLI を使用してタスクロールを作成するには、次の手順に従ってください。

1. ecs-tasks-trust-policy.json というファイルを次の内容で作成します。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "ecs-tasks.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

2. IAM ロールを作成します。名前 `ecs-exec-demo-task-role` は置き替えられますが、次の手順のために名前を保持しておいてください。

   次のコマンドでは、sh と互換性のあるシェルで使用されるバックスラッシュ継続文字を使用します。このコマンドは PowerShell と互換性がありません。PowerShell で使用するには、コマンドを変更する必要があります。

   ```
   $ aws iam create-role --role-name ecs-exec-demo-task-role \
   --assume-role-policy-document file://ecs-tasks-trust-policy.json
   ```

   `ecs-tasks-trust-policy.json` ファイルを削除できます。

3. `ecs-exec-demo-task-role-policy.json` というファイルを次の内容で作成します。

   JSON

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": [
                   "ssmmessages:CreateControlChannel",
                   "ssmmessages:CreateDataChannel",
                   "ssmmessages:OpenControlChannel",
   ```

```
                "ssmmessages:OpenDataChannel"
            ],
            "Resource": "*"
        }
    ]
}
```

4. IAM ポリシーを作成し、前述のロールにアタッチします。

   次のコマンドでは、sh と互換性のあるシェルで使用されるバックスラッシュ継続文字を使用します。このコマンドは PowerShell と互換性がありません。PowerShell で使用するには、コマンドを変更する必要があります。

   ```
   $ aws iam put-role-policy \
       --role-name ecs-exec-demo-task-role \
       --policy-name ecs-exec-demo-task-role-policy \
       --policy-document file://ecs-exec-demo-task-role-policy.json
   ```

   ecs-exec-demo-task-role-policy.json ファイルを削除できます。

ステップ 9: ドメインレス gMSA を使用するタスク定義を登録する

このステップでは、AWS CLI を使用します。AWS CloudShell のこれらのコマンドは、デフォルトのシェルである bash で実行できます。

1. windows-gmsa-domainless-task-def.json というファイルを次の内容で作成します。

   ```
   {
     "family": "windows-gmsa-domainless-task",
     "containerDefinitions": [
       {
         "name": "windows_sample_app",
         "image": "mcr.microsoft.com/windows/servercore/iis",
         "cpu": 1024,
         "memory": 1024,
         "essential": true,
         "credentialSpecs": [
                   "credentialspecdomainless:arn:aws:s3:::ecs-domainless-gmsa-
   credspec/gmsa-cred-spec.json"
         ],
         "entryPoint": [
   ```

```
        "powershell",
        "-Command"
      ],
      "command": [
        "New-Item -Path C:\\inetpub\\wwwroot\\index.html -ItemType file -Value
 '<html> <head> <title>Amazon ECS Sample App</title> <style>body {margin-top:
 40px; background-color: #333;} </style> </head><body> <div style=color:white;text-
align:center> <h1>Amazon ECS Sample App</h1> <h2>Congratulations!</h2> <p>Your
 application is now running on a container in Amazon ECS.</p>' -Force ; C:\
\ServiceMonitor.exe w3svc"
      ],
      "portMappings": [
        {
          "protocol": "tcp",
          "containerPort": 80,
          "hostPort": 8080
        }
      ]
    }
  ],
  "taskRoleArn": "arn:aws:iam::111122223333:role/ecs-exec-demo-task-role",
  "executionRoleArn": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole"
}
```

2. 次のコマンドを実行して、タスク定義を登録します。

   次のコマンドでは、sh と互換性のあるシェルで使用されるバックスラッシュ継続文字を使用します。このコマンドは PowerShell と互換性がありません。PowerShell で使用するには、コマンドを変更する必要があります。

```
$ aws ecs register-task-definition \
--cli-input-json file://windows-gmsa-domainless-task-def.json
```

ステップ 10: Windows コンテナインスタンスをクラスターに登録する

Amazon EC2 Windows インスタンスを起動し、ECS コンテナエージェントを実行して、それをコンテナインスタンスとしてクラスターに登録します。ECS は、タスクが開始されたクラスターに登録されているコンテナインスタンスでタスクを実行します。

1. AWS マネジメントコンソール で Amazon ECS 用に設定された Amazon EC2 Windows インスタンスを起動するには、「Amazon ECS Windows コンテナインスタンスの起動」を参照してください。ユーザーデータのステップで停止します。
2. gMSA には、ECS コンテナエージェントを開始する前に、ユーザーデータで環境変数 ECS_GMSA_SUPPORTED を設定する必要があります。

   ECS Exec の場合、エージェントは引数 -EnableTaskIAMRole で始まる必要があります。

   タスクが EC2 IMDS Web サービスに到達してロール認証情報を取得できないようにしてインスタンス IAM ロールを保護するには、引数 -AwsvpcBlockIMDS を追加します。これは、awsvpc ネットワークモードを使用するタスクにのみ適用されます。

   ```
   <powershell>
   [Environment]::SetEnvironmentVariable("ECS_GMSA_SUPPORTED", $TRUE, "Machine")
   Import-Module ECSTools
   Initialize-ECSAgent -Cluster windows-domainless-gmsa-cluster -EnableTaskIAMRole -
   AwsvpcBlockIMDS
   </powershell>
   ```
3. Summary (概要) パネルでインスタンス設定の要約を確認します。準備が完了したら、[Launch instance] (インスタンスを起動) を選択してください。

ステップ 11: コンテナインスタンスを検証する

AWS マネジメントコンソール を使用して、クラスター内にコンテナインスタンスがあることを確認できます。ただし、gMSA は属性として示される追加の機能が必要です。これらの属性は AWS マネジメントコンソール では表示されないため、このチュートリアルでは AWS CLI を使用します。

このステップでは、AWS CLI を使用します。AWS CloudShell のこれらのコマンドは、デフォルトのシェルである bash で実行できます。

1. クラスター内のコンテナインスタンスをリストします。コンテナインスタンスの ID は EC2 インスタンスの ID とは異なります。

   ```
   $ aws ecs list-container-instances
   ```

   出力:

```
{
    "containerInstanceArns": [
        "arn:aws:ecs:aws-region:111122223333:container-
instance/default/MyContainerInstanceID"
    ]
}
```

例えば、526bd5d0ced448a788768334e79010fd は有効なコンテナインスタンス ID です。

2. 前のステップのコンテナインスタンス ID を使用して、コンテナインスタンスの詳細を取得します。MyContainerInstanceID を ID に置き換えます。

   次のコマンドでは、sh と互換性のあるシェルで使用されるバックスラッシュ継続文字を使用します。このコマンドは PowerShell と互換性がありません。PowerShell で使用するには、コマンドを変更する必要があります。

   ```
   $ aws ecs describe-container-instances \
        ----container-instances MyContainerInstanceID
   ```

   出力が非常に長いことに注意してください。

3. attributes リストに、name というキーと値 ecs.capability.gmsa-domainless を持つオブジェクトがあることを確認します。以下は、オブジェクトの例です。

   出力:

   ```
   {
       "name": "ecs.capability.gmsa-domainless"
   }
   ```

ステップ 12: Windows タスクを実行する

Amazon ECS タスクを実行します。クラスターにコンテナインスタンスが 1 つしかない場合は、run-task を使用できます。さまざまなコンテナインスタンスが多数ある場合は、タスク定義に配置制約を追加して、このタスクを実行するコンテナインスタンスの種類を制御するよりも、タスクを実行するには start-task を使用してコンテナインスタンス ID を指定する方が簡単な場合があります。

このステップでは、AWS CLI を使用します。AWS CloudShell のこれらのコマンドは、デフォルトのシェルである bash で実行できます。

1. 次のコマンドでは、sh と互換性のあるシェルで使用されるバックスラッシュ継続文字を使用します。このコマンドは PowerShell と互換性がありません。PowerShell で使用するには、コマンドを変更する必要があります。

```
aws ecs run-task --task-definition windows-gmsa-domainless-task \
    --enable-execute-command --cluster windows-domainless-gmsa-cluster
```

   コマンドによって返されるタスク ID をメモします。

2. 次のコマンドを実行して、タスクが開始されたことを確認します。このコマンドは、タスクが開始されるまで待機し、シェルプロンプトを返しません。MyTaskID を前のステップのタスク ID に置き換えます。

```
$ aws ecs wait tasks-running --task MyTaskID
```

ステップ 13: コンテナに gMSA 認証情報があることを検証する

タスク内のコンテナに Kerberos トークン gMSA があることを確認します。

このステップでは、AWS CLI を使用します。AWS CloudShell のこれらのコマンドは、デフォルトのシェルである bash で実行できます。

1. 次のコマンドでは、sh と互換性のあるシェルで使用されるバックスラッシュ継続文字を使用します。このコマンドは PowerShell と互換性がありません。PowerShell で使用するには、コマンドを変更する必要があります。

```
$ aws ecs execute-command \
--task MyTaskID \
--container windows_sample_app \
--interactive \
--command powershell.exe
```

   出力は PowerShell プロンプトになります。

2. コンテナ内の PowerShell ターミナルで次のコマンドを実行します。

```
PS C:\> klist get ExampleAccount$
```

出力では、Principal は以前に作成したものであることに注意してください。

ステップ 14: クリーンアップする

このチュートリアルが終了したら、未使用のリソースに対する料金が発生しないように、それに関連付けられたリソースをクリーンアップする必要があります。

このステップでは、AWS CLI を使用します。AWS CloudShell のこれらのコマンドは、デフォルトのシェルである bash で実行できます。

1. タスクを停止します。MyTaskID をステップ 12 のタスク ID ステップ 12: Windows タスクを実行する に置き換えます。

   ```
   $ aws ecs stop-task --task MyTaskID
   ```

2. Amazon EC2 インスタンスを停止します。その後、クラスター内のコンテナインスタンスは 1 時間後に自動で削除されます。

   Amazon EC2 コンソールを使用し、インスタンスを検索して終了できます。または、以下のコマンドを実行できます。コマンドを実行するには、ステップ 1、ステップ 11: コンテナインスタンスを検証する の aws ecs describe-container-instances コマンドの出力で EC2 インスタンス ID を見つけます。i-10a64379 は EC2 インスタンス ID の例です。

   ```
   $ aws ec2 terminate-instances --instance-ids MyInstanceID
   ```

3. Amazon S3 の CredSpec ファイルを削除します。MyBucket をお使いの Amazon S3 バケットの名前に置き換えます。

   ```
   $ aws s3api delete-object --bucket MyBucket --key ecs-domainless-gmsa-credspec/
   gmsa-cred-spec.json
   ```

4. シークレットを Secrets Manager から削除するには 代わりに SSM Parameter Store を使用した場合は、パラメータを削除してください。

次のコマンドでは、sh と互換性のあるシェルで使用されるバックスラッシュ継続文字を使用します。このコマンドは PowerShell と互換性がありません。PowerShell で使用するには、コマンドを変更する必要があります。

```
$ aws secretsmanager delete-secret --secret-id gmsa-plugin-input \
     --force-delete-without-recovery
```

5. タスク定義を登録解除して削除します。タスク定義を登録解除すると、そのタスク定義を非アクティブとしてマークするため、新しいタスクを開始するのに使用できなくなります。その後、タスク定義を削除できます。

   a. バージョンを指定してタスク定義の登録を解除します。ECS は、1 から始まる番号の付いたタスク定義のバージョンを自動的に作成します。:1 などのコンテナイメージのラベルと同じ形式のバージョンを参照します。

   ```
   $ aws ecs deregister-task-definition --task-definition windows-gmsa-domainless-task:1
   ```

   b. タスク定義を削除します。

   ```
   $ aws ecs delete-task-definitions --task-definition windows-gmsa-domainless-task:1
   ```

6. (オプション) クラスターを作成した場合は、ECS クラスターを削除します。

```
$ aws ecs delete-cluster --cluster windows-domainless-gmsa-cluster
```

Windows コンテナ用 Amazon ECS ドメインレス gMSA のデバッグ

Amazon ECS タスクの状態

ECS はタスクを 1 回のみ開始しようとします。問題のあるタスクはすべて停止され、STOPPED 状態に設定されます。タスクに関する一般的な問題には 2 つのタイプがあります。最初に、開始できなかったタスクです。2 つ目は、アプリケーションがいずれかのコンテナ内で停止したタスクです。AWS マネジメントコンソール で、タスクが停止された理由については、タスクの [停止理由] フィールドを確認してください。AWS CLI に、タスクを説明し

て `stoppedReason` を見ます。AWS マネジメントコンソール および AWS CLI での手順については、「Amazon ECS の停止したタスクのエラーを表示する」を参照してください。

Windows イベント

コンテナ内の gMSA 用 Windows イベントは `Microsoft-Windows-Containers-CCG` ログファイルに記録され、`Logs\Microsoft\Windows\Containers-CCG\Admin` にある「アプリケーションとサービス」セクションのイベント ビューアに表示されます。デバッグのヒントの詳細については、Microsoft Learn Web サイトの「Windows コンテナの gMSA のトラブルシューティング」を参照してください。

ECS エージェント gMSA プラグイン

Windows コンテナインスタンス上の ECS エージェントの gMSA プラグインのログは、次のディレクトリ `C:/ProgramData/Amazon/gmsa-plugin/` にあります。このログを調べて、ドメインレスユーザーの認証情報がSecrets Manager などのストレージ場所からダウンロードされたかどうか、および認証情報の形式が正しく読み取られたかどうかを確認してください。

Amazon ECS の EC2 Windows コンテナで gMSA を使用する方法について説明します。

Amazon ECS は、グループマネージドサービスアカウント (gMSA) と呼ばれる特殊なサービスアカウントを使用して、Windows コンテナの Active Directory 認証をサポートします。

.NET アプリケーションなどの Windows ベースのネットワークアプリケーションは、多くの場合、Active Directory を使用して、ユーザーとサービス間の認証と認可の管理を容易にします。開発者は、通常、Active Directory と統合し、この目的のためにドメインに参加しているサーバーで実行するようにアプリケーションを設計します。Windows コンテナはドメインに結合できないため、gMSA で実行するように Windows コンテナを設定する必要があります。

gMSA で実行されている Windows コンテナは、ホスト Amazon EC2 インスタンスに依存して、Active Directory ドメインコントローラから gMSA 資格情報を取得し、コンテナインスタンスに提供します。詳細については、「Windows コンテナでの gMSA の使用」を参照してください。

> **Note**
>
> この機能は、Fargate の Windows コンテナではサポートされていません。

トピック

- 考慮事項
- 前提条件
- Amazon ECS での Windows コンテナ向け gMSA の設定

考慮事項

Windows コンテナに gMSAs を使用する場合は、次の点を考慮する必要があります。

- コンテナインスタンスに Amazon ECSに最適化されたWindows Server 2016 Full AMI を使用する場合、コンテナのホスト名は、認証情報仕様ファイルで定義されている gMSA アカウント名と同じである必要があります。コンテナのホスト名を指定するには、`hostname` コンテナ定義パラメータを使用します。詳細については、「ネットワーク設定」を参照してください。
- ドメインレス gMSAか各インスタンスを単一のドメインに結合するを選択しました。ドメインレス gMSA を使用すると、コンテナインスタンスはドメインに参加せず、インスタンス上の他のアプリケーションは認証情報を使用してドメインにアクセスできなくなり、異なるドメインに参加するタスクを同じインスタンス上で実行できます。

  次に、CredSpec のデータ ストレージを選択し、必要に応じて、ドメインレス gMSA の Active Directory ユーザー認証情報を選択します。

  Amazon ECS は Active Directory の認証情報仕様ファイル (CredSpec) を使用します。このファイルは、gMSA アカウントコンテキストをコンテナに伝達するために使用される gMSA メタデータを含む認証情報仕様ファイルを使用します。CredSpec ファイルを生成し、コンテナインスタンスのオペレーティングシステムに応じて、次の表にある CredSpec ストレージオプションのいずれかに保存します。ドメインレス方式を使用するには、CredSpec ファイル内のオプションのセクションで、コンテナインスタンスのオペレーティングシステムに固有の、次の表の domainless user credentials ストレージオプションのいずれかの認証情報を指定できます。

ストレージの場所	Linux	Windows
Amazon Simple Storage Service	CredSpec	CredSpec
AWS Secrets Manager	ドメインレスユーザー認証情報	ドメインレスユーザー認証情報

ストレージの場所	Linux	Windows
Amazon EC2 Systems Manager Parameter Store	CredSpec	CredSpec、ドメインレスユーザー認証情報
ローカルファイル	該当なし	CredSpec

前提条件

Amazon ECS で Windows コンテナ向け gMSA 機能を使用する前に、必ず次の作業を完了してください。

- コンテナがアクセスするリソースを含む Active Directory ドメインを設定します。Amazon ECS は以下の設定をサポートしています。
  - Directory Service Active Directory。Directory Service は、Amazon EC2 でホストされる AWS マネージド Active Directory です。詳細については、「AWS Directory Service 管理ガイド」の「AWS Managed Microsoft AD の開始方法」を参照してください。
  - オンプレミスの Active Directory。Amazon ECS Linux コンテナインスタンスがドメインに参加できることを確認する必要があります。詳細については、「AWS Direct Connect」を参照してください。
- Active Directory に既存の gMSA アカウントがあります。詳細については、「Windows コンテナでの gMSA の使用」を参照してください。
- ドメインレス gMSA を使用することを選択した場合、または Amazon ECS タスクをホストする Amazon ECS Windows コンテナインスタンスは、Active Directory にドメイン参加しており、gMSA アカウントにアクセスできる Active Directory セキュリティグループのメンバーである必要があります。

  ドメインレス gMSA を使用すると、コンテナインスタンスはドメインに参加せず、インスタンス上の他のアプリケーションは認証情報を使用してドメインにアクセスできなくなり、異なるドメインに参加するタスクを同じインスタンス上で実行できます。
- 必須の IAM アクセス権限を追加しました。必要な権限は、初期認証情報と認証情報の仕様の保存に選択した方法によって異なります。
  - 初期認証情報にドメインレス gMSA を使用する場合は、Amazon EC2 インスタンスロールに AWS Secrets Manager の IAM アクセス許可が必要です。

- 認証情報の仕様を SSM Parameter Store に保存する場合は、タスク実行ロールで Amazon EC2 Systems Manager Parameter Store に対する IAM 許可が必要です。
- 認証情報の仕様を Amazon S3 で保存する場合は、タスク実行ロールで Amazon Simple Storage Service の IAM 許可が必要です。

Amazon ECS での Windows コンテナ向け gMSA の設定

前提条件 AWS CLI を使用するドメインレス gMSA で Amazon ECS Windows コンテナを使用する の設定を含む完全なチュートリアルに従い、Windows コンテナ向け gMSA を Amazon ECS 上で設定できます。

次のセクションでは、CredSpec の設定について詳しく説明します。

トピック

- CredSpec の例
- ドメインレス gMSA 設定
- タスク定義での認証情報仕様ファイルの参照

CredSpec の例

Amazon ECS は、gMSA アカウントコンテキストを Windows コンテナに伝達するために使用される gMSA メタデータを含む認証情報仕様ファイルを使用します。認証情報仕様ファイルを生成し、タスク定義の `credentialSpec` フィールドで参照できます。認証情報仕様ファイルにはシークレットが含まれていません。

認証情報仕様ファイルの例を次に示します。

```
{
    "CmsPlugins": [
        "ActiveDirectory"
    ],
    "DomainJoinConfig": {
        "Sid": "S-1-5-21-2554468230-2647958158-2204241789",
        "MachineAccountName": "WebApp01",
        "Guid": "8665abd4-e947-4dd0-9a51-f8254943c90b",
        "DnsTreeName": "contoso.com",
        "DnsName": "contoso.com",
        "NetBiosName": "contoso"
    },
```

```
    "ActiveDirectoryConfig": {
        "GroupManagedServiceAccounts": [
            {
                "Name": "WebApp01",
                "Scope": "contoso.com"
            }
        ]
    }
}
```

ドメインレス gMSA 設定

コンテナインスタンスを単一のドメインに参加させるのではなく、ドメインレス gMSA をお勧めします。ドメインレス gMSA を使用すると、コンテナインスタンスはドメインに参加せず、インスタンス上の他のアプリケーションは認証情報を使用してドメインにアクセスできなくなり、異なるドメインに参加するタスクを同じインスタンス上で実行できます。

1. CredSpec をいずれかのストレージオプションにアップロードする前に、Secrets Manager または SSM Parameter Store のシークレットの ARN を含む情報を CredSpec に追加します。詳細については、Microsoft Learn Web サイトの「ドメイン結合されていないコンテナホストのユースケース向けの追加の認証情報仕様設定」を参照してください。

   ドメインレス gMSA 認証情報の形式

   以下は、Active Directory のドメインレス gMSA 認証情報の JSON 形式です。認証情報を Secrets Manager または SSM Parameter Store に保存します。

   ```
   {
       "username":"WebApp01",
       "password":"Test123!",
       "domainName":"contoso.com"
   }
   ```

2. `ActiveDirectoryConfig` 内部の CredSpec ファイルに以下の情報を追加します。ARN を Secrets Manager または SSM Parameter Store のシークレットに置き換えます。

   `PluginGUID` 値は次のサンプルのスニペットの GUID と一致する必要があり、必須であることに注意してください。

   ```
       "HostAccountConfig": {
   ```

```
        "PortableCcgVersion": "1",
        "PluginGUID": "{859E1386-BDB4-49E8-85C7-3070B13920E1}",
        "PluginInput": "{\"credentialArn\": \"arn:aws:secretsmanager:aws-
region:111122223333:secret:gmsa-plugin-input\"}"
    }
```

また、この形式 \"arn:aws:ssm:*aws-region*:111122223333:parameter/*gmsa-plugin-input*\" の ARN を使用して SSM Parameter Store でシークレットを使用できます。

3. CredSpec ファイルを修正すると、次の例のようになります。

```
{
  "CmsPlugins": [
    "ActiveDirectory"
  ],
  "DomainJoinConfig": {
    "Sid": "S-1-5-21-4066351383-705263209-1606769140",
    "MachineAccountName": "WebApp01",
    "Guid": "ac822f13-583e-49f7-aa7b-284f9a8c97b6",
    "DnsTreeName": "contoso",
    "DnsName": "contoso",
    "NetBiosName": "contoso"
  },
  "ActiveDirectoryConfig": {
    "GroupManagedServiceAccounts": [
      {
        "Name": "WebApp01",
        "Scope": "contoso"
      },
      {
        "Name": "WebApp01",
        "Scope": "contoso"
      }
    ],
    "HostAccountConfig": {
      "PortableCcgVersion": "1",
      "PluginGUID": "{859E1386-BDB4-49E8-85C7-3070B13920E1}",
      "PluginInput": "{\"credentialArn\": \"arn:aws:secretsmanager:aws-
region:111122223333:secret:gmsa-plugin-input\"}"
    }
  }
}
```

タスク定義での認証情報仕様ファイルの参照

Amazon ECS では、タスク定義の credentialSpecs フィールドでファイルパスを参照する次の方法がサポートされています。これらの各オプションでは、コンテナインスタンスを単一のドメインに参加させるかドメインレス gMSA を使用するかに応じて、credentialspec: または domainlesscredentialspec: を指定できます。

Amazon S3 バケット

認証情報仕様を Amazon S3 バケットに追加し、タスク定義の credentialSpecs フィールドで Amazon S3 バケットの Amazon リソースネーム (ARN) を参照します。

```
{
    "family": "",
    "executionRoleArn": "",
    "containerDefinitions": [
        {
            "name": "",
            ...
            "credentialSpecs": [
                "credentialspecdomainless:arn:aws:s3:::${BucketName}/${ObjectName}"
            ],
            ...
        }
    ],
    ...
}
```

また、次のアクセス権限をインラインポリシーとして Amazon ECS タスク実行 IAM ロール に追加して、タスクに Amazon S3 バケットへのアクセスを付与する必要があります。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor",
            "Effect": "Allow",
            "Action": [
                "s3:Get*",
```

```
                "s3:List*"
            ],
            "Resource": [
                "arn:aws:s3:::{bucket_name}",
                "arn:aws:s3:::{bucket_name}/{object}"
            ]
        }
    ]
}
```

SSM パラメータストアパラメータ

認証情報仕様を SSM パラメータストアパラメータに追加し、タスク定義の `credentialSpecs` フィールドで SSM パラメータストアパラメータの Amazon リソースネーム (ARN) を参照します。

```
{
    "family": "",
    "executionRoleArn": "",
    "containerDefinitions": [
        {
            "name": "",
            ...
            "credentialSpecs": [

 "credentialspecdomainless:arn:aws:ssm:region:111122223333:parameter/parameter_name"
            ],
            ...
        }
    ],
    ...
}
```

また、次のアクセス権限をインラインポリシーとして Amazon ECS タスク実行 IAM ロール に追加して、タスクに SSM パラメータストアパラメータへのアクセス権を付与する必要があります。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameters"
            ],
            "Resource": "arn:aws:ssm:*:111122223333:parameter/example*"
        }
    ]
}
```

ローカルファイル

ローカルファイルの認証情報仕様の詳細を使用して、タスク定義の credentialSpecs フィールドでファイルパスを参照します。参照されるファイルパスは C:\ProgramData\Docker\CredentialSpecs ディレクトリからの相対パスで、ファイルパスの区切り文字にはバックスラッシュ ('\') を使用する必要があります。

```
{
    "family": "",
    "executionRoleArn": "",
    "containerDefinitions": [
        {
            "name": "",
            ...
            "credentialSpecs": [
                "credentialspec:file://CredentialSpecDir\CredentialSpecFile.json"
            ],
            ...
        }
    ],
    ...
}
```

EC2 Image Builder を使用して、Amazon ECS に最適化されたカスタム AMI を構築する

AWS では、コンテナ ワークロードを実行するための要件および推奨事項であらかじめ設定されている、Amazon ECS に最適化された AMI の使用をお勧めしています。ソフトウェアの追加には AMI のカスタマイズが必要になる場合があります。EC2 Image Builder は、サーバーイメージの作成、管理、デプロイに使用できます。アカウントで作成されるカスタムイメージの所有権は、お客様に保持

されます。EC2 Image Builder パイプラインを使用してイメージの更新やシステムパッチを自動化したり、スタンドアロンコマンドを使用して定義済みの設定リソースでイメージを作成することができます。

イメージのレシピを作成します。レシピには親画像とその他の追加コンポーネントが含まれます。さらに、カスタマイズした AMI を配信するパイプラインを作成します。

イメージのレシピを作成します。Image Builder イメージレシピは、ベースイメージとベースイメージに適用するコンポーネントを定義し、必要な設定の出力イメージを生成するためのドキュメントです。さらに、カスタマイズした AMI を配信するパイプラインを作成します。詳細については、「EC2 Image Builder ユーザーガイド」の「EC2 Image Builder の仕組み」を参照してください。

EC2 Image Builder の「親イメージ」として、次の Amazon ECS に最適化された AMI のいずれかの使用をお勧めします。

- Linux
  - Amazon ECS 最適化 AL2023 x86
  - Amazon ECS 最適化 Amazon Linux 2023 (arm64) AMI
  - Amazon ECS 最適化 Amazon Linux 2 カーネル 5 AMI
  - Amazon ECS 最適化 Amazon Linux 2 x86 AMI
- Windows
  - Amazon ECS 最適化 Windows 2022 Full x86
  - Amazon ECS 最適化 Windows 2022 Core x86
  - Amazon ECS 最適化 Windows 2019 Full x86
  - Amazon ECS 最適化 Windows 2019 Core x86
  - Amazon ECS 最適化 Windows 2016 Full x86

利用可能な最新バージョンを使用することをお勧めします。パイプラインでは親イメージにセマンティックバージョニングを使用します。これにより、自動スケジューリングされたジョブの依存関係の更新を検出しやすくなります。詳細については、「EC2 Image Builder ユーザーガイド」の「セマンティックバージョニング」を参照してください。

AWS は、Amazon ECS に最適化された AMI イメージを、セキュリティパッチと新しいコンテナエージェントバージョンで定期的に更新しています。AMI ID をイメージレシピの親イメージとして使用する場合、親イメージの更新がないかを定期的に確認する必要があります。更新がある場合は、更新した AMI を使用して新しいバージョンのレシピを作成する必要があります。これにより、カス

タムイメージに最新バージョンの親イメージが組み込まれます。新しく作成した AMI を使用して Amazon ECS クラスターの EC2 インスタンスを自動的に更新するワークフローを作成する方法については、「AMI 強化パイプラインを作成して ECS インスタンスフリートの更新を自動化する方法」を参照してください。

マネージド EC2 Image Builder パイプラインを通じて公開された親イメージの Amazon リソースネーム (ARN) を指定することもできます。Amazon は、マネージドパイプラインを通じて、Amazon ECS 最適化 AMI イメージを定期的に公開します。これらのイメージはパブリックアクセス可能です。イメージにアクセスするには、適切なアクセス許可が必要です。Image Builder レシピで AMI の代わりにイメージ ARN を使用すると、パイプラインは実行のたびに最新バージョンの親イメージを自動的に使用します。この方法により、更新のたびに新しいレシピバージョンを手動で作成する必要がなくなります。

Amazon ECS に最適化された AMI のクリーンアップ

Amazon ECS に最適化された AMI を親イメージとして使用する場合は、一時的な問題を防ぐためにイメージをクリーンアップする必要があります。Amazon ECS に最適化された AMI は、Amazon ECS エージェントが自動起動してコンテナインスタンスとして Amazon ECS に登録するように事前設定されています。適切なクリーンアップを行わずにベースイメージとして使用すると、カスタム AMI で問題が発生する可能性があります。

今後の使用のためにイメージをクリーンアップするには、次のコマンドを実行して ecs-init パッケージと Docker プロセスを停止するコンポーネントを作成します。

```
sudo systemctl stop ecs
sudo systemctl stop docker
```

イメージの保存時に一緒に保存されないように、すべてのログファイルを現在のインスタンスから削除します。EC2 Image Builder のセキュリティのベストプラクティスのスクリプト例を使用して、インスタンスから各種ファイルをクリーンアップします。

Amazon ECS 固有のデータをクリーンアップするには、次のコマンドを実行します。

```
sudo rm -rf /var/log/ecs/*
sudo rm /var/lib/ecs/data/agent.db
```

カスタム Amazon ECS 最適化 AMI を作成する方法については、AWS ナレッジセンターの「How do I create a custom AMI from an Amazon ECS-optimized AMI?」を参照してください。

イメージ ARN と Infrastructure as Code (IaC) の使用

EC2 Image Builder コンソール、Infrastructure as Code (例: CloudFormation)、または AWS SDK を使用してレシピを設定できます。レシピで親イメージを指定する場合、EC2 AMI ID、Image Builder イメージ ARN、AWS Marketplace プロダクト ID、またはコンテナイメージを指定できます。AWS は、Amazon ECS 最適化 AMI の AMI ID と Image Builder イメージ ARN の両方を公開します。イメージの ARN 形式は以下の通りです。

```
arn:${Partition}:imagebuilder:${Region}:${Account}:image/${ImageName}/${ImageVersion}
```

`ImageVersion` の形式は以下のようになっています。*`[####]`*、*`[####]`*、*`[###]`*を最新の値に置き換えます。

```
<major>.<minor>.<patch>
```

`major`、`minor`、`patch` を特定の値に置き換えたり、イメージのバージョンレス ARN を使用することができます。これにより、パイプラインは常に親イメージの最新バージョンに更新された状態に保たれます。バージョンレス ARN では、ワイルドカード形式「x.x.x」でイメージバージョンを表現します。この方法により、Image Builder サービスは最新バージョンのイメージに自動的に解決されます。バージョンレス ARN を使用すると、参照先は常に利用可能な最新のイメージを示すため、デプロイ内のイメージを最新の状態に維持するプロセスが効率化されます。コンソールでレシピを作成すると、EC2 Image Builder は親イメージの ARN を自動的に識別します。IaC を使用してレシピを作成する場合、ARN を特定して目的のイメージバージョンを選択するか、バージョンレス ARN を使用して使用可能な最新のイメージを示す必要があります。フィルタリングにより条件に合致するイメージのみを表示する自動スクリプトを作成することをお勧めします。次の Python スクリプトは、Amazon ECS 最適化 AMI のリストを取得する方法を示しています。

このスクリプトは、`owner` と `platform` の 2 つのオプション引数を受け付けます。デフォルト値はそれぞれ「Amazon」と「Windows」です。owner 引数の有効値は、`Self`、`Shared`、`Amazon`、`ThirdParty` です。プラットフォーム引数の有効値は、`Windows` と `Linux` です。たとえば、`owner` 引数を `Amazon` に設定し、`platform` を `Linux` に設定してスクリプトを実行すると、スクリプトは Amazon ECS 最適化イメージを含む、Amazon によって公開されたイメージのリストを生成します。

```
import boto3
import argparse

def list_images(owner, platform):
    # Create a Boto3 session
```

```
    session = boto3.Session()

    # Create an EC2 Image Builder client
    client = session.client('imagebuilder')

    # Define the initial request parameters
    request_params = {
        'owner': owner,
        'filters': [
            {
                'name': 'platform',
                'values': [platform]
            }
        ]
    }

    # Initialize the results list
    all_images = []

    # Get the initial response with the first page of results
    response = client.list_images(**request_params)

    # Extract images from the response
    all_images.extend(response['imageVersionList'])

    # While 'nextToken' is present, continue paginating
    while 'nextToken' in response and response['nextToken']:
        # Update the token for the next request
        request_params['nextToken'] = response['nextToken']

        # Get the next page of results
        response = client.list_images(**request_params)

        # Extract images from the response
        all_images.extend(response['imageVersionList'])

    return all_images

def main():
    # Initialize the parser
    parser = argparse.ArgumentParser(description="List AWS images based on owner and
 platform")

    # Add the parameters/arguments
```

```
    parser.add_argument("--owner", default="Amazon", help="The owner of the images.
 Default is 'Amazon'.")
    parser.add_argument("--platform", default="Windows", help="The platform type of the
 images. Default is 'Windows'.")

    # Parse the arguments
    args = parser.parse_args()

    # Retrieve all images based on the provided owner and platform
    images = list_images(args.owner, args.platform)

    # Print the details of the images
    for image in images:
        print(f"Name: {image['name']}, Version: {image['version']}, ARN:
 {image['arn']}")

if __name__ == "__main__":
    main()
```

CloudFormation でのイメージ ARN の使用

Image Builder のイメージレシピは、出力イメージの意図した設定を実現するために必要な親イメージとコンポーネントを指定するブループリントです。AWS::ImageBuilder::ImageRecipe リソースを使用します。ParentImage 値をイメージ ARN に設定します。パイプラインが常に最新バージョンのイメージを使用するように、目的のイメージのバージョンレス ARN を使用してください。例えば、arn:aws:imagebuilder:us-east-1:aws:image/amazon-linux-2023-ecs-optimized-x86/x.x.x。以下の AWS::ImageBuilder::ImageRecipe リソースの定義は、Amazon マネージドイメージ ARN を使用しています。

```
ECSRecipe:
    Type: AWS::ImageBuilder::ImageRecipe
    Properties:
      Name: MyRecipe
      Version: '1.0.0'
      Components:
        - ComponentArn: [<The component arns of the image recipe>]
      ParentImage: "arn:aws:imagebuilder:us-east-1:aws:image/amazon-linux-2023-ecs-
optimized-x86/x.x.x"
```

これらの AWS::ImageBuilder::ImageRecipe リソースの詳細については、「AWS CloudFormation ユーザーガイド」を参照してください。

AWS::ImageBuilder::ImagePipeline リソースの Schedule プロパティを設定することで、パイプラインでの新しいイメージの作成を自動化できます。スケジュールには開始条件と cron 式が含まれています。詳細については、AWS CloudFormation ユーザーガイドの AWS::ImageBuilder::ImagePipelineを参照してください。

AWS::ImageBuilder::ImagePipeline次の例では、パイプラインで毎日協定世界時 (UTC) の午前 10 時にビルドを実行しています。以下の Schedule 値を設定します。

- PipelineExecutionStartCondition を EXPRESSION_MATCH_AND_DEPENDENCY_UPDATES_AVAILABLE に設定します。ビルドは、セマンティックバージョンでワイルドカード「x」を使用する親イメージやコンポーネントなどの依存リソースが更新された場合にのみ開始されます。これにより、ビルドにはそれらのリソースの最新の更新が確実に組み込まれます。
- ScheduleExpression を cron 式 (`0 10 * * ? *`) に設定します。

```
ECSPipeline:
    Type: AWS::ImageBuilder::ImagePipeline
    Properties:
      Name: my-pipeline
      ImageRecipeArn: <arn of the recipe you created in previous step>
      InfrastructureConfigurationArn: <ARN of the infrastructure configuration
 associated with this image pipeline>
      Schedule:
        PipelineExecutionStartCondition:
 EXPRESSION_MATCH_AND_DEPENDENCY_UPDATES_AVAILABLE
        ScheduleExpression: 'cron(0 10 * * ? *)'
```

Terraform でのイメージ ARN の使用

Terraform でパイプラインの親イメージとスケジュールを指定する方法は、AWS CloudFormation での方法と同様です。aws_imagebuilder_image_recipe リソースを使用します。parent_image 値をイメージ ARN に設定します。パイプラインが常に最新バージョンのイメージを使用するように、目的のイメージのバージョンレス ARN を使用してください。詳細については、Terraform ドキュメントの aws_imagebuilder_image_recipe を参照してください。

aws_imagebuilder_image_pipeline resource のスケジュール設定ブロックで、schedule_expression 引数の値を任意の cron 式に設定してパイプラインの実行頻度を指定し、pipeline_execution_start_condition を EXPRESSION_MATCH_AND_DEPENDENCY_UPDATES_AVAILABLE に設定します。詳細については、Terraform ドキュメントの aws_imagebuilder_image_pipeline を参照してください。

Amazon ECS で AWS 深層学習コンテナを使用する

AWSDeep Learning Containersは、 Amazon ECS の TensorFlow と Apache MXNet(インキュベーション)のモデルをトレーニングして提供するためのDockerイメージのセットを提供します。Deep Learning Containersは、TensorFlow、NVIDIA CUDA(GPUインスタンス用)、Intel MKL(CPUインスタンス用)のライブラリを使用し、最適化された環境を実現します。Deep Learning Containers のコンテナイメージは、Amazon ECR で利用でき、Amazon ECS タスク定義で参照できます。Deep Learning Containers と Amazon Elastic Inference を併用することで、推論コストを削減できます。

Elastic Inference を使用せずに Deep Learning Containers の使用を開始するには、「AWS Deep Learning AMIs デベロッパーガイド」の「Amazon ECS setup」を参照してください。

チュートリアル: Blue/Green デプロイを使用するサービスの作成

Amazon ECS では、Blue/Green デプロイが Amazon ECS コンソールのサービスの作成ウィザードに統合されています。詳細については、「Amazon ECS のローリング更新デプロイの作成」を参照してください。

以下のチュートリアルでは、AWS CLI で ブルー/グリーンデプロイタイプを使用する Fargate タスクを含む、Amazon ECS サービスを作成する方法を示します。

> **Note**
>
> ブルー/グリーンデプロイを実行するためのサポートが CloudFormation に追加されています。詳細については、[AWS CloudFormationユーザーガイド]の[CloudFormationを使用したCodeDeploy による Perform Amazon ECS ブルー/グリーンデプロイの実行]を参照してください。

前提条件

このチュートリアルでは、以下の前提条件をすでに満たしているものとします。

- AWS CLI の最新バージョンがインストールされ、設定されていること。AWS CLI のインストールまたはアップグレードの詳細については、「AWS Command Line Interface のインストール」を参照してください。
- 「Amazon ECS を使用するようにセットアップする」のステップを完了していること。
- IAM ユーザーに AmazonECS_FullAccess IAM ポリシー例で指定されている必要なアクセス権限があること。
- VPC およびセキュリティグループが使用できるように作成されていること。詳細については、「the section called “仮想プライベートクラウドを作成する”」を参照してください。
- Amazon ECS CodeDeploy IAM ロールが作成されます。詳細については、「Amazon ECS CodeDeploy IAM ロール」を参照してください。

ステップ 1: Application Load Balancer の作成

ブルー/グリーンデプロイタイプを使用している Amazon ECS サービスでは、Application Load Balancer、Network Load Balancer、Service Connect、またはヘッドレスのいずれかを使用する必要があります。マネージドトラフィックのシフトには、ロードバランサーまたは Service Connect が必要です。このチュートリアルでは、Application Load Balancer を使用します。

Application Load Balancer を作成するには

1. create-load-balancer コマンドを使用して、Application Load Balancer を作成します。異なるアベイラビリティーゾーンにある 2 つのサブネット、およびセキュリティグループを指定します。

   ```
   aws elbv2 create-load-balancer \
        --name bluegreen-alb \
        --subnets subnet-abcd1234 subnet-abcd5678 \
        --security-groups sg-abcd1234 \
        --region us-east-1
   ```

   出力には、次の形式でロードバランサーの Amazon リソースネーム (ARN) が含まれます。

   ```
   arn:aws:elasticloadbalancing:region:aws_account_id:loadbalancer/app/bluegreen-alb/
   e5ba62739c16e642
   ```

2. ターゲットグループを作成するには、create-target-group コマンドを使用します。このターゲットグループは、サービスで設定されている元のタスクにトラフィックをルーティングします。

```
aws elbv2 create-target-group \
     --name bluegreentarget1 \
     --protocol HTTP \
     --port 80 \
     --target-type ip \
     --vpc-id vpc-abcd1234 \
     --region us-east-1
```

出力には、以下の形式でターゲットグループの ARN が含まれます。

```
arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup/
bluegreentarget1/209a844cd01825a4
```

3. ターゲットグループにリクエストを転送するデフォルトルールが関連付けられた、ロードバランサーのリスナーを作成するには、create-listener コマンドを使用します。

```
aws elbv2 create-listener \
     --load-balancer-arn
 arn:aws:elasticloadbalancing:region:aws_account_id:loadbalancer/app/bluegreen-alb/
e5ba62739c16e642 \
     --protocol HTTP \
     --port 80 \
     --default-actions
 Type=forward,TargetGroupArn=arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup
bluegreentarget1/209a844cd01825a4 \
     --region us-east-1
```

出力には、以下の形式でリスナーの ARN が含まれます。

```
arn:aws:elasticloadbalancing:region:aws_account_id:listener/app/bluegreen-alb/
e5ba62739c16e642/665750bec1b03bd4
```

ステップ 2: Amazon ECS クラスターを作成する

create-cluster コマンドを使用して、使用するtutorial-bluegreen-clusterという名前のクラスターを作成します。

```
aws ecs create-cluster \
     --cluster-name tutorial-bluegreen-cluster \
```

```
    --region us-east-1
```

出力には、以下の形式でクラスターの ARN が含まれます。

```
arn:aws:ecs:region:aws_account_id:cluster/tutorial-bluegreen-cluster
```

ステップ 3: タスク定義を登録する

register-task-definition コマンドを使用して、Fargate と互換性のあるタスク定義を登録します。これには、awsvpc ネットワークモードを使用する必要があります。このチュートリアルで使用するタスク定義の例を以下に示します。

まず、以下の内容で fargate-task.json というファイルを作成します。タスク実行ロールの ARN を使用していることを確認します。詳細については、「Amazon ECS タスク実行IAM ロール」を参照してください。

```
{
    "family": "sample-fargate",
    "networkMode": "awsvpc",
    "containerDefinitions": [
        {
            "name": "sample-app",
            "image": "public.ecr.aws/docker/library/httpd:latest",
            "portMappings": [
                {
                    "containerPort": 80,
                    "hostPort": 80,
                    "protocol": "tcp"
                }
            ],
            "essential": true,
            "entryPoint": [
                "sh",
  "-c"
            ],
            "command": [
                "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample
 App</title> <style>body {margin-top: 40px; background-color: #333;} </style> </
head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1>
 <h2>Congratulations!</h2> <p>Your application is now running on a container in Amazon
 ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html && httpd-
foreground\""
```

```
            ]
        }
    ],
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "cpu": "256",
    "memory": "512"
}
```

次に、作成した fargate-task.json ファイルを使用して、タスク定義を登録します。

```
aws ecs register-task-definition \
     --cli-input-json file://fargate-task.json \
     --region us-east-1
```

ステップ 4: Amazon ECS サービスを作成する

サービスを作成するには、create-service コマンドを使用します。

まず、以下の内容で service-bluegreen.json というファイルを作成します。

```
{
    "cluster": "tutorial-bluegreen-cluster",
    "serviceName": "service-bluegreen",
    "taskDefinition": "tutorial-task-def",
    "loadBalancers": [
        {
            "targetGroupArn":
 "arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup/
bluegreentarget1/209a844cd01825a4",
            "containerName": "sample-app",
            "containerPort": 80
        }
    ],
    "launchType": "FARGATE",
    "schedulingStrategy": "REPLICA",
    "deploymentController": {
        "type": "CODE_DEPLOY"
    },
    "platformVersion": "LATEST",
    "networkConfiguration": {
```

```
        "awsvpcConfiguration": {
           "assignPublicIp": "ENABLED",
           "securityGroups": [ "sg-abcd1234" ],
           "subnets": [ "subnet-abcd1234", "subnet-abcd5678" ]
        }
    },
    "desiredCount": 1
}
```

次に、作成した service-bluegreen.json ファイルを使用してサービスを作成します。

```
aws ecs create-service \
     --cli-input-json file://service-bluegreen.json \
     --region us-east-1
```

出力には、以下の形式でサービスの ARN が含まれます。

```
arn:aws:ecs:region:aws_account_id:service/tutorial-bluegreen-cluster/service-bluegreen
```

ステップ 5: AWS CodeDeploy リソースを作成する

次のステップに従って、CodeDeploy アプリケーション、CodeDeploy デプロイグループの Application Load Balancer ターゲットグループ、および CodeDeploy デプロイグループを作成します。

CodeDeploy リソースを作成するには

1. create-application コマンドを使用して、CodeDeploy アプリケーションを作成します。ECS コンピューティングプラットフォームを指定します。

   ```
   aws deploy create-application \
        --application-name tutorial-bluegreen-app \
        --compute-platform ECS \
        --region us-east-1
   ```

   出力には、以下の形式でアプリケーションの ID が含まれます。

   ```
   {
       "applicationId": "b8e9c1ef-3048-424e-9174-885d7dc9dc11"
   }
   ```

2. create-target-group コマンドを使用して、CodeDeploy デプロイ グループの作成時に使用される 2 つ目のApplication Load Balancer ターゲットグループを作成します。

```
aws elbv2 create-target-group \
     --name bluegreentarget2 \
     --protocol HTTP \
     --port 80 \
     --target-type ip \
     --vpc-id "vpc-0b6dd82c67d8012a1" \
     --region us-east-1
```

出力には、以下の形式でターゲットグループの ARN が含まれます。

```
arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup/
bluegreentarget2/708d384187a3cfdc
```

3. create-deployment-group コマンドを使用して CodeDeploy デプロイ グループを作成します。

まず、以下の内容で tutorial-deployment-group.json というファイルを作成します。この例では、作成したリソースを使用します。serviceRoleArn には、Amazon ECS CodeDeploy IAM ロールの ARN を指定します。詳細については、「Amazon ECS CodeDeploy IAM ロール」を参照してください。

```
{
   "applicationName": "tutorial-bluegreen-app",
   "autoRollbackConfiguration": {
      "enabled": true,
      "events": [ "DEPLOYMENT_FAILURE" ]
   },
   "blueGreenDeploymentConfiguration": {
      "deploymentReadyOption": {
         "actionOnTimeout": "CONTINUE_DEPLOYMENT",
         "waitTimeInMinutes": 0
      },
      "terminateBlueInstancesOnDeploymentSuccess": {
         "action": "TERMINATE",
         "terminationWaitTimeInMinutes": 5
      }
   },
   "deploymentGroupName": "tutorial-bluegreen-dg",
   "deploymentStyle": {
      "deploymentOption": "WITH_TRAFFIC_CONTROL",
```

```
      "deploymentType": "BLUE_GREEN"
   },
   "loadBalancerInfo": {
      "targetGroupPairInfoList": [
        {
          "targetGroups": [
             {
                 "name": "bluegreentarget1"
             },
             {
                 "name": "bluegreentarget2"
             }
          ],
          "prodTrafficRoute": {
              "listenerArns": [
                  "arn:aws:elasticloadbalancing:region:aws_account_id:listener/
app/bluegreen-alb/e5ba62739c16e642/665750bec1b03bd4"
              ]
          }
        }
      ]
   },
   "serviceRoleArn": "arn:aws:iam::aws_account_id:role/ecsCodeDeployRole",
   "ecsServices": [
       {
           "serviceName": "service-bluegreen",
           "clusterName": "tutorial-bluegreen-cluster"
       }
   ]
}
```

次に、CodeDeploy デプロイ グループを作成します。

```
aws deploy create-deployment-group \
     --cli-input-json file://tutorial-deployment-group.json \
     --region us-east-1
```

出力には、以下の形式でデプロイグループの ID が含まれます。

```
{
    "deploymentGroupId": "6fd9bdc6-dc51-4af5-ba5a-0a4a72431c88"
}
```

ステップ 6: CodeDeploy デプロイを作成してする

以下のステップを従って、アプリケーション仕様ファイル (AppSpec ファイル) と CodeDeploy デプロイを作成してアップロードします。

CodeDeploy デプロイを作成してモニタリングするには

1. 以下の手順を使用して、AppSpec ファイルを作成してアップロードします。

   a. CodeDeploy デプロイグループの内容で `appspec.yaml` というファイルを作成します。この例では、チュートリアルの前半で作成したリソースを使用します。

   ```
   version: 0.0
   Resources:
     - TargetService:
         Type: AWS::ECS::Service
         Properties:
           TaskDefinition: "arn:aws:ecs:region:aws_account_id:task-
   definition/first-run-task-definition:7"
           LoadBalancerInfo:
             ContainerName: "sample-app"
             ContainerPort: 80
           PlatformVersion: "LATEST"
   ```

   b. s3 mb コマンドを使用して、AppSpec ファイルの Amazon S3 バケットを作成します。

   ```
   aws s3 mb s3://tutorial-bluegreen-bucket
   ```

   c. s3 cp コマンドを使用して、AppSpec ファイルを Amazon S3 バケットにアップロードします。

   ```
   aws s3 cp ./appspec.yaml s3://tutorial-bluegreen-bucket/appspec.yaml
   ```

2. 以下のステップを使用して、CodeDeploy デプロイを作成します。

   a. CodeDeploy デプロイの内容で `create-deployment.json` というファイルを作成します。この例では、チュートリアルの前半で作成したリソースを使用します。

   ```
   {
       "applicationName": "tutorial-bluegreen-app",
       "deploymentGroupName": "tutorial-bluegreen-dg",
       "revision": {
   ```

```
        "revisionType": "S3",
        "s3Location": {
            "bucket": "tutorial-bluegreen-bucket",
            "key": "appspec.yaml",
            "bundleType": "YAML"
        }
    }
}
```

b. デプロイを作成するには、create-deployment コマンドを使用します。

```
aws deploy create-deployment \
     --cli-input-json file://create-deployment.json \
     --region us-east-1
```

出力には、以下の形式でデプロイの ID が含まれます。

```
{
    "deploymentId": "d-RPCR1U3TW"
}
```

c. デプロイの詳細を取得するには、get-deployment-target コマンドで、前の出力からの deploymentId を指定します。

```
aws deploy get-deployment-target \
     --deployment-id "d-IMJU3A8TW" \
     --target-id tutorial-bluegreen-cluster:service-bluegreen \
     --region us-east-1
```

以下の出力に示しているように、ステータスが Succeeded になるまで、デプロイの詳細が取得され続けます。

```
{
    "deploymentTarget": {
        "deploymentTargetType": "ECSTarget",
        "ecsTarget": {
            "deploymentId": "d-RPCR1U3TW",
            "targetId": "tutorial-bluegreen-cluster:service-bluegreen",
            "targetArn": "arn:aws:ecs:region:aws_account_id:service/tutorial-
bluegreen-cluster/service-bluegreen",
            "lastUpdatedAt": 1543431490.226,
```

```
            "lifecycleEvents": [
                {
                    "lifecycleEventName": "BeforeInstall",
                    "startTime": 1543431361.022,
                    "endTime": 1543431361.433,
                    "status": "Succeeded"
                },
                {
                    "lifecycleEventName": "Install",
                    "startTime": 1543431361.678,
                    "endTime": 1543431485.275,
                    "status": "Succeeded"
                },
                {
                    "lifecycleEventName": "AfterInstall",
                    "startTime": 1543431485.52,
                    "endTime": 1543431486.033,
                    "status": "Succeeded"
                },
                {
                    "lifecycleEventName": "BeforeAllowTraffic",
                    "startTime": 1543431486.838,
                    "endTime": 1543431487.483,
                    "status": "Succeeded"
                },
                {
                    "lifecycleEventName": "AllowTraffic",
                    "startTime": 1543431487.748,
                    "endTime": 1543431488.488,
                    "status": "Succeeded"
                },
                {
                    "lifecycleEventName": "AfterAllowTraffic",
                    "startTime": 1543431489.152,
                    "endTime": 1543431489.885,
                    "status": "Succeeded"
                }
            ],
            "status": "Succeeded",
            "taskSetsInfo": [
                {
                    "identifer": "ecs-svc/9223370493425779968",
                    "desiredCount": 1,
                    "pendingCount": 0,
```

```
                    "runningCount": 1,
                    "status": "ACTIVE",
                    "trafficWeight": 0.0,
                    "targetGroup": {
                        "name": "bluegreentarget1"
                    }
                },
                {
                    "identifer": "ecs-svc/9223370493423413672",
                    "desiredCount": 1,
                    "pendingCount": 0,
                    "runningCount": 1,
                    "status": "PRIMARY",
                    "trafficWeight": 100.0,
                    "targetGroup": {
                        "name": "bluegreentarget2"
                    }
                }
            ]
        }
    }
}
```

ステップ 7: クリーンアップする

このチュートリアルが終了したら、使用していないリソースに対する料金が発生しないように、チュートリアルに関連付けられたリソースをクリーンアップします。

チュートリアルに関連付けられたリソースのクリーンアップ

1. delete-deployment-group コマンドを使用して、CodeDeploy デプロイグループを削除します。

```
aws deploy delete-deployment-group \
     --application-name tutorial-bluegreen-app \
     --deployment-group-name tutorial-bluegreen-dg \
     --region us-east-1
```

2. delete-application コマンドを使用して、 CodeDeploy アプリケーションを削除します。

```
aws deploy delete-application \
     --application-name tutorial-bluegreen-app \
```

```
    --region us-east-1
```

3. delete-service コマンドを使用して、Amazon ECS サービスを削除します。--force フラグを使用すると、タスクがゼロになっていなくてもサービスを削除できます。

```
aws ecs delete-service \
    --service arn:aws:ecs:region:aws_account_id:service/tutorial-bluegreen-cluster/service-bluegreen \
    --force \
    --region us-east-1
```

4. delete-cluster コマンドを使用して、Amazon ECS クラスターを削除します。

```
aws ecs delete-cluster \
    --cluster tutorial-bluegreen-cluster \
    --region us-east-1
```

5. s3 rm コマンドを使用して、Amazon S3 バケットから AppSpec ファイルを削除します。

```
aws s3 rm s3://tutorial-bluegreen-bucket/appspec.yaml
```

6. s3 rb コマンドを使用して、 Amazon S3 バケットを削除します。

```
aws s3 rb s3://tutorial-bluegreen-bucket
```

7. delete-load-balancer コマンドを使用して、Application Load Balancerを削除します。

```
aws elbv2 delete-load-balancer \
    --load-balancer-arn
 arn:aws:elasticloadbalancing:region:aws_account_id:loadbalancer/app/bluegreen-alb/e5ba62739c16e642 \
    --region us-east-1
```

8. delete-target-group コマンドを使用して、2 つのApplication Load Balancer ターゲットグループを削除します。

```
aws elbv2 delete-target-group \
    --target-group-arn
 arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup/bluegreentarget1/209a844cd01825a4 \
    --region us-east-1
```

```
aws elbv2 delete-target-group \
     --target-group-arn
 arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup/
bluegreentarget2/708d384187a3cfdc \
     --region us-east-1
```

Amazon ECS の Service Quotas

AWS アカウントを作成するときに、リージョンごとに AWS リソースに対してデフォルトのクォータ (制限とも呼ばれます) が設定されます。リソースのクォータの制限を超える処理を実行しようとすると、リクエストは失敗します。

Service Quotas コンソールは、AWS サービスのクォータを表示および管理したり、使用する多くのリソースのクォータの引き上げをリクエストしたりできる一元的な場所です。提供されるクォータとに関する情報に基づいてインフラストラクチャを管理してください。クォータの引き上げに対するリクエストは、クォータの引き上げが必要となる前に計画してください。

Amazon ECS と Fargate のクォータに関する詳細については、「Amazon Web Services 全般のリファレンス」の「Amazon ECS の エンドポイントとクォータ」を参照してください。Amazon ECS API での API スロットリングの詳細については、「Request throttling for the Amazon ECS API」(Amazon ECS API のスロットリングをリクエストする) を参照してください。

Amazon ECS マネージドインスタンスは Amazon EC2 のクォータに従います。詳細については、「Amazon EC2 ユーザーガイド」の「Amazon EC2 の Service Quotas」を参照してください。

AWS マネジメントコンソール での Amazon ECS とAWS FargateService Quotasの管理

Amazon ECS は、Service Quotas と統合されています。Service Quotas は、クォータを一元的な場所から表示および管理できる AWS サービスです。詳細については、「Service Quotas ユーザーガイド」の「Service Quotas とは」を参照してください。

Service Quotas を使用すると、Amazon ECS Service Quotasの値を簡単に調べることができます。

AWS マネジメントコンソール

AWS マネジメントコンソール を使用して Amazon ECS Service QuotasおよびFargate Service Quotasを表示するには

1. https://console.aws.amazon.com/servicequotas/ で Service Quotasのコンソールを開きます。
2. ナビゲーションペインで、[AWS のサービス] を選択します。
3. [AWS services] リストから、[Amazon Elastic Container Service (Amazon ECS)] または [AWS Fargate] を選択します。

Service quotas (Service Quotas)の一覧には、Service Quotas名、適用された値 (使用可能な場合)、AWS デフォルトのクォータ、クォータ値が調整可能かどうかが表示されます。

4. 説明など、Service Quotas に関する追加情報を表示するには、クォータ名を選択します。
5. (オプション) クォータの引き上げをリクエストするには、引き上げるクォータを選択し、[クォータ引き上げリクエスト] を選択します。必要な情報を入力または選択して、[リクエスト] を選択します。

AWS マネジメントコンソール を使用してさらにService Quotasの操作を行うには、Service Quotas ユーザーガイドを参照してください。クォータの引き上げをリクエストするには、Service Quotas ユーザーガイド の「クォータ引き上げリクエスト」を参照してください。

AWS CLI

AWS CLI を使用して Amazon ECS Service QuotasおよびFargate Service Quotasを表示するには

次のコマンドを実行して、デフォルトの Amazon ECS クォータを表示します。

```
aws service-quotas list-aws-default-service-quotas \
    --query 'Quotas[*].
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}' \
    --service-code ecs \
    --output table
```

次のコマンドを実行して、デフォルトの Fargate クォータを表示します。

```
aws service-quotas list-aws-default-service-quotas \
    --query 'Quotas[*].
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}' \
    --service-code fargate \
    --output table
```

次のコマンドを実行して、適用された Fargate クォータを表示します。

```
aws service-quotas list-service-quotas \
    --service-code fargate
```

> **Note**
>
> Amazon ECS は、適用されたクォータをサポートしていません。

AWS CLI を使用して Service Quotas を操作する方法の詳細については、「AWS CLI コマンドリファレンス Service Quotas」を参照してください。クォータの引き上げをリクエストするには、AWS CLICommand Reference の `request-service-quota-increase` コマンドを参照してください。

Amazon ECS Service Quotas と API スロットリング制限を処理する

Amazon ECS は、Elastic Load Balancing、AWS Cloud Map、Amazon EC2 など、いくつかの AWS のサービス と統合されています。この緊密な統合により、Amazon ECS ではサービスの負荷分散、Service Connect、タスクネットワーク、クラスターの自動スケーリングなどのいくつかの機能を利用できます。Amazon ECS および、統合されているその他の AWS のサービス はすべて、一貫したパフォーマンスと利用率を確保するために、サービスクォータと API レート制限を維持します。また、これらのサービスクォータは、必要以上のリソースを誤ってプロビジョニングすることを防ぎ、請求額を増やす可能性のある悪意のある行為からも保護します。

サービスクォータと AWS API のレート制限をよく理解しておけば、予期しないパフォーマンスの低下を心配することなく、ワークロードのスケーリングを計画できます。詳細については、「Amazon ECS API のリクエストのスロットリング」を参照してください。

Amazon ECS でワークロードをスケーリングする場合は、次のサービスクォータを検討することをお勧めします。

- AWS Fargate には、各 のタスクで同時に実行できるタスクの数を制限するクォータがあります。AWS リージョンAmazon ECS では、オンデマンドタスクと Fargate Spot タスクの両方にクォータが設定されています。各サービスクォータには、Fargate で実行するすべての Amazon EKS ポッドも含まれます。
- Amazon EC2 インスタンスで実行されるタスクの場合、各クラスターに登録できる Amazon EC2 インスタンスの最大数は 5,000 です。Auto Scaling グループの容量プロバイダーで Amazon ECS クラスターの自動スケーリングを使用する場合、またはクラスターの Amazon EC2 インスタンスを自分で管理する場合、このクォータがデプロイのボトルネックになる可能性があります。さらに

容量が必要な場合は、クラスターをさらに作成するか、サービスクォータの引き上げをリクエストすることができます。

- Auto Scaling グループのキャパシティプロバイダーで Amazon ECS クラスターの自動スケーリングを使用する場合は、サービスをスケーリングする際に `Tasks in the PROVISIONING state per cluster` クォータを考慮してください。このクォータは、キャパシティプロバイダーがキャパシティを増やすことができる各クラスターの PROVISIONING 状態にあるタスクの最大数です。多数のタスクを同時に起動すると、すぐにこのクォータに達してしまいます。たとえば、それぞれ数百のタスクを含む数十のサービスを同時にデプロイする場合です。この場合、クラスターのキャパシティが不足すると、キャパシティープロバイダーは新しいコンテナインスタンスを起動してタスクを配置する必要があります。キャパシティープロバイダーが追加の Amazon EC2 インスタンスを起動している間、Amazon ECS サービススケジューラーは引き続きタスクを並行して起動する可能性があります。ただし、クラスターのキャパシティが不十分なため、このアクティビティが抑制される可能性があります。Amazon ECS サービススケジューラは、新しいコンテナインスタンスが起動したときにタスク配置を再試行するためのバックオフおよびエクスポネンシャルスロットリング戦略を実装しています。その結果、デプロイやスケールアウトに時間がかかる可能性があります。このような状況を回避するには、以下のいずれかの方法でサービスのデプロイを計画してください。クラスター容量を増やす必要がないように多数のタスクをデプロイするか、新しいタスクの起動に備えて予備のクラスター容量を確保しておくかのどちらかです。

ワークロードをスケーリングする際には Amazon ECS サービスクォータを考慮することに加えて、Amazon ECS と統合されている他の AWS のサービス 用のサービスクォータも考慮してください。

Elastic Load Balancing

Fargate の Amazon ECS サービスは、Elastic Load Balancing を使用してタスク間でトラフィックを均等に分散するように設定できます。ロードバランサーの選択方法の詳細と推奨ベストプラクティスについては、ロードバランサーを使用して Amazon ECS サービストラフィックを分散する を参照してください。

Elastic Load Balancing のサービスクォータ

ワークロードをスケーリングするときは、以下の Elastic Load Balancing のサービスクォータを考慮してください。Elastic Load Balancing のサービスクォータのうち、ほとんどは調整可能で、Service Quotas コンソールでリクエストできます。

Application Load Balancer:

Application Load Balancer を使用する際、ユースケースによっては、以下のクォータ増加をリクエストする必要がある場合があります。

- Application Load Balancer の背後にあるターゲットの数である `Targets per Application Load Balancer` クォータ。
- ターゲットグループの背後にあるターゲットの数である `Targets per Target Group per Region` クォータ。

詳細については、「Application Load Balancer ユーザーガイド」の「Application Load Balancer のクォータ」を参照してください。

Network Load Balancer

Network Load Balancer に登録できるターゲットの数には、より厳しい制限があります。Network Load Balancer を使用する場合、クロスゾーンサポートを有効にする必要があることがよくあります。これには、各 Network Load Balancer のアベイラビリティーゾーンあたりの `Targets per Availability Zone Per Network Load Balancer` 最大ターゲット数に対する追加のスケーリング制限が伴います。詳細については、「Network Load Balancer ユーザーガイド」の「Network Load Balancer のクォータ」を参照してください。

Elastic Load Balancing API のスロットリング

ロードバランサーを使用するように Amazon ECS サービスを設定する場合、サービスが正常であると見なされるには、ターゲットグループのヘルスチェックに合格する必要があります。これらのヘルスチェックを実行するために、Amazon ECS はユーザーに代わって Elastic Load Balancing API 操作を呼び出します。アカウントにロードバランサーが設定されたサービスが多数ある場合、特に `RegisterTarget`、`DeregisterTarget`、`DescribeTargetHealth` Elastic Load Balancing API 操作のスロットリングが発生する可能性があるため、サービスのデプロイが遅くなる可能性があります。スロットリングが発生すると、Amazon ECS サービスのイベントメッセージにスロットリングエラーが発生します。

AWS Cloud Map API でスロットリングが発生した場合は、AWS Cloud Map API のスロットリング制限を引き上げる方法について サポート までお問い合わせください。これらのスロットリングエラーの監視とトラブルシューティングの詳細については、「Amazon ECS のスロットリングに関する問題を処理する」を参照してください。

Elastic Network Interface

タスクが awsvpc ネットワークモードを使用する場合、Amazon ECS はタスクごとに独自の Elastic Network Interface (ENI) をプロビジョニングします。Amazon ECS サービスが Elastic Load Balancing ロードバランサーを使用する場合、これらのネットワークインターフェイスは、サービスで定義された適切なターゲットグループのターゲットとしても登録されます。

Elastic Network Interface のサービスクォータ

awsvpc ネットワークモードを使用するタスクを実行すると、固有のエラスティックネットワークインターフェイスが各タスクにアタッチされます。インターネット経由でこれらのタスクにアクセスする必要がある場合は、タスクのエラスティックネットワークインターフェイスにパブリック IP アドレスを割り当てます。Amazon ECS ワークロードをスケーリングするときは、次の 2 つの重要なクォータを考慮してください。

- アカウントの AWS リージョン におけるネットワークインターフェイスの最大数である `Network interfaces per Region` クォータ。
- AWS リージョン に含まれる Elastic IP アドレスの最大数である `Elastic IP addresses per Region` クォータ。

これらのサービスクォータは両方とも調整可能で、Service Quotas コンソールから増加をリクエストできます。詳細については、「Amazon Virtual Private Cloud ユーザーガイド」の「Amazon VPC サービスクォータ」を参照してください。

Amazon EC2 インスタンスでホストされている Amazon ECS ワークロードの場合、awsvpc ネットワークモードを使用するタスクを実行するときは、各 Amazon EC2 インスタンスの最大ネットワークインスタンスの数である `Maximum network interfaces` サービスクォータを考慮してください。このクォータは、1 つのインスタンスに配置できるタスクの数を制限します。クォータを調整することはできず、Service Quotas コンソールでは使用できません。詳細については、「Amazon EC2 ユーザーガイド」の「各インスタンスタイプのネットワークインターフェイスあたりの IP アドレス」を参照してください。

Amazon EC2 インスタンスにアタッチできるネットワークインターフェイスの数は変更できませんが、エラスティックネットワークインターフェイスのトランキング機能を使用して利用可能なネットワークインターフェイスの数を増やすことができます。たとえば `c5.large` インスタンスにはデフォルトでネットワークインターフェイスを最大 3 つアタッチできます。このインスタンスのプライマリネットワークインターフェイスも、1 個としてカウントされます。そのため、このインスタンスに追加で 2 個のネットワークインターフェイスをアタッチできます。awsvpc ネットワークモード

を使用する各タスクにはネットワークインターフェイスが必要なため、通常このインスタンスタイプでは、これらのタスクを 2 つのみ実行できます。これにより、クラスターの容量が十分に活用されない可能性があります。エラスティックネットワークインターフェイスでトランキングを有効にすると、ネットワークインターフェイスの密度を上げて、各インスタンスにより多くのタスクを配置できます。トランキングをオンにすると、1 つの `c5.large` インスタンスに最大 12 のネットワークインターフェースを設定できます。インスタンスはプライマリネットワークインターフェイスを持ち、Amazon ECS は「トランク」ネットワークインターフェイスを作成し、アタッチします。その結果、この構成では、デフォルトの 2 つのタスクではなく、10 のタスクをインスタンスで実行できます。詳細については、「Amazon ECS Linux コンテナインスタンスのネットワークインターフェイスを増やす」を参照してください。

エラスティックネットワークインターフェイス API のスロットリング

`awsvpc` ネットワークモードを使用するタスクを実行する場合、Amazon ECS は次の Amazon EC2 API に依存します。これらの API にはそれぞれ異なる API スロットリングがあります。詳細については、「Amazon EC2 API リファレンス」の「Amazon EC2 API におけるリクエストのスロットリング」を参照してください。

- CreateNetworkInterface
- AttachNetworkInterface
- DetachNetworkInterface
- DeleteNetworkInterface
- DescribeNetworkInterfaces
- DescribeVpcs
- DescribeSubnets
- DescribeSecurityGroups
- DescribeInstances

エラスティックネットワークインターフェイスのプロビジョニングワークフロー中に Amazon EC2 API コールがスロットリングされた場合、Amazon ECS サービススケジューラは自動的にエクスポネンシャルバックオフを行って再試行します。これらの自動廃止により、タスクの起動が遅れ、デプロイ速度が遅くなることがあります。API スロットリングが発生すると、サービスイベントメッセージにメッセージ `Operations are being throttled. Will try again later.` が表示されます。Amazon EC2 API でスロットリングが継続的に発生する場合は、API のスロットリング制限を引き上げる方法について サポート までお問い合わせください。スロットリングエラーの監視とトラブルシューティングの詳細については、「スロットリング問題の処理」を参照してください。

AWS Cloud Map

Amazon ECS サービス検出と Service Connect では、AWS Cloud Map API を使用して Amazon ECS サービスの名前空間を管理します。サービスに多数のタスクがある場合は、以下の推奨事項を考慮してください。

AWS Cloud Map サービスクォータ

Amazon ECS サービスがサービス検出または Service Connect を使用するように設定されている場合、サービスの最大タスク数である `Tasks per service` クォータは、そのサービスの最大インスタンス数である AWS Cloud Map `Instances per service` サービスクォータの影響を受けます。特に、AWS Cloud Map サービスクォータにより、実行できるインスタンスの数は、サービスあたり最大 1,000 インスタンスに減少します。AWS Cloud Map のクォータは変更できません。詳細については、「AWS Cloud Map サービスクォータ」を参照してください。

AWS Cloud Map API スロットリング

Amazon ECS はユーザーに代わって`ListInstances`、`GetInstancesHealthStatus`、`RegisterInstance`、および `DeregisterInstance` AWS Cloud Map API を呼び出します。これらはサービス検出を支援し、タスクを起動するとヘルスチェックを実行します。多数のタスクを伴うサービス検出を使用する複数のサービスを同時にデプロイすると、AWS Cloud Map API のスロットリング制限を超える可能性があります。この場合、Amazon ECS サービスのイベントメッセージに次のようなメッセージが表示される場合があります: Amazon ECS サービスイベントで `Operations are being throttled. Will try again later` が発生しました。デプロイとタスクの起動速度が遅くなっています。AWS Cloud Map には、これらの API のスロットリング制限は記録されていません。これらのスロットリングが発生した場合は、API のスロットリング制限を引き上げる方法について サポート までお問い合わせください。これらのスロットリングエラーの監視とトラブルシューティングに関する推奨事項については、「Amazon ECS のスロットリングに関する問題を処理する」を参照してください。

Amazon ECS の API の使用

AWS マネジメントコンソール と AWS Command Line Interface (AWS CLI) の他に、Amazon ECS には API も用意されています。API を使用して、Amazon ECS リソースを管理するタスクを自動化できます。

- Amazon ECS リソースごとの API オペレーションの一覧については、「Amazon ECS リソースによるアクション」を参照してください。
- API オペレーションのアルファベット順リストについては、「アクション」を参照してください。
- データ型のアルファベット順リストについては、「データ型」を参照してください。
- 共通クエリパラメータのリストについては、「共通パラメータ」を参照してください。
- エラーコードの説明については、「共通エラー」を参照してください。

AWS CLI の詳細については、「Amazon ECS の AWS Command Line Interface リファレンス」を参照してください。

Amazon ECS リソースによるアクション

次のセクションでは、リソース別の API アクションを一覧表示します。

トピック

- アカウント設定のアクション
- クラスターのアクション
- クラスターキャパシティプロバイダーのアクション
- コンテナのアクション
- コンテナエージェントのアクション
- コンテナインスタンスのアクション
- 名前空間のアクション
- サービスアクション
- サービスデプロイのアクション
- サービスリビジョンのアクション

- タスク定義のアクション
- タスクのアクション
- タスクセットのアクション
- タグのアクション

アカウント設定のアクション

アカウント設定では次の API アクションを使用できます。

- DeleteAccountSetting
- ListAccountSettings
- PutAccountSetting
- PutAccountSettingDefault

クラスターのアクション

クラスターでは次の API アクションを使用できます。

- CreateCluster
- DeleteCluster
- DescribeClusters
- ListClusters
- UpdateCluster
- UpdateClusterSettings

クラスターキャパシティプロバイダーのアクション

クラスターキャパシティプロバイダーでは次の API を使用できます。

- CreateCapacityProvider
- DeleteCapacityProvider
- DescribeCapacityProviders
- PutClusterCapacityProviders

- UpdateCapacityProvider

コンテナのアクション

コンテナでは次の API を使用できます。

- ExecuteCommand

コンテナエージェントのアクション

コンテナエージェントでは次の API を使用できます。

- UpdateContainerAgent

コンテナインスタンスのアクション

コンテナインスタンスでは次の API を使用できます。

- DeregisterContainerInstance
- DescribeContainerInstances
- DeleteAttributes
- ListAttributes
- ListContainerInstances
- PutAttributes
- UpdateContainerInstancesState

名前空間のアクション

> **Note**
>
> 名前空間は AWS Cloud Map リソースです。ECS Service Connect または ECS サービス検出機能を使用するには、Amazon ECS で名前空間が必要となります。Amazon ECS は AWS Cloud Map から名前空間をリクエストしてから、AWS マネジメントコンソール に名前空間を表示します。追加の設定オプションには、AWS Cloud Map コンソールまたは API を使用します。

名前空間では次の API を使用できます。

- CreateCluster

サービスアクション

サービスでは次の API を利用できます。

- CreateService
- DeleteService
- DescribeServices
- ListServices
- UpdateService

サービスデプロイのアクション

サービスデプロイでは次の API を使用できます。

- DescribeServiceDeployments
- ListServiceDeployments
- StopServiceDeployment

サービスリビジョンのアクション

サービスリビジョンでは次の API を使用できます。

- DescribeServiceRevisions

タスク定義のアクション

タスク定義では次の API を使用できます。

- DeleteTaskDefinitions
- DeregisterTaskDefinition
- DescribeTaskDefinition

- ListTaskDefinitionFamilies
- ListTaskDefinitions
- RegisterTaskDefinition

タスクのアクション

タスクでは次の API を使用できます。

- DescribeTasks
- GetTaskProtection
- ListTasks
- RunTask
- StartTask
- StopTask
- UpdateTaskProtection

タスクセットのアクション

タスクセットでは次の API を使用できます。

- CreateTaskSet
- DeleteTaskSet
- DescribeTaskSets
- UpdateServicePrimaryTaskSet
- UpdateTaskSet

タグのアクション

タグでは次の API を使用できます。

- ListTagsForResource
- TagResource
- UntagResource

Amazon ECS API のリクエストスロットリング

Amazon Elastic Container Service では、リージョンごとに、各 AWS アカウントのすべての API リクエストをスロットリングします。これは、一貫したパフォーマンスを維持し、Amazon ECS のすべてのお客様に平等にサービスをご利用いただくために行われています。スロットリングにより、Amazon ECS API への呼び出しが、Amazon ECS および統合するその他の AWS サービスの両方で許可される API リクエストの最大クォータを超えないようになります。API コールは、送信元が以下であるかどうかにかかわらず、リクエストクォータの対象となります。

- サードパーティーのアプリケーション
- コマンドラインツール
- Amazon ECS コンソール

API スロットリングクォータを超えると、ThrottlingException エラーコードが返されます。

```
An error occurred (ThrottlingException) when calling the DescribeClusters operation
 (reached max retries: 4): Rate exceeded.
com.amazonaws.services.ecs.model.AmazonECSException: Rate exceeded (Service: AmazonECS;
 Status Code: 400; Error
Code: ThrottlingException; Request ID: 5ed90669-e454-464d-9b2f-6523bc86f537; Proxy:
 null)
```

スロットリングの適用方法

Amazon ECS ではトークンバケットアルゴリズムを使用して、API スロットリングを実装します。このアルゴリズムでは、アカウントには、特定の数のトークンを保持するバケットがあります。バケット内のトークンの数は、特定の秒におけるスロットリングクォータを表します。

Amazon ECS は、アカウント内のすべての Amazon ECS API の API リクエスト送信のレートをリージョンごとに調べ、持続とバーストの 2 種類の API スロットリングクォータを適用します。持続レートは、 オペレーションで 1 秒あたりに許可される API リクエストの平均数です。バーストレートは、任意の 1 秒で許可される API リクエストの最大数です。バーストレートでは、持続レートよりも多くの API リクエストを定期的に作成できます。その後、Amazon ECS は、時間の経過とともに許可される API リクエストのレートが持続レートに安定するまで、後続の API リクエストをスロットリングします。トークンバケットアルゴリズムでは、バケットの最大容量はバーストレートを表します。バケットリフィルレートは持続レートです。これらの用語を使用して、次の例の Amazon ECS API リクエストスロットリングの図を紹介します。

実行する API リクエストの数でスロットリングされ、各リクエストはトークンバケットから 1 つのトークンを削除します。例えば、DescribeClusters API などのクラスター読み取りアクションのバケットサイズは 50 トークンであるため、1 秒間に最大 50 個の DescribeClusters リクエストを行うことができます。1 秒間に 50 リクエストを超えると、スロットリングされ、その秒以内の残りのリクエストは失敗します。

バケットは設定されたレートで自動的に補充されます。バケットが最大容量に達していない場合、最大容量に達するまで、設定した数のトークンが毎秒追加されます。リフィルトークンが到着したときにバケットが満杯である場合、リフィルトークンは破棄されます。バケットは最大数以上のトークンを保持できません。例えば、クラスター読み取りアクションのバケットサイズ (DescribeClusters API など) が 50 トークンで、リフィルレートが 1 秒あたり 20 トークンであるとします。1 秒間に 50 の DescribeClusters API リクエストを行うと、バケットはすぐにゼロトークンに削減されます。バケットは、最大容量の 50 トークンに達するまで、毎秒最大 20 トークン補充されます。つまり、以前空だったバケットは 2.5 秒後に最大容量に達します。

API リクエストを行う前に、バケットが完全にいっぱいになるまで待つ必要はありません。トークンはバケットに追加されるとすぐに使用できます。補充トークンをすぐに使用すると、バケットが最大容量に達することはありません。例えば、クラスター読み取りアクションのバケットサイズ (DescribeClusters API など) が 50 トークンで、リフィルレートが 1 秒あたり 20 トークンであるとします。1 秒あたり 50 個の API リクエストを行ってバケットを使い果たした場合、1 秒あたり 20 個の API リクエストを引き続き行うことができます。バケットは、1 秒あたり 20 個未満の API リクエストを行う場合のみ、最大容量に補充できます。

リクエストトークンバケットのサイズとリフィルレート

リクエストレート制限の目的で、API アクションはカテゴリにグループ化されます。カテゴリ内のすべての API アクションは、同じトークンバケットを共有します。例えば、DescribeClusters と ListClusters API が、容量が 50、補充レートが 20 のクラスター読み取りアクションバケットを共有しているとします。この場合、すべてのクラスター読み取りアクションの API リクエストの累積数は、50 件の API リクエストの同一バーストレートクォータによって調整されます。したがって、1 秒あたり 25 の DescribeClusters と 25 の ListClusters API リクエスト、または 30 の DescribeClusters と 20 の ListClusters、または 50 の DescribeClusters と 0 の ListClusters、または 0 の DescribeClusters と 50 の ListClusters のリクエストを実行できますが、同時に 50 の DescribeClusters と 50 の ListClusters リクエストを行うことはできません。持続レートは、バケット内のすべての API リクエストに同様に累積的に適用されます。

次の表は、すべての AWS リージョンのバケット容量 (またはバースト) とリフィルレート (または持続) を示しています。すべての API アクションカテゴリは、リージョンごとに各 AWS アカウントのレートクォータを適用します。

API アクションカテゴリ	アクション	バケット最大容量 (またはバーストレート)	バケット補充率 (または持続率)
クラスター変更アクション	• CreateCluster • DeleteCluster • PutClusterCapacityProviders • UpdateCluster • UpdateClusterSettings	20	1
クラスター読み取りアクション	• DescribeClusters • ListClusters	50	20
タスク定義変更アクション	• DeregisterTaskDefinition • RegisterTaskDefinition	20	1
タスク定義読み取りアクション	• DescribeTaskDefinition • ListTaskDefinitions • ListTaskDefinitionFamilies	50	20
タスク定義削除アクション	• DeleteTaskDefinitions	5	1
キャパシティプロバイダー変更アクション	• CreateCapacityProvider • DeleteCapacityProvider • UpdateCapacityProvider	10	1
キャパシティプロバイダー読み取りアクション	• DescribeCapacityProviders	50	20
タグ変更アクション	• TagResource	20	10

API アクションカテゴリ	アクション	バケット最大容量 (またはバーストレート)	バケット補充率 (または持続率)
	• UntagResource		
タグ読み取りアクション	• ListTagsForResource	50	20
変更アクションの設定	• DeleteAccountSetting • PutAccountSetting • PutAccountSettingDefault	10	1
読み取りアクションの設定	• ListAccountSettings	50	20
クラスターリソース変更アクション	• DeleteAttributes • DeregisterContainerInstance • ExecuteCommand • PutAttributes • RunTask[1] • StartTask • StopTask • UpdateContainerAgent • UpdateContainerInstancesStates	100	40
クラスターリソース読み取りアクション	• DescribeContainerInstances • DescribeTasks • ListAttributes • ListContainerInstances • ListTasks	100	20

API アクションカテゴリ	アクション	バケット最大容量 (またはバーストレート)	バケット補充率 (または持続率)
エージェント変更アクション	• RegisterContainerInstance • SubmitAttachmentStateChanges • SubmitContainerStateChange • SubmitTaskStateChange	200	120
サービス変更アクション	• CreateService • DeleteService • UpdateService	50	5
サービス読み取りアクション	• DescribeServices • ListServices	100	20
サービスデプロイアクション	• DescribeServiceDeployments • ListServiceDeployments	50	20
サービスリビジョンアクション	• DescribeServiceRevisions	50	20
タスク保護アクション	• UpdateTaskProtection • GetTaskProtection	200	80
クラスターサービスリソース読み取りアクション	• ListServicesByNamespace	10	1

[1] AWS Fargate は、「Amazon ECS デベロッパーガイド」のこちらに記載されているレートに Amazon ECS RunTask APIをさらにスロットリングします。

API スロットリングのクォータの調整

AWS アカウントの API スロットリングクォータの引き上げをリクエストできます。クォータの調整をリクエストするには、AWS サポート センターまでお問い合わせください。

API スロットリングの処理

エラーの再試行とエクスポネンシャルバックオフ戦略を実装して、スロットリングエラーがワークロードに与える影響を回避できます。AWS SDK を使用する場合、自動再試行ロジックは既に組み込まれており、設定可能です。詳細については、次のリソースを参照してください。

- 「AWS 全般リファレンスガイド」の「AWS でのエラー再試行とエクスポネンシャルバックオフ」を参照してください。
- 「エクスポネンシャルバックオフとジッター」のブログ記事
- Amazon Builders' Library の「ジッターを伴うタイムアウト、再試行、およびバックオフ」の記事

べき等性の確保

変更オペレーションを実行すると、リソースが変更された後にタイムアウトやサーバーの問題が発生するため、例外が表示されることがあります。これにより、ミューテーションが発生したかどうかを判断することが難しくなり、複数回の再試行につながる場合があります。ただし、元のオペレーションとその後の再試行が実際にミューテーションを実行した場合、スタックの変更を適用したか、意図したよりも多くのリソースを作成している可能性があります。べき等性により、 オペレーションはリソースを 1 回しか変更しません。べき等性リクエストでは、元のリクエストが変更された場合、その後の再試行は追加のミューテーションを実行することなく正しく完了します。

トピック

- Amazon ECS のべき等性
- RunTask のべき等性
- 例
- べき等リクエストの再試行の推奨事項

Amazon ECS のべき等性

次の API アクションは、オプションでクライアントトークンを使用したべき等性をサポートしています。対応する AWS CLI コマンドも、クライアントトークンを使用したべき等性をサポートしてい

ます。クライアントトークンは、大文字と小文字を区別する一意の文字列です。これらのアクションのいずれかを使用してべき等 API リクエストを行うには、リクエストでクライアントトークンを指定します。同じクライアントトークンを他の API リクエストに再利用しないでください。同じクライアントトークンと同じパラメータを使用して正常に完了したリクエストを再試行すると、それ以上のアクションを実行せずに再試行が成功します。

クライアントトークンを使用したべき等性

- CreateService

  クライアントトークンは、33 ~ 126 (含む) の範囲の最大 36 文字の ASCII 文字です。

- CreateTaskSet

  クライアントトークンは、33 ~ 126 (含む) の範囲の最大 36 文字の ASCII 文字です。

- RunTask

  クライアントトークンは、33 ~ 126 (含む) の範囲の最大 64 文字の ASCII 文字です。

べき等性のタイプ

- クラスター – 同じクラスターに同じトークンを持つリクエストはべき等性です。例えば、ClientToken A は、クラスター X の RunTask リクエストに対して 1 回のみリクエストパラメータとして使用できます。他のクラスターへの RunTask リクエストは、別のリクエストと見なされます。したがって、クラスター Y の RunTask リクエストに ClientToken A を使用できます。

RunTask のべき等性

RunTask API は、クライアントトークンを使用したべき等性に対応しています。クライアントトークンは、API リクエストを行うときに指定する一意の文字列です。API リクエストが正常に完了した後で、同じクライアントトークンと同じリクエストパラメータを使用して API リクエストを再試行すると、元のリクエストの結果が返されます。同じクライアントトークンを使用して成功したリクエストを再試行しても、リージョンまたはアベイラビリティーゾーン以外の 1 つ以上のパラメータが異なる場合、再試行は `ConflictException` で失敗します。独自のクライアントトークンを指定しない場合、AWS SDK および AWS Command Line Interface はリクエストのクライアントトークンを自動生成し、それがべき等性であることを確認します。クライアントトークンには、33 ~ 126 (含む) の範囲の最大 64 文字の ASCII 文字を含む任意の文字列を指定できます。

RunTask クライアントトークンの有効期限 (TTL) は 24 時間です。同じクライアントトークンを異なるリクエストに再利用しないでください。クライアントトークンの最大 TTL は、次の 2 つの値のいずれか低い方に対して有効です。

- 24 時間
- リソースの有効期間 + 1 時間

  リソースの有効期間は、タスクが作成されたタイムスタンプから、最後のステータス (lastStatus) が STOPPED に移行したタイムスタンプです。RunTask を使用して 1 つ以上のタスクを起動する場合、リソースの有効期間は、STOPPED に移行した最後のタスクの有効期間と等しくなります。

RunTask 再試行ルールとレスポンス

5xx 例外を受け取ったためにリクエストを再試行すると、再試行された成功レスポンスには通常、元のリクエストが返すすべての情報が含まれます。1 時間以内に停止されたタスクには、タスク ARN、最終ステータス、目的のステータスのみが含まれます。

次に示すのは、実行中のタスクが 1 つ、停止したタスクが 1 つ、起動に失敗したタスクが 1 つある場合の再試行からのレスポンスのスニペットの例です。

```
{
  "failures": [
    {
      "arn": "arn:aws:ecs:us-east-1:123456789012:container/4df26bb4-f057-467b-
a079-961675296e64",
      "reason": "RESOURCE:MEMORY"
    }
  ],
  "tasks": [
    {
      "desiredStatus": "RUNNING",
      "taskArn": "arn:aws:ecs:us-east-1:123456789012:task/default/fdf2c302-468c-4e55-
b884-5331d816e7fb",
      ...
    },
    {
      "taskArn": "arn:aws:ecs:us-east-1:123456789012:task/default/fdf2c302-468c-4e55-
b884-5331d819999",
      "lastStatus": "STOPPED",
      ...
```

```
      }
  ]
}
```

1 時間以上経過したエラーには、失敗したタスクの数のみが含まれます。

例

AWS CLI コマンドの例

AWS CLI コマンドをべき等にするには、`--client-token` オプションを追加します。

例: create-service

次の create-service コマンドは、クライアントトークンを含むため、べき等性を使用します。

```
aws ecs create-service \
    --cluster MyCluster \
    --service MyService \
    --task-definition MyTaskDefinition:2 \
    --desired-count 2 \
    --launch-type FARGATE \
    --platform-version LATEST \
    --network-configuration
 "awsvpcConfiguration={subnets=["subnet-12344321"],securityGroups=["sg-12344321"],assignPublicI
 \
    --client-token 550e8400-e29b-41d4-a716-44665544
```

例: create-service

次の create-service コマンドは、クライアントトークンを含むため、べき等性を使用します。

```
aws ecs create-task-set \
    --cluster MyCluster \
    --service MyService \
    --task-definition MyTaskDefinition:2 \
    --network-configuration
 "awsvpcConfiguration={subnets=["subnet-12344321"],securityGroups=["sg-12344321"]}" \
    --client-token 550e8400-e29b-41d4-a716-44665544
```

例: run-task

次の run-task コマンドは、クライアントトークンを含むため、べき等性を使用します。

```
aws ecs run-task \
    --cluster MyCluster \
    --task-definition MyTaskDefinition:2 \
    --client-token 550e8400-e29b-41d4-a716-446655440000
```

API リクエストの例

API リクエストをべき等にするには、clientToken パラメータを追加します。

例: CreateService

次の CreateService API リクエストは、クライアントトークンを含むため、べき等性を使用します。

```
POST / HTTP/1.1
Host: ecs.us-east-1.amazonaws.com
Accept-Encoding: identity
Content-Length: 87
X-Amz-Target: AmazonEC2ContainerServiceV20141113.CreateService
X-Amz-Date: 20150429T170125Z
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
  "serviceName": "MyService",
  "taskDefinition": "MyTaskDefinition:2",
  "desiredCount": 10,
   "capacityProviderStrategy": [
      {
         "base": "number",
         "capacityProvider": "FARGATE",
         "weight": 1
      }
   ],
   "capacityProviderStrategy": [
      {
         "base": "number",
         "capacityProvider": "FARGATE_SPOT",
         "weight": 1
      }
   ],
   "clientToken": "550e8400-e29b-41d4-a716-44665544"
```

```
}
```

例: CreateTaskSet

次の CreateTaskSet API リクエストは、クライアントトークンを含むため、べき等性を使用します。

```
POST / HTTP/1.1
Host: ecs.us-east-1.amazonaws.com
Accept-Encoding: identity
Content-Length: 87
X-Amz-Target: AmazonEC2ContainerServiceV20141113.CreateTaskSet
X-Amz-Date: 20150429T170125Z
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
  "serviceName": "MyService",
  "taskDefinition": "mytask:1",
  "desiredCount": 1,
   "capacityProviderStrategy": [
      {
         "base": "number",
         "capacityProvider": "FARGATE",
         "weight": 1
      }
   ],
   "capacityProviderStrategy": [
      {
         "base": "number",
         "capacityProvider": "FARGATE_SPOT",
         "weight": 1
      }
   ],
    "clientToken": "550e8400-e29b-41d4-a716-44665544"
}
```

例: RunTask

次の RunTask API リクエストは、クライアントトークンを含むため、べき等性を使用します。

```
POST / HTTP/1.1
Host: ecs.us-east-1.amazonaws.com
```

```
Accept-Encoding: identity
Content-Length: 45
X-Amz-Target: AmazonEC2ContainerServiceV20141113.RunTask
X-Amz-Date: 20161121T215740Z
User-Agent: aws-cli/1.11.13 Python/2.7.12 Darwin/16.1.0 botocore/1.4.66
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
  "count": 1,
  "taskDefinition": "mytask:1",
  "clientToken": "550e8400-e29b-41d4-a716-446655440000"
}
```

べき等リクエストの再試行の推奨事項

次の表は、べき等 API リクエストに対して返される一般的な応答と、再試行の推奨事項を示しています。

応答	推奨事項	コメント
200 (OK)	再試行しないでください	元のリクエストは正しく完了しています。それ以降に再試行しても正常に戻ります。
400 シリーズ応答コード (クライアントエラー)	再試行しないでください	次のうち、リクエストに問題があります。 • 無効なパラメータまたはパラメータの組み合わせが含まれています。 • 権限のないアクションまたはリソースを使用しています。 • 状態の変更処理中のリソースを使用しています。 リクエストに状態の変更処理中のリソースが含まれている場合、リクエストを再試行すると成功する可能性があります。
500 シリーズ応答コード (サーバーエラー)	再試行	このエラーは AWS サーバー側の問題によって発生し、通常は一時的なものです。適切なバッ

応答	推奨事項	コメント
		クオフ戦略を使用してリクエストを繰り返してください。

ドキュメント履歴

以下の表は、Amazon Elastic Container Service デベロッパーガイドの主な更新や新機能を示しています。また、お客様からいただいたフィードバックに対応するために、ドキュメントを頻繁に更新しています。

変更	説明	日付
アクセス許可を AmazonECS_FullAccess マネージドポリシーに追加します。	AmazonECS_FullAccess マネージド IAM ポリシーが更新され、cloudformation:TagResource および cloudformation:UntagResource アクセス許可が含まれるようになりました。これらのアクセス許可により、スタックの作成時または更新時に CloudFormation スタックにタグを追加できます。	2026 年 9 月 2 日
アクセス許可を AmazonECSServiceRolePolicy マネージドポリシーに追加します。	AmazonECSServiceRolePolicy マネージド IAM ポリシーが更新され、autoscaling:TerminateInstanceInAutoScalingGroup アクセス許可が含まれるようになりました。このアクセス許可により、Amazon ECS は Amazon ECS クラスターへの登録に失敗したカスタマー ASG の Amazon EC2 インスタンスを終了できます。	2026 年 4 月 24 日
AmazonECSInfrastructureRolePolicyForManagedInstances への許可を更新	パーティション間で EC2 サービス名の一貫性を維持するために許可を更新しました。この更新によって既存の許可モデルが変更されることはなく、追加のアクセス権も付与されません。	2026 年 2 月 26 日
AmazonECSInfrastructureRolePolicyForManagedInstances への許可を更新	以下の許可を追加して AmazonECSInfrastructureRolePolicyForManagedInstances ポリシーを更新し、Amazon ECS マネージドインスタンスでキャパシティ予約がサポートされるようにしました。 •	2026 年 2 月 24 日

変更	説明	日付
	Amazon ECS がキャパシティ予約のグループを取得できるようにする `resource-groups:ListGroupResources` 許可が追加されました。このコマンドを実行するには、`cloudformation:ListStackResources` 許可、`cloudformation:DescribeStacks` 許可、および `tag:GetResources` 許可も必要です。 • Amazon ECS が Amazon ECS マネージドインスタンスとの統合に必要なキャパシティ予約の詳細情報を収集できるようにする `ec2:DescribeCapacityReservations` 許可が追加されました。 • Amazon ECS が対象の Amazon EC2 キャパシティ予約をプロビジョニングするための Amazon ResourceGroups グループリソースを含めるように `ec2:RunInstances` 許可が変更されました。	
アクセス許可を AmazonECSServiceRolePolicy マネージドポリシーに追加します。	`AmazonECSServiceRolePolicy` マネージド IAM ポリシーが更新され、`ssmmessages:OpenDataChannel` アクセス許可が含まれるようになりました。この許可は、Amazon ECS が ECS Exec セッションのデータチャネルを開くことを可能にします。	2026 年 1 月 20 日

変更	説明	日付
AmazonECSInfrastructureRole PolicyFor ManagedInstances への許可を更新	AmazonECSInfrastructureRolePolicyFor ManagedInstances ポリシーが更新され、CreateFleet 許可が変更されました。以下の理由により、サブネット、セキュリティグループ、EC2 イメージに対するリソースベースの条件が削除されました。 • サブネットとセキュリティグループは Amazon ECS が作成または管理するものではないため、必須の AmazonECSManaged タグが含まれていない。 • Amazon ECS が EC2 イメージを所有および管理していても、タグがお客様に表示されないため、リソースベースのポリシーが適用されない。 この変更によって、期待される ECS 管理タグがないリソースでもポリシーが正しく機能するようになります。	2025 年 12 月 15 日
マネージドインスタンスの存続期間ポリシーを修正	マネージドインスタンスのドキュメントを更新してインスタンスの存続期間という用語を明確化し、インスタンスの最大存続期間を 14 日から 21 日に延長しました。	2025 年 12 月 11 日
アクセス許可を AmazonECSServiceRolePolicy マネージドポリシーに追加します。	AmazonECSServiceRolePolicy マネージド IAM ポリシーが更新され、ec2:DescribeInstanceEventWindows アクセス許可が含まれるようになりました。このアクセス許可により、Amazon ECS は Event Windows に関連付けられたサービスとクラスターのために Amazon EC2 Event Windows を取得することが可能になります。	2025 年 11 月 20 日

変更	説明	日付
AmazonECSServiceRolePolicy へのアクセス許可を追加します。	AmazonECSServiceRolePolicy が更新され、Amazon ECS がタスク ENI をプロビジョニングおよびプロビジョニング解除できるようにする ec2:AssociateTrunkInterface および ec2:DisassociateTrunkInterface アクセス許可が追加されました。	2025 年 11 月 14 日
the section called “AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecurity” ポリシーを更新する	the section called “AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecurity” ポリシーを更新して、Amazon ECS が Amazon ECS で作成されたシークレットのライフサイクルをモニタリングできるようにします。	2025 年 11 月 13 日
Amazon ECS カナリアデプロイ	Amazon ECS カナリアデプロイでは、大部分のトラフィックを安定したバージョンで処理しながら、トラフィックのごく一部を新しいバージョンにルーティングすることで、新しいバージョンのアプリケーションをテストできます。詳細については、「Amazon ECS カナリアデプロイ」を参照してください。	2025 年 10 月 28 日
Amazon ECS 線形デプロイ	Amazon ECS 線形デプロイでは、トラフィックを現在のサービスリビジョンから新しいリビジョンに均等な割合で徐々に移行できるため、各ステップでモニタリングしながら制御されたデプロイを行うことができます。詳細については、「Amazon ECS リニアデプロイ」を参照してください。	2025 年 10 月 28 日

変更	説明	日付
Amazon ECS マネージドインスタンスリージョン	Amazon ECS マネージドインスタンスを利用できるリージョンが追加されました。詳細については、「Architect for Amazon ECS Managed Instances」を参照してください。	2025 年 10 月 27 日
AWS for Fluent Bit バージョン 3.0.0	Fluent Bit バージョン `4.1.1` と AL2023 に基づいて、AWS for Fluent Bit バージョン `3.0.0` が利用可能になりました。詳細については、「Amazon ECS の Fluent Bit イメージリポジトリの AWS」を参照してください。	2025 年 10 月 14 日
FireLens コンテナでの非ルートユーザーのサポート	Amazon ECS が非ルートユーザーとしての FireLens コンテナの実行をサポート 詳細については、「Amazon ECS ログを AWS サービスまたは AWS Partner に送信する」を参照してください。	2025 年 10 月 13 日
コンソールでの Amazon ECS イベントキャプチャ	Amazon ECS コンソールには、サービスアクションやタスク状態の変更など、Amazon ECS で生成されたイベントを EventBridge 経由で CloudWatch Logs に保存するイベントキャプチャ機能が用意されています。詳細については、「Task lifecycle events」を参照してください。	2025 年 10 月 2 日
Amazon ECS がコンテナ化されたワークロードを実行するための Amazon ECS マネージドインスタンスをサポート	Amazon ECS は、コンテナ化されたワークロードを実行するための Amazon ECS マネージドインスタンスをサポートします。詳細については、「Architect for Amazon ECS Managed Instances」を参照してください。	2025 年 9 月 30 日
新しい the section called “`AmazonECSInstanceRolePolicyForManagedInstances`” の追加	Amazon ECS マネージドインスタンスから Amazon ECS クラスターに登録するためのアクセス許可を提供する、新しい AmazonECSInstanceRolePolicyForManagedInstances ポリシーを追加しました。	2025 年 9 月 30 日

変更	説明	日付
新しい the section called “AmazonECSInfrastructureRolePolicyForManagedInstances” の追加	Amazon EC2 マネージドリソースを作成および管理するための Amazon ECS アクセスを提供する新しい AmazonECSInfrastructureRolePolicyForManagedInstances ポリシーを追加しました。	2025 年 9 月 30 日
IPv6-only 設定での Amazon ECS ワークロードの実行をサポート	Amazon ECS ワークロードの実行が、IPv6 経由でのみタスクの通信が可能な IPv6-only 設定でできるようになりました。詳細については、「EC2 起動タイプの Amazon ECS タスクネットワークオプション」と「Fargate 起動タイプの Amazon ECS タスクのネットワークオプション」を参照してください。	2025 年 9 月 29 日
共有 AWS Cloud Map 名前空間を使用した Amazon ECS Service Connect のサポート	Amazon ECS Service Connect は、クロスアカウントのサービス間通信のために、共有 AWS Cloud Map 名前空間の使用をサポートするようになりました。詳細については、「Amazon ECS Service Connect with shared AWS Cloud Map namespaces」を参照してください。	2025 年 9 月 12 日
Amazon ECS が、Amazon Q を使用した AWS コンソールでのタスク定義編集を強化	詳細については、「Using Amazon Q Developer to provide task definition recommendations in the Amazon ECS console」を参照してください。	2025 年 9 月 10 日
コンソールからの ECS Exec コマンドの実行をサポート	ECS Exec コマンドを Amazon ECS コンソールから直接実行できるようになりました。詳細については、「Running commands using ECS Exec」に関するページを参照してください。	2025 年 9 月 4 日

変更	説明	日付
新しい the section called “AmazonECSInfrastructureRolePolicyForManagedInstances” の追加	Amazon EC2 マネージドリソースを作成および管理するための Amazon ECS アクセスを提供する新しい AmazonECSInfrastructureRolePolicyForManagedInstances ポリシーを追加しました。	2025 年 8 月 31 日
アクセス許可を AmazonEC2ContainerServiceforEC2Role マネージドポリシーに追加します。	AmazonEC2ContainerServiceforEC2Role マネージド IAM ポリシーが更新され、ecs:ListTagsForResource アクセス許可が含まれるようになりました。このアクセス許可により、Amazon ECS エージェントはタスクメタデータエンドポイント (${ECS_CONTAINER_METADATA_URI_V4}/taskWithTags) を介してタスクおよびコンテナインスタンスのタグを取得できます。詳細については、「AmazonEC2ContainerServiceforEC2Role」を参照してください。	2025 年 8 月 4 日
AmazonECSInfrastructureRolePolicyForLoadBalancers にアクセス許可を追加します。	AmazonECSInfrastructureRolePolicyForLoadBalancers マネージド IAM ポリシーが更新され、ターゲットグループを記述、登録解除、登録するための新しいアクセス許可が追加されました。	2025 年 7 月 25 日
サービスデプロイコントローラーの更新をサポート	Amazon ECS はサービスデプロイコントローラーの更新をサポートしており、既存のサービスをさまざまなデプロイタイプに変更できます。詳細については、「Amazon ECS サービスのパラメーターの更新」を参照してください。	2025 年 7 月 15 日

変更	説明	日付
Amazon ECS のネイティブなブルー/グリーンデプロイ	Amazon ECS は、ネイティブにブルー/グリーンデプロイをサポートしており、本番環境のトラフィックを新しいサービスバージョンに切り替える前に、そのバージョンを検証できます。詳細については、「Amazon ECS blue/green deployments」を参照してください。	2025 年 7 月 15 日
新しい AmazonECSInfrastructureRolePolicyForLoadBalancers マネージドポリシーを追加します。	このポリシーは、ユーザーに代わって Amazon ECS ワークロードに関連するロードバランサーを管理するために必要なその他の AWS サービスリソースへのアクセスを提供します。詳細については、「AmazonECSInfrastructureRolePolicyForLoadBalancers」を参照してください。	2025 年 7 月 15 日
SOCI Index Manifest v2 のサポート	SOCI の使用を検討している Fargate のお客様は、SOCI Index Manifest v2 のみを使用できます。Fargate で SOCI を以前使用したことのある既存のお客様は、引き続き SOCI Index Manifest v1 を使用できますが、SOCI Index Manifest v2 に移行することを強くお勧めします。詳細については、「Lazy loading container images using Seekable OCI (SOCI)」を参照してください。	2025 年 7 月 2 日
ヘルス障害によって生成されたサービスアクションイベントにタスク ID を追加しました。	詳細については、「Amazon ECS サービスの異常なイベントメッセージ」を参照してください。	2025 年 6 月 30 日

変更	説明	日付
デフォルトのログドライバーモードに更新します。	Amazon ECS のデフォルトのログ記録モードが `blocking` から `non-blocking` に更新されました。このデフォルトモードを上書きするには、コンテナ定義の `mode` に `logConfiguration` を指定するか、Amazon ECS コンソールまたは AWS CLI を使用して `defaultLogDriverMode` アカウント設定を設定します。詳細については、「Default log driver mode」を参照してください。	2025 年 6 月 25 日
AmazonECSServiceRolePolicy にアクセス許可を追加します。	`AmazonECSServiceRolePolicy` マネージド IAM ポリシーは、Amazon ECS が管理するサービスの AWS Cloud Map サービス属性を Amazon ECS が更新できる新しい AWS Cloud Map アクセス許可で更新されました。	2025 年 6 月 24 日
Amazon ECS に最適化された Windows Server 2025 Full AMI と Amazon ECS に最適化された Windows Server 2025 Core AMI のサポート。	詳細については、「Amazon ECS 最適化 Windows AMI」を参照してください。	2025 年 6 月 20 日
サービスプロバイダー戦略の更新のサポートです。	サービスプロバイダー戦略を更新できるようになりました。詳細については、「Update service parameters」を参照してください。	2025 年 6 月 12 日
AmazonECSInfrastructureRolePolicyForVolumes へのアクセス許可を追加する	`AmazonECSInfrastructureRolePolicyForVolumes` ポリシーが更新され、`ec2:DescribeInstances` のアクセス許可が追加されました。アクセス許可は、同じコンテナインスタンスで実行される Amazon ECS タスクにアタッチされている Amazon EBS ボリュームのデバイス名のコリジョンを防ぐのに役立ちます。	2025 年 6 月 2 日

変更	説明	日付
Amazon ECS タスクにアタッチされた Amazon EBS ボリュームのボリューム初期化レートのサポート。	`volumeIntiializationRate` を設定することにより既存の Amazon EBS ボリュームのスナップショットからデータを取得するレートを MiB/秒単位で指定して、Amazon ECS タスクにアタッチする新しいボリュームを作成できるようになりました。詳細については、「volumeInitializationRate (ボリューム初期化レート)」を参照してください。	2025 年 5 月 13 日
サービスデプロイの停止のサポート。	サービスのデプロイを停止できるようになりました。詳細については、「Amazon ECS サービスのデプロイの停止」を参照してください。	2025 年 5 月 2 日
Amazon ECS デュアルスタックエンドポイントのサポート。	デュアルスタックエンドポイントを使用して Amazon ECS とやり取りできるようになりました。デュアルスタックエンドポイントは、インターネットプロトコルバージョン 4 (IPv4) とインターネットプロトコルバージョン 6 (IPv6) の両方を介した Amazon ECS へのリクエストをサポートします。詳細については、「Amazon ECS デュアルスタックエンドポイントの使用」を参照してください。	2025 年 4 月 30 日
AWS リージョン のアカウントレベルでのデフォルトのログドライバーモードの設定をサポート。	`defaultLogDriverMode` アカウント設定を使用して、コンテナから `logConfiguration` で指定したログドライバーへのログメッセージのデフォルトの配信モードを設定できるようになりました。詳細については、「Default log driver mode」を参照してください。	2025 年 4 月 17 日
短い ARN 形式から長い ARN 形式へのサービス移行をサポート。	短い ARN 形式のサービスを長い ARN 形式のパケットに移行できるようになりました。詳細については、「Migrate an Amazon ECS short service ARN to a long ARN」を参照してください。	2025 年 2 月 13 日

変更	説明	日付
Amazon ECS と Fargate での 3 種類のフォールトインジェクションをサポート。	Amazon ECS と Fargate で、ネットワークブラックホールポート、ネットワークレイテンシー、ネットワークパケット損失フォールトインジェクションが利用可能になりました。詳細については、「Use AWS Fault Injection Service to test your Amazon ECS workloads」を参照してください。	2024 年 12 月 19 日
オブザーバビリティが強化された CloudWatch Container Insights のサポート。	オブザーバビリティが強化された CloudWatch Container Insights を使用できます。詳細については、「Container Insights を使用して Amazon ECS コンテナをモニタリングする」を参照してください。	2024 年 12 月 2 日
予測スケーリングのサポート。	予測スケーリングを使用すると、将来のニーズを予測し、必要に応じてサービスのタスクをプロアクティブに増やすことができます。詳細については、「履歴パターンを使用して予測スケーリングで CloudWatch サービスをスケールする」を参照してください。	2024 年 11 月 21 日
サービスの再調整のサポート。	Amazon ECS で、サービスタスクのバランスをアベイラビリティーゾーン間で自動調整できるようになりました。詳細については、「アベイラビリティーゾーン間での Amazon ECS サービスの調整」を参照してください。	2024 年 11 月 19 日
コンテナイメージタグをイメージダイジェストに解決する設定のサポート。	サービスの一部であるタスクで、コンテナごとにコンテナイメージタグをダイジェストに解決する設定が可能になりました。詳細については、「コンテナイメージの解決」および「versionConsistency」を参照してください。	2024 年 11 月 19 日
新しい AmazonECSInfrastructureRolePolicyForVpcLattice IAM ポリシーの追加	AmazonECSInfrastructureRolePolicyForVpcLattice ポリシーは、ユーザーに代わって Amazon ECS ワークロードの VPC Lattice 機能を管理するために必要な他の AWS サービスリソースへのアクセスを提供します。	2024 年 11 月 18 日

変更	説明	日付
VPC Lattice が Amazon ECS サービスで利用できるようになりました。	VPC Lattice を使用して、サービス間の接続、セキュリティ、およびオブザーバビリティを標準化できるようになりました。詳細については、「Amazon VPC Lattice を使用して Amazon ECS サービスを接続、監視、保護する」を参照してください。	2024 年 11 月 18 日
サービスデプロイとサービスリビジョンを使用するサービス履歴のサポート。	サービスデプロイとサービスリビジョンを使用してデプロイ履歴を表示できます。詳細については、「Amazon ECS サービスデプロイを使用してサービス履歴を表示する」と「Amazon ECS サービスリビジョン」を参照してください。	2024 年 10 月 30 日
Windows オペレーティングシステムを使用する EC2 起動タイプの Amazon EBS ボリュームのサポート。	Amazon ECS は、Windows の EC2 インスタンスで実行されているタスクにアタッチされた Amazon EBS ボリュームをサポートします。詳細については、「Amazon ECS での Amazon EBS ボリュームの使用」を参照してください。	2024 年 10 月 24 日
AmazonECSInfrastructureRolePolicyForVolumes へのアクセス許可を追加する。	`AmazonECSInfrastructureRolePolicyForVolumes` ポリシーが更新され、お客様がスナップショットから Amazon EBS ボリュームを作成できるようになりました。詳細については、「AmazonECSInfrastructureRolePolicyForVolumes」を参照してください。	2024 年 10 月 10 日
Fargate Spot が Linux ARM で利用可能。	Linux ARM での Fargate Spot のサポートが Amazon ECS に追加されました。	2024 年 9 月 6 日
Amazon ECS タスク内の個々のコンテナのコンテナ再起動ポリシー。	タスク定義で定義されている重要なコンテナと重要でないコンテナの再起動ポリシーを有効にして、一時的な障害を迅速に克服し、タスクの可用性を維持できます。詳細については、「コンテナ再起動ポリシーを使用して Amazon ECS タスク内の個々のコンテナを再起動する」を参照してください。	2024 年 8 月 15 日

変更	説明	日付
アクセス許可を the section called “AmazonECS_FullAccess” に追加しました。	AmazonECS_FullAccess ポリシーが更新され、ecsInfrastructureRole という名前のロールの IAM ロールの iam:PassRole アクセス許可が追加されました。これは、AWS マネジメントコンソール によって作成されたデフォルトの IAM ロールで、Amazon ECS が ECS タスクにアタッチされた Amazon EBS ボリュームを管理できる ECS インフラストラクチャロールとして使用されることを目的としています。	2024 年 8 月 13 日
Amazon Linux 2023、CentOS Stream 9、Fedora 40、Ubuntu 24 のサポートが ECS Anywhere に追加されました	Amazon Linux 2023、CentOS Stream 9、Fedora 40、Ubuntu 24 オペレーティングシステムのサポートが ECS Anywhere に追加されました。詳細については、「サポートされるオペレーティングシステムとシステムアーキテクチャ」を参照してください。	2024 年 7 月 23 日
ローリングアップデート (ECS) デプロイタイプを使用するサービスのコンテナイメージ解決。	ローリングアップデートデプロイコントローラーを使用するサービス内のすべてのタスクで同じコンテナイメージが使用されるように、Amazon ECS はタスク定義で指定されたコンテナイメージ名とイメージタグをコンテナイメージダイジェストに解決します。詳細については、「コンテナイメージの解決」を参照してください。	2024 年 7 月 10 日
Debian 11 と Debian 12 のサポートが ECS Anywhere に追加されました	Debian 11 と Debian 12 オペレーティングシステムのサポートが ECS Anywhere に追加されました。詳細については、「サポートされるオペレーティングシステムとシステムアーキテクチャ」を参照してください。	2024 年 3 月 28 日

変更	説明	日付
Fargate サポートの Linux コンテナ用 gMSA	Amazon ECS は、グループマネージドサービスアカウント (gMSA) と呼ばれる特殊なサービスアカウントを使用して、Fargate 上の Linux コンテナの Active Directory 認証をサポートします。詳細については、「Fargate の Linux コンテナに gMSA を使用する」を参照してください。	2024 年 3 月 5 日
タスクにアタッチされた Amazon EBS ボリュームに CloudWatch メトリクスが追加されました	Amazon ECS は、Amazon EBS ボリュームがアタッチされているタスクの、Amazon EBS ストレージの使用状況に関する CloudWatch メトリクスを発行できるようになりました。詳細については、「Amazon ECS CloudWatch メトリクス」を参照してください。	2024 年 2 月 8 日
Service Connect TLS	Service Connect で TLS を使用できるようになりました。	2024 年 1 月 22 日
Service Connect TLS マネージドポリシー	新しく、AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecurity ポリシーを追加しました。	2024 年 1 月 22 日
Service Connect タイムアウト設定の更新	Service Connect のタイムアウト設定を更新できるようになりました。これには、2 つのオプションパラメータ `idleTimeout` 、`perRequestTimeout` が含まれます。	2024 年 1 月 22 日
Amazon ECS マネージドインスタンスドレイニング	Amazon ECS マネージドインスタンスドレイニングを使用すると、Amazon ECS インスタンスの正常な終了が容易になります。	2024 年 1 月 19 日
ECS Anywhere に Ubuntu 22 サポートが追加されました	Ubuntu 22 オペレーティングシステムのサポートが ECS Anywhere に追加されました。詳細については、「サポートされるオペレーティングシステムとシステムアーキテクチャ」を参照してください。	2024 年 1 月 16 日

変更	説明	日付
AmazonECSInfrastructureRolePolicyForVolumes IAM ポリシーを追加する	AmazonECSInfrastructureolePolicyForVolumes が追加されました。このポリシーにより、Amazon ECS が AWS API コールを行い、Amazon ECS ワークロードに関連付けられた Amazon EBS ボリュームを管理するために必要な権限が付与されます。	2024 年 1 月 11 日
Amazon ECS タスク用の Amazon EBS データボリューム	スタンドアロンの Amazon ECS タスク、またはタスクのデプロイ時に ECS サービスによって管理されるタスクにアタッチするために、タスクごとに 1 つの Amazon EBS データボリュームを設定できます。デプロイ時にボリュームを設定すると、特定のボリュームタイプや設定に限定されず、再利用可能なタスク定義を作成できます。Amazon EBS ボリュームは、データ集約型のコンテナ化されたワークロード向けに、可用性に優れ、費用対効果が高く、耐久性があり、高性能なブロックストレージを提供します。	2024 年 1 月 11 日
Amazon ECS 従来型コンソールのサポートが終了しました	Amazon ECS コンソールのサポートが終了しました	2023 年 12 月 4 日
ポリシーの更新	AmazonECSServiceRolePolicy マネージド IAM ポリシーでは、新しい events アクセス許可が更新され、autoscaling 、および autoscaling-plans アクセス許可が追加、更新されました。	2023 年 12 月 4 日
Runtime Monitoring サポート	Runtime Monitoring を使用すると、Amazon ECS ワークロードを監視して、悪意のある動作や不正な動作を識別できます。詳細については、「Runtime Monitoring」を参照してください。	2023 年 11 月 26 日
ポリシーの更新	AmazonECSServiceRolePolicy マネージド IAM ポリシーが AWS Cloud Map DiscoverInstancesRevision API へのアクセスを許可するように更新されました。	2023 年 10 月 4 日

変更	説明	日付
AWS Fargate タスクの廃止設定	Fargate タスクが廃止されるまでの待機時間を設定できます。詳細については、「AWS Fargate タスクのメンテナンス」を参照してください。	2023 年 9 月 5 日
AWS Fargate での追加のタスク定義パラメータ	AWS Fargate は、Linux プラットフォームバージョン 1.4.0 で pidMode および systemControls のサポートを追加します。詳細については、「タスク定義」を参照してください。	2023 年 8 月 9 日
Amazon ECS コンソールのタスク定義ページの再設計	Amazon ECS コンソールのタスク定義ページが再設計され、オプションが追加されました。詳細については、「コンソールを使用したタスク定義の作成」を参照してください。	2023 年 7 月 26 日
Fargate は Seekable OCI インデックスによる遅延読み込みをサポートしています	AWS Fargate は Seekable OCI (SOCI) インデックスを導入しています。SOCI では、コンテナは起動前にイメージプルに数秒しかかからないため、イメージがバックグラウンドでダウンロードされている間、環境のセットアップとアプリケーションのインスタンス化に時間を割けます。詳細については、「Lazy loading container images using Seekable OCI (SOCI)」を参照してください。	2023 年 7 月 17 日
Linux と Windows での gMSA のサポートが改善されました	タスク定義には、Linux および Windows 用の gMSA 用の新しい credentialSpecs フィールドがあります。Windows 上のドメインレス gMSA の新しい完全なチュートリアルが追加されました。「チュートリアル: AWS CLI を使用したドメインレス gMSA での Windows コンテナの使用」を参照してください。詳細については、「Linux コンテナ向け gMSAs を使用する」と「Windows コンテナ向けの gMSAs の使用」を参照してください。	2023 年 7 月 14 日

変更	説明	日付
ECS エージェントバージョンのドキュメントが改善されました	Amazon ECS エージェントバージョンのドキュメントが更新されました。Amazon ECS コンテナエージェントの最新バージョンでは、Docker の `v20.10.13` バージョン以降を使用することをお勧めします。リリースされたバージョンとエージェントの変更は GitHub で確認できます。詳細については、「Amazon ECS Linux コンテナエージェントのバージョン」を参照してください。	2023 年 6 月 20 日
Fargate ARM64 サポートの利用可能なリージョンを更新	Fargate ARM64 サポートの利用可能なリージョンを更新しました。詳細については、「考慮事項」を参照してください。	2023 年 6 月 19 日
クラスターの自動スケーリングに関するドキュメントの改善	Amazon EC2 Auto Scaling の Amazon ECS スケーリングに関するドキュメントでは、精度と読みやすさが大幅に向上しています。詳細については、「Amazon ECS のクラスター自動スケーリング」を参照してください。	2023 年 5 月 4 日
リソース作成のタグ付け認可。	ユーザーには、リソースを作成するアクションの許可が必要です (`ecsCreateCluster` など)。リソースを作成し、そのリソースのタグを指定すると、AWS は、追加の認可を実行して、タグを作成するための許可があることを検証します。詳細については、「タグ付け権限」と「作成時にリソースにタグを付ける権限の付与」を参照してください。	2023 年 4 月 18 日
EC2 での Linux コンテナの gMSA のサポート	gMSA を使用して、EC2 での Linux コンテナ用に、Active Directory に対して認証できます。詳細については、「Linux コンテナに gMSA を使用する」を参照してください。	2023 年 4 月 14 日
AWS Fargate での Windows コンテナのエフェメラルストレージのサポート	AWS Fargate での Windows コンテナにエフェメラルストレージを使用できます。詳細については、「Fargate タスクストレージ」を参照してください。	2023 年 4 月 14 日

変更	説明	日付
AWS Cost Managemen tタスクレベルの CUR データの サポート	コストと使用状況レポートで、タスクレベルのコストとリソースの使用状況をオンにできます。これにより、AWS Fargate と EC2 で実行されるタスクについてのコスト配分データの分割が追加されます。詳細については、「タスクレベルのコストと使用状況レポート」を参照してください。	2023 年 4 月 12 日
Amazon Linux 2023 Amazon ECS 最適化 AMI	ワークロードを Amazon Linux 2023 Amazon ECS 最適化 AMI にデプロイできます。詳細については、「Amazon ECS に最適化された Linux AMI」を参照してください。	2023 年 4 月 10 日
AWS Fargate 連邦情報処理標準 (FIPS) 140	連邦情報処理標準 (FIPS) 140 に準拠した態様で、AWS Fargate での Amazon ECS でワークロードをデプロイできます。詳細については、「AWS Fargate 連邦情報処理標準 (FIPS-140)」を参照してください。	2023 年 4 月 10 日
タスク定義の削除	タスク定義は、Amazon ECS コンソール、SDK、および AWS CLI を使用して削除できます。詳細については、「コンソールを使用したタスク定義リビジョンの削除」および「タスク定義」を参照してください。	2023 年 2 月 24 日
AWS FargateCompute Optimizer 内の サービスについての推奨事項	AWS Compute Optimizer は、AWS Fargate の Amazon ECS サービスで実行中のタスクの使用率に基づいて、タスクとコンテナサイズに関する推奨事項を生成します。詳細については、「Fargate の Amazon ECS サービスに関する推奨事項の表示」を参照してください。	2023 年 1 月 27 日
Amazon ECS コンソール	新しい Amazon ECS コンソールが、デフォルトのコンソールになりました。詳細については、「新しい Amazon Elastic Container Service コンソール」を参照してください。	2023 年 1 月 19 日

変更	説明	日付
AmazonECS_FullAccess IAM ポリシーが更新されました	AmazonECS_FullAccess IAM ポリシーが更新され、作成時にロードバランサーにタグを追加する権限が含まれるようになりました。詳細については、「AmazonECS_FullAccess」を参照してください。	2023 年 1 月 4 日
CloudWatch アラームを使用して Amazon ECS サービスのデプロイ障害を検出する	指定された CloudWatch アラームが ALARM 状態になったことを検出したとき、デプロイが失敗に設定されるように Amazon ECS を設定できます。詳細については、「the section called “ローリング更新デプロイ”」を参照してください。	2022 年 12 月 19 日
コンテナポートマッピングのサポート	動的にマッピングされたホストポート範囲にバインドされるコンテナポートの番号範囲を設定できます。詳細については、「the section called “ポートマッピング”」を参照してください。	2022 年 12 月 15 日
Amazon ECS Service Connect の一般提供	この機能により、Amazon ECS サービスデプロイによって制御されるサービス検出とサービスメッシュが追加されます。詳細については、「the section called “Service Connect”」を参照してください。	2022 年 11 月 27 日
新しい Amazon ECS コンソールエクスペリエンスでは、タスク定義の操作が更新されました。	新しい Amazon ECS コンソールエクスペリエンスでは、タスク定義に JSON エディタが使用できるようになりました。詳細については、「the section called “コンソールを使用したタスク定義の作成”」を参照してください。	2022 年 10 月 27 日
新しい Amazon ECS コンソールエクスペリエンスでは、タスク定義の操作が更新されました。	新しい Amazon ECS コンソールエクスペリエンスでは、タスク定義に JSON エディタが使用できるようになりました。詳細については、「the section called “コンソールを使用したタスク定義の作成”」を参照してください。	2022 年 10 月 27 日

変更	説明	日付
新しい Amazon ECS コンソールのエクスペリエンスが更新されました	新しい Amazon ECS コンソールエクスペリエンスでは、サービスとタスクのパラメータが追加され、更新されています。詳細については、「the section called “ローリング更新デプロイの作成”」および「the section called “タスクとしてのアプリケーションの実行”」を参照してください。	2022 年 10 月 7 日
タスクメタデータエンドポイントバージョン 4 の新しい情報	タスクメタデータのエンドポイントバージョン 4 には、VPC ID とサービス名が含まれるようになりました。詳細については、「the section called “タスクメタデータエンドポイントバージョン 4”」を参照してください。	2022 年 10 月 7 日
新しいタスク定義サイズ	Fargate の Amazon ECS は 8 個の vCPU と 16 個の vCPU タスクサイズをサポートするようになりました。詳細については、「the section called “タスクサイズ”」を参照してください。	2022 年 9 月 16 日
ECS CLI ページがアーカイブされました	ECS CLI のドキュメントがアーカイブされました。コマンドラインツールのニーズに合わせて AWS Copilot を使用することをお勧めします。詳細については、「AWS Copilot コマンドラインインターフェイスを使用した Amazon ECS リソースの作成」を参照してください。	2022 年 9 月 15 日
Fargate クォータ	Fargate は、タスク数ベースのクォータから vCPU ベースのクォータに移行しています。詳細については「Amazon ECS のエンドポイントとクォータ」を参照してください。	2022 年 9 月 8 日
Amazon EC2 Auto Scaling でのウォームプールのサポート	Amazon EC2 Auto Scaling のウォームプールを使用して、アプリケーションのスケールアウトを高速化し、コストを削減できるようになりました。詳細については、「Amazon ECS Auto Scaling グループ用に事前初期化されたインスタンスを設定する」を参照してください。	2022 年 3 月 23 日

変更	説明	日付
ECS Anywhere での Windows インスタンスのサポート。	ECS Anywhere は Windows インスタンスをサポートするようになりました。詳細については、「外部インスタンスの Amazon ECS クラスター」を参照してください。	2022 年 3 月 3 日
外部インスタンスに対する ECS Exec サポートを追加	外部インスタンスに対して ECS Exec が対応するようになりました。詳細については、「ECS Exec を使用して Amazon ECS コンテナをモニタリングする」を参照してください。	2022 年 1 月 24 日
新しい Amazon ECS コンソールのエクスペリエンスが更新されました	新しい Amazon ECS コンソールエクスペリエンスでは、クラスターの作成と削除、タスク定義の更新、およびタスク定義の登録解除がサポートされています。詳細については、「Fargate ワークロード用の Amazon ECS クラスターを作成する」、「Amazon ECS クラスターを削除する」、「コンソールを使用した Amazon ECS タスク定義の更新」、および「新しいコンソールを使用した Amazon ECS タスク定義リビジョンの登録解除」を参照してください。	2021 年 12 月 8 日
新しい Amazon ECS コンソールのエクスペリエンスが更新されました	新しい Amazon ECS コンソールエクスペリエンスでは、タスク定義の作成がサポートされています。詳細については、「コンソールを使用した Amazon ECS タスク定義の作成」を参照してください。	2021 年 11 月 23 日
Amazon ECS は、Linux 用の 64 ビット ARM アーキテクチャをサポートしています。	Amazon ECS は、Linux オペレーティングシステム 64 ビット ARM CPU アーキテクチャをサポートしています。詳細については、「the section called “64 ビット ARM ワークロードでのタスク定義”」を参照してください。	2021 年 11 月 23 日
fluentd log-driver-buffer-limit オプションに対する Amazon ECS のサポート	Amazon ECS は、fluentd `log-driver-buffer-limit` オプションをサポートしています。詳細については、「Amazon ECS ログを AWS サービスまたは AWS Partner に送信する」を参照してください。	2021 年 11 月 22 日

変更	説明	日付
Amazon ECS 最適化 Linux AMI のビルドスクリプト	Amazon ECS は、Amazon ECS に最適化された AMI の Linux バリアントの構築に使用するビルドスクリプトをオープンソース化しました。詳細については、「Amazon ECS 最適化 Linux AMI のビルドスクリプト」を参照してください。	2021 年 11 月 19 日
コンテナインスタンスのヘルス	Amazon ECS は、コンテナインスタンスのヘルスモニタリングのサポートを追加します。詳細については、「Amazon ECS コンテナインスタンスの正常性をモニタリングする」を参照してください。	2021 年 11 月 10 日
Windows Amazon ECS Exec のサポート	Amazon ECS Exec は、Windows をサポートしています。詳細については、「ECS Exec を使用して Amazon ECS コンテナをモニタリングする」を参照してください。	2021 年 11 月 1 日
Fargate での Windows コンテナのサポート。	Amazon ECS は、Fargate で Windows コンテナをサポートします。詳細については、「Fargate Windows プラットフォームバージョンの変更ログ」を参照してください。	2021 年 10 月 28 日
Amazon ECS Anywhere での外部インスタンスの GPU サポート	Amazon ECS は、外部インスタンスで実行するタスクのタスク定義で GPU 要件の指定をサポートしています。詳細については、「GPU ワークロード向けの Amazon ECS タスク定義」および「Amazon ECS クラスターに外部インスタンスを登録する」を参照してください。	2021 年 10 月 8 日
Windows のawsvpcネットワークモードのサポート	Amazon ECS が Windows で awsvpc ネットワークモードをサポートします。詳細については、「Amazon ECS タスクにネットワークインターフェイスを割り当てる」を参照してください。	2021 年 7 月 15 日

変更	説明	日付
Bottlerocket の一般提供	Amazon ECS は、Bottlerocket オペレーティングシステムの Amazon ECS に最適化された AMI バリアントを AMI として提供します。詳細については、「Amazon ECS 最適化 Bottlerocket AMI」を参照してください。	2021 年 6 月 30 日
Amazon ECS のスケジュールされたタスクの更新	Amazon EventBridge は、Amazon ECS のスケジュールされたタスクをトリガーするルールを作成するときに、追加のパラメータのサポートを追加しました。	2021 年 6 月 25 日
AWSAmazon ECS の 管理ポリシー	Amazon ECS は、サービスにリンクされたロールの AWS 管理ポリシーのドキュメントを追加しました。詳細については、「AWSAmazon Elastic Container Service に関する 管理ポリシー」を参照してください。	2021 年 6 月 8 日
AWS CDK の開始方法	Amazon ECS でAWS CDKを使用するための入門ガイドを追加しました 詳細については、「AWS CDK を使用した Amazon ECS リソースの作成」を参照してください。	2021 年 5 月 27 日
Amazon ECS Anywhere	Amazon ECS は、オンプレミスのサーバーまたは仮想マシン (VM) をクラスターに登録するためのサポートを追加しました。詳細については、「外部インスタンスの Amazon ECS クラスター」を参照してください。	2021 年 5 月 25 日
Amazon ECS に最適化された Windows Server 20H2 Core AMI	Amazon ECS では、Windows サーバー 20H2 コア用の新しい Windows Amazon ECS 最適化 AMI バリアントのサポートが追加されました。詳細については、「Amazon ECS に最適化された Linux AMI」を参照してください。	2021 年 4 月 19 日

変更	説明	日付
Amazon ECS Exec	Amazon ECS は、ECS Exec という新しいデバッグツールをリリースしました。詳細については、「ECS Exec を使用して Amazon ECS コンテナをモニタリングする」を参照してください。	2021 年 3 月 15 日
VPC エンドポイントポリシーのサポート	Amazon ECS は Amazon VPC エンドポイントポリシーをサポートするようになりました。詳細については、「Amazon ECS 用の VPC エンドポイントポリシーの作成」を参照してください。	2021 年 1 月 11 日
新しいコンソールエクスペリエンス	Amazon ECSは、サービスの作成もしくは更新、またはスタンドアロンタスクの実行をサポートする新しいコンソールエクスペリエンスをリリースしました。詳細については、「Amazon ECS のローリング更新デプロイの作成」および「Amazon ECS タスクとしてのアプリケーションの実行」を参照してください。	2020 年 12 月 28 日
キャパシティープロバイダーの更新	Amazon ECS は既存の Auto Scaling グループキャパシティープロバイダーの更新をサポートするようになりました。	2020 年 11 月 23 日
ECS は、Windows タスク用の Amazon FSx for Windows File Server をサポートするようになりました。	Amazon ECS は Windows タスク定義で Amazon FSx for Windows File Server ボリュームを指定するサポートを追加しました。詳細については、「Amazon ECS での FSx for Windows File Server ボリュームの使用」を参照してください。	2020 年 11 月 11 日
VPC デュアルスタックモードのサポートを追加	Amazon ECS は、IPv6 アドレスのサポートを提供する `awsvpc` ネットワークモードを使用するタスクで VPC をデュアルスタックモードで使用するためのサポートを追加しました。詳細については、「デュアルスタックモードでの VPC の使用」を参照してください。	2020 年 11 月 5 日

変更	説明	日付
タスクメタデータエンドポイント v4 更新	Amazon ECS はタスクメタデータエンドポイント v4 出力にメタデータを追加しました。詳細については、「Amazon ECS タスクメタデータエンドポイントバージョン 4」を参照してください。	2020 年 11 月 5 日
Local Zones と Wavelength Zone のサポート	Amazon ECS では、Local Zones と Wavelength Zone のワークロードのサポートが追加されました。詳細については、「共有サブネット、Local Zones、および Wavelength Zones の Amazon ECS アプリケーション」を参照してください。	2020 年 9 月 4 日
Bottlerocket AMI の Amazon ECS バリアント	Bottlerocket は Linux ベースのオープンソースのオペレーティングシステムで、コンテナを実行するために AWS によって専用に構築されています。Amazon ECS コンテナインスタンスを起動するときに利用できる AMI として、Bottlerocket オペレーティングシステムの Amazon ECS 最適化された AMI バリアントが用意されています。詳細については、「Amazon ECS 最適化 Bottlerocket AMI」を参照してください。	2020 年 8 月 31 日
ネットワークレートの統計情報用に更新されたタスクメタデータエンドポイントバージョン 4	タスクメタデータエンドポイントバージョン 4 が更新され、コンテナエージェントのバージョン `1.43.0` 以上を実行している Amazon EC2 インスタンスでホストされる `awsvpc` または `bridge` ネットワークモードを使用する Amazon ECS タスクのネットワークレート統計が提供されました。詳細については、「Amazon ECS タスクメタデータエンドポイントバージョン 4」を参照してください。	2020 年 8 月 10 日
Fargate 使用状況メトリクス	AWS Fargate は、Fargate·と·Fargate·Spot リソースのアカウント使用状況を可視化する CloudWatch 使用状況メトリクスを提供します。詳細については、「使用状況メトリクス」を参照してください。	2020 年 8 月 3 日

変更	説明	日付
AWS Copilot バージョン 0.1.0	ローカル開発環境からAmazon ECS のコンテナ化されたアプリケーションのモデル化、作成、リリース、管理を簡素化するハイレベルのコマンドを提供する新しいAWSコパイロット CLI が公開されました。詳細については、「AWS Copilot コマンドラインインターフェイスを使用した Amazon ECS リソースの作成」を参照してください。	2020 年 7 月 9 日
AWS Fargate プラットフォームバージョンの廃止スケジュール	Fargateプラットフォームバージョンの廃止スケジュールが追加されました。詳細については、「AWS Fargate Linux プラットフォームバージョンの廃止」を参照してください。	2020 年 7 月 8 日
AWS Fargateリージョンの拡大	Amazon ECS onAWS Fargate が欧州 (ミラノ) リージョンに拡大しました。	2020 年 6 月 25 日
Amazon ECS 最適化 Amazon Linux 2 (Neuron) AMI がリリースされました	Amazon ECS は、Inferentia ワークロード用に Amazon ECS 最適化 Amazon Linux 2 (Neuron) AMI をリリースしました。 詳細については、「Amazon ECS に最適化された Linux AMI」を参照してください。	2020 年 6 月 24 日
キャパシティープロバイダーの削除に関するサポートを追加	Auto Scaling グループキャパシティープロバイダーの削除に関するサポートを Amazon ECS に追加しました。	2020 年 6 月 11 日
AWS Fargateプラットフォームバージョン 1.4.0 の更新	2020 年 5 月 28 日以降、プラットフォームバージョン 1.4.0 を使用して起動されるすべての新しいFargateタスクには、AWS Fargateマネージド 暗号化キーを使用し AES-256 暗号化アルゴリズムで暗号化された 20 GB のエフェメラルストレージが搭載されます。詳細については、「Amazon ECS 向けの Fargate タスクエフェメラルストレージ」を参照してください。	2020 年 5 月 28 日

変更	説明	日付
環境変数ファイルのサポート	タスク定義で環境変数ファイルを指定するためのサポートが追加されました。これにより、環境変数をコンテナに一括追加できます。詳細については、「個々の環境変数を Amazon ECS コンテナに渡す」を参照してください。	2020 年 5 月 18 日
AWS Fargateリージョンの拡大	AWS Fargate with Amazon ECS がアフリカ (ケープタウン) リージョンに拡大しました。	2020 年 5 月 11 日
サービスクォータが更新されました	次のサービスクォータが更新されました。 • アカウントあたりのクラスターが 2,000 から 10,000 に引き上げられました。 詳細については「Amazon ECS のエンドポイントとクォータ」を参照してください。	2020 年 4 月 17 日

変更	説明	日付
AWS Fargate プラットフォームバージョン 1.4.0	AWS Fargate プラットフォームバージョン 1.4.0 がリリースされました。このバージョンには、次の機能が含まれます: • 永続的なタスクストレージとしてAmazon EFS ファイルシステムボリュームを使用するサポートを追加しました。詳細については、「Amazon ECS での Amazon EFS ボリュームの使用」を参照してください。 • エフェメラルタスクストレージを 20 GB に増加しました。詳細については、「Amazon ECS 向けの Fargate タスクエフェメラルストレージ」を参照してください。 • タスクとの間のネットワークトラフィック動作を更新しました。プラットフォームバージョン 1.4 以降、すべての Fargate タスクは単一の Elastic Network Interface (タスク ENI と呼ばれる) を受け取り、すべてのネットワークトラフィックは VPC 内でこの ENI を通過し、VPC フローログを通じて表示されます。詳細については、「Fargate 起動タイプの Amazon ECS タスクのネットワークオプション」を参照してください。 • タスク ENI は、ジャンボフレームのサポートを追加しています。ネットワークインターフェイスは、最大転送単位 (MTU) で設定されます。MTU は、1 つのフレームに収まるペイロードの最大サイズです。MTU が大きいほど、1 つのフレーム内に収まるアプリケーションのペイロードが増えるため、フレームあたりのオーバーヘッドが減少し、効率が向上します。ジャンボフレームをサポートすると、オーバーヘッドが減ります。タスクと転送先との	2020 年 4 月 8 日

変更	説明	日付
	ネットワークパスでジャンボフレームをサポートすると、VPC 内に残っているすべてのトラフィックなどのオーバーヘッドが軽減されます。 • CloudWatch Container Insights には、Fargate タスクのネットワークパフォーマンスメトリクスが含まれます。詳細については、「オブザーバビリティが強化された Container Insights を使用し、Amazon ECS コンテナを監視する」を参照してください。 • タスクメタデータエンドポイント v4 のサポートを追加しました。これにより、タスクのネットワーク統計情報や、タスクが実行されているアベイラビリティーゾーンなど、Fargate タスクに関する追加情報が提供されます。詳細については、 を参照してください。Amazon ECS タスクメタデータエンドポイントバージョン 4 • コンテナの定義に SYS_PTRACE Linux パラメータのサポートを追加しました。詳細については、「Linux パラメータ」を参照してください。 • Fargateコンテナエージェントは、Amazon ECS コンテナエージェントの使用をすべての Fargate タスクに置き換えます。この変更は、タスクの実行方法には影響しません。 • コンテナランタイムは Docker の代わりに Containerd を使用するようになりました。この変更は、タスクの実行方法には影響しません。コンテナランタイムで発生するいくつかのエラーメッセージは、より一般的な内容になり、Docker には言及されなくなります。	

変更	説明	日付
	詳細については、「Amazon ECS 向け Fargate プラットフォームバージョン」を参照してください。	
タスクボリュームに対するAmazon EFS ファイルシステムのサポート	Amazon EFS ファイルシステムは、Amazon ECS タスクと Fargate タスクの両方でデータボリュームとして使用できます。詳細については、「Amazon ECS での Amazon EFS ボリュームの使用」を参照してください。	2020 年 4 月 8 日
Amazon ECS タスクメタデータエンドポイントバージョン 4	Amazon ECS コンテナエージェントバージョン 1.39.0 および Fargate プラットフォームバージョン 1.4.0 以降、`ECS_CONTAINER_METADATA_URI_V4` という環境変数がタスク内の各コンテナに挿入されます。タスクメタデータバージョン 4 エンドポイントに対してクエリを実行すると、さまざまなタスクメタデータおよび Docker 統計情報がタスクで利用可能になります。詳細については、「Amazon ECS タスクメタデータエンドポイントバージョン 4」を参照してください。	2020 年 4 月 8 日
環境変数として挿入される特定のバージョンの Secrets Manager シークレットのサポート	特定のバージョンの Secrets Manager のシークレットを使用して機密データを指定するためのサポートが追加されました。詳細については、「Amazon ECS コンテナに機密データを渡す」を参照してください。	2020 年 2 月 24 日
ブルー/グリーンデプロイ用の CodeDeploy デプロイ設定オプションをさらに追加しました	CodeDeploy サービスの Amazon ECS デプロイタイプとして新しい Canary および線形デプロイ設定を追加しました。カスタムのデプロイ設定を定義する機能も使用できます。詳細については、「Amazon ECS 用の CodeDeploy ブルー/グリーンデプロイ」を参照してください。	2020 年 2 月 6 日

変更	説明	日付
efsVolumeConfiguration タスク定義パラメータを追加	`efsVolumeConfiguration` タスク定義パラメータはパブリックプレビューであるため、Amazon ECS タスクで Amazon EFS ファイルシステムを簡単に使用できます。詳細については、「Amazon ECS での Amazon EFS ボリュームの使用」を参照してください。	2020 年 1 月 17 日
Amazon ECS コンテナエージェントのログ記録動作を更新	Amazon ECS コンテナエージェントのログの場所とローテーション動作が更新されました。詳細については、「Amazon ECS コンテナエージェントのログ設定パラメータ」を参照してください。	2020 年 1 月 13 日
Fargate Spot	Amazon ECSでは、Fargate Spot を使用したタスクの実行のサポートが追加されました。詳細については、「Fargate 用の Amazon ECS クラスター」を参照してください。	2019 年 12 月 3 日
クラスターの自動スケーリング	Amazon ECS クラスターの自動スケーリング を使用すると、クラスター内のタスクをスケールする方法をより詳細に制御できます。詳細については、「クラスターの自動スケーリングで Amazon ECS キャパシティーを自動的に管理する」を参照してください。	2019 年 12 月 3 日
クラスターキャパシティープロバイダー	Amazon ECS クラスターキャパシティープロバイダーは、タスクに使用するインフラストラクチャを決定します。詳細については、「Amazon ECS クラスター」を参照してください。	2019 年 12 月 3 日
AWS Outposts でのクラスターの作成	Amazon ECS では、AWS Outposts でのクラスターの作成がサポートされるようになりました。詳細については、「the section called “AWS OutpostsのAmazon Elastic Container Service”」を参照してください。	2019 年 12 月 3 日

変更	説明	日付
サービスアクションイベント	Amazon ECS では、特定のサービスアクションが発生したときにAmazon EventBridge にイベントを送信するようになりました。詳細については、「Amazon ECS サービスアクションイベント」を参照してください。	2019 年 11 月 25 日
Amazon ECS GPU 最適化 AMI は、G4 インスタンスをサポートします。	Amazon ECS では、Amazon ECS GPU 最適化 AMI の使用時に g4 インスタンスタイプファミリーのサポートが追加されました。詳細については、「GPU ワークロード向けの Amazon ECS タスク定義」を参照してください。	2019 年 10 月 8 日
FireLens for Amazon ECS	FireLens for Amazon ECS は一般提供されています。FireLens for Amazon ECS では、タスク定義パラメータを使用して、ログのストレージと分析のために AWS サービスまたはパートナーの宛先にログをルーティングできます。詳細については、「Amazon ECS ログを AWS サービスまたは AWS Partner に送信する」を参照してください。	2019 年 9 月 30 日
AWS Fargate リージョンの拡大	AWS Fargate with Amazon ECS は、欧州 (パリ)、欧州 (ストックホルム)、および中東 (バーレーン) の各リージョンに拡大しました。	2019 年 9 月 30 日
Amazon ECS で Elastic Inference を使用する Deep Learning Containers	Amazon ECS では、Amazon Elastic Inference アクセラレーターをコンテナにアタッチして、深層学習推論ワークロードをより効率的に実行できます。	2019 年 9 月 3 日
FireLens for Amazon ECS	FireLens for Amazon ECSはパブリックプレビューです。FireLens for Amazon ECS では、タスク定義パラメータを使用して、ログのストレージと分析のために AWS サービスまたはパートナーの宛先にログをルーティングできます。詳細については、「Amazon ECS ログを AWS サービスまたは AWS Partner に送信する」を参照してください。	2019 年 8 月 30 日

変更	説明	日付
CloudWatch コンテナインサイト	CloudWatch Container Insights の一般提供が開始されました。コンテナ化されたアプリケーションとマイクロサービスのメトリクスとログを収集、集計、要約できます。詳細については、「オブザーバビリティが強化された Container Insights を使用し、Amazon ECS コンテナを監視する」を参照してください。	2019 年 8 月 30 日
コンテナレベルのスワップ設定	Amazon ECS では、Linux コンテナインスタンスのスワップメモリ空間の使用状況をコンテナレベルで制御するサポートが追加されました。コンテナごとのスワップ設定を使用すると、タスク定義内の各コンテナでスワップを有効または無効にできます。有効になっているコンテナでは、使用されるスワップ領域の最大量を制限できます。詳細については、「Amazon ECS のコンテナスワップメモリ空間の管理」を参照してください。	2019 年 8 月 16 日
AWS Fargate リージョンの拡大	AWS Fargate with Amazon ECSは、アジアパシフィック (香港) リージョンに拡大されました。	2019 年 8 月 6 日
Elastic Network Interface のトランキング	ENI トランキング機能用にサポートされている追加の Amazon EC2 インスタンスタイプが追加されました。詳細については、「Amazon ECS コンテナネットワークインターフェイスの増加でサポートされるインスタンス」を参照してください。	2019 年 8 月 1 日
サービスに複数のターゲットグループを登録する	サービス定義で複数のターゲットグループを指定するためのサポートが追加されました。詳細については、「Amazon ECS サービスに複数のターゲットグループを登録する」を参照してください。	2019 年 7 月 30 日
Secrets Manager のシークレットを使用した機密データの指定	Secrets Manager のシークレットを使用して機密データを指定するためのチュートリアルが追加されました。詳細については、「Amazon ECS の Secrets Manager シークレットを使用した機密データの指定」を参照してください。	2019 年 7 月 20 日

変更	説明	日付
CloudWatch コンテナインサイト	Amazon ECS は、CloudWatch Container Insights のサポートを追加しました。詳細については、「オブザーバビリティが強化された Container Insights を使用し、Amazon ECS コンテナを監視する」を参照してください。	2019 年 7 月 9 日
Amazon ECS サービスとタスクセットのリソースレベルのアクセス許可	Amazon ECSは、Amazon ECSサービスとタスクのリソースレベルのアクセス許可のサポートを拡張しました。詳細については、「IAM を使用するAmazon Elastic Container Service」を参照してください。	2019 年 6 月 27 日
新しい Amazon ECS 最適化 AMI に AWS-2019-005 用のパッチが適用されました。	Amazon ECS は AWS-2019-005 で説明されている脆弱性に対応するため、Amazon ECS 最適化 AMI を更新しました。	2019 年 6 月 17 日
Elastic Network Interface のトランキング	Amazon ECS で、Elastic Network Interface (ENI) の密度が高い、サポートされている Amazon EC2 インスタンスタイプを使用したコンテナインスタンスの起動サポートが導入されました。これらのインスタンスタイプを使用し、`awsvpcTrunking` アカウント設定をオプトインすることで、新しく起動されたコンテナインスタンスの ENI の密度が高くなります。これにより、各コンテナインスタンスにより多くのタスクを配置することができます。詳細については、「Amazon ECS Linux コンテナインスタンスのネットワークインターフェイスを増やす」を参照してください。	2019 年 6 月 6 日
AWS Fargateプラットフォームバージョン 1.3.0 の更新	2019 年 5 月 1 日以降、起動されるすべての新しい Fargate タスクでは、`splunk` ログドライバーに加えて`awslogs` ログドライバーをサポートします。詳細については、「ストレージとログ記録」を参照してください。	2019 年 5 月 1 日

変更	説明	日付
AWS Fargateプラットフォームバージョン 1.3.0 の更新	2019 年 5 月 1 日以降、起動される新しい Fargate タスクでは、`secretOptions` コンテナ定義パラメータを使用してコンテナのログ設定内の機密データの参照をサポートします。詳細については、「Amazon ECS コンテナに機密データを渡す」を参照してください。	2019 年 5 月 1 日
AWS Fargateプラットフォームバージョン 1.3.0 の更新	2019 年 4 月 2 日以降、起動される新しい Fargateタスクでは、機密データをAWS Secrets Manager シークレットまたは AWS Systems Manager パラメータストアのパラメータに保存してコンテナの定義でそれらを参照することにより、コンテナへの機密データの挿入をサポートします。詳細については、「Amazon ECS コンテナに機密データを渡す」を参照してください。	2019 年 4 月 2 日
AWS Fargate プラットフォームバージョン 1.3.0 の更新	2019 年 3 月 27 日以降、発表された新しい Fargate タスクでは、プロキシ設定、コンテナの起動およびシャットダウンの依存関係、コンテナ別の起動および停止のタイムアウト値を定義できる、追加タスク定義パラメータを使用できます。詳細については、プロキシ設定、コンテナの依存関係、および コンテナのタイムアウト を参照してください。	2019 年 3 月 27 日
Amazon ECS外部デプロイタイプの導入	この 外部 デプロイタイプでは、Amazon ECS サービスのデプロイプロセスを完全に制御するために、すべてのサードパーティーのデプロイコントローラーを使用できます。詳細については、「サードパーティーのコントローラーを使用して Amazon ECS サービスをデプロイする」を参照してください。	2019 年 3 月 27 日

変更	説明	日付
AWSAmazon ECS の Deep Learning Containers	AWS Deep Learning Containers は、Amazon Elastic Container Service (Amazon ECS) の TensorFlow で、モデルをトレーニングおよび処理するための Docker イメージのセットです。Deep Learning Containers は、TensorFlow、Nvidia CUDA (GPU インスタンス用)、およびインテル MKL (CPU インスタンス用) ライブラリを使用して、最適化された環境を提供し、Amazon ECR で利用できます。詳細については、「Amazon ECS で AWS 深層学習コンテナを使用する」を参照してください。	2019 年 3 月 27 日
Amazon ECS、強化されたコンテナ依存関係管理を導入	Amazon ECS は、コンテナのスタートアップとシャットダウンの依存関係を定義したり、開始と停止のタイムアウト値をコンテナごとに定義したりできる、新たなタスク定義のパラメータを導入します。詳細については、「コンテナの依存関係」を参照してください。	2019 年 3 月 7 日
Amazon ECSでの `PutAccountSettingDefault` APIの導入	Amazon ECS では、`PutAccountSettingDefault` API を導入しました。これにより、ユーザーはアカウントのすべてのユーザーおよびロールに対して、デフォルトの ARN/ID 形式のオプトインステータスを設定できます。以前は、アカウントのデフォルトのオプトインステータスを設定するには、アカウント所有者を使用する必要がありました。 詳細については、「Amazon リソースネーム (ARN) と ID」を参照してください。	2019 年 2 月 8 日

変更	説明	日付
Amazon ECS で GPU ワークロードをサポート	Amazon ECS では、GPU 対応コンテナインスタンスを使用してクラスターを作成できるようにすることで、GPU ワークロードのサポートが導入されました。タスク定義で GPU の必要数を指定でき、ECS エージェントにより物理 GPU がコンテナに固定されます。 詳細については、「GPU ワークロード向けの Amazon ECS タスク定義」を参照してください。	2019 年 2 月 4 日
Amazon ECS のシークレットサポートの拡張	Amazon ECS では、機密データをコンテナに挿入するため、タスク定義で直接 AWS Secrets Manager シークレットを使用するためのサポートを拡大しました。 詳細については、「Amazon ECS コンテナに機密データを渡す」を参照してください。	2019 年 1 月 21 日
インターフェイス VPC エンドポイント (AWS PrivateLink)	AWS PrivateLinkを搭載したインターフェイスVPCエンドポイントを設定するためのサポートが追加されました。これにより、インターネット、NAT インスタンス、VPN 接続、またはDirect Connect を経由せずに、VPC とAmazon ECS とをプライベートに接続することができます。 詳細については、インターフェイス VPC エンドポイント (AWS PrivateLink) を参照してください。	2018 年 12 月 26 日

変更	説明	日付
AWS Fargate プラットフォームバージョン 1.3.0	以下を含む新しくリリースされたAWS Fargateプラットフォームバージョン: • AWS Systems Manager パラメータストアパラメータを使用して、コンテナに機密データを挿入する機能のサポートが追加されました。 詳細については、「Amazon ECS コンテナに機密データを渡す」を参照してください。 • Fargateタスクのタスクリサイクルが追加されました。これは、Amazon ECS サービスの一部であるタスクを更新するプロセスです。 詳細については、AWS Fargate on Amazon ECS のタスクの廃止とメンテナンス」を参照してください。 詳細については、「Amazon ECS 向け Fargate プラットフォームバージョン」を参照してください。	2018 年 12 月 17 日
サービスの制限を更新しました	次のサービスの制限が更新されました。 • リージョンごと、アカウントごとのクラスターの数が 1000 から 2000 に引き上げられました。 • クラスターごとのコンテナインスタンスの数が 1000 から 2000 に引き上げられました。 • クラスターごとのサービスの数が 500 から 1000 に引き上げられました。 詳細については、「Amazon ECS の Service Quotas」を参照してください。	2018 年 12 月 14 日

変更	説明	日付
AWS Fargate リージョンの拡大	AWS Fargate with Amazon ECSは、アジアパシフィック (ムンバイ)、およびカナダ (中部) の各リージョンに拡大されました。 詳細については、「AWS Fargate で使用する Amazon ECS でサポートされているリージョン」を参照してください。	2018 年 12 月 7 日
Amazon ECS ブルー/グリーンデプロイ	CodeDeploy を使用した ブルー/グリーンデプロイのサポートがAmazon ECS に追加されました。このデプロイタイプでは、本番稼働用トラフィックを送信する前にサービスの新しいデプロイメントを検証することができます。 詳細については、「Amazon ECS 用の CodeDeploy ブルー/グリーンデプロイ」を参照してください。	2018 年 11 月 27 日
Amazon ECS 最適化 Amazon Linux 2 (arm64) AMI がリリースされました	Amazon ECS は、Amazon ECS に最適化された、arm64 アーキテクチャのための Amazon Linux 2 AMI をリリースしました。 詳細については、「Amazon ECS に最適化された Linux AMI」を参照してください。	2018 年 11 月 26 日
タスク定義に追加の Docker フラグのサポートが追加されました	Amazon ECS ではタスク定義に次の Docker フラグのサポートを導入しました。 • IPC モード • PID モード	2018 年 11 月 16 日

変更	説明	日付
Amazon ECS シークレットのサポート	AWS Systems Manager パラメータストアパラメータを使用して、コンテナに機密データを挿入する機能のサポートが Amazon ECS に追加されました。 詳細については、「Amazon ECS コンテナに機密データを渡す」を参照してください。	2018 年 11 月 15 日
リソースのタグ付け	サービス、タスク定義、タスク、クラスター、コンテナインスタンスにメタデータタグを追加する機能のサポートがAmazon ECS に追加されました。 詳細については、「Amazon ECS リソースにタグ付けする」を参照してください。	2018 年 11 月 15 日
AWS Fargateリージョンの拡大	AWS Fargate with Amazon ECS は米国西部 (北カリフォルニア)、アジアパシフィック (ソウル) の各リージョンに拡張されました。 詳細については、「Amazon ECS の AWS Fargate 用のアーキテクト」を参照してください。	2018 年 11 月 7 日
サービスの制限を更新しました	次のサービスの制限が更新されました。 • リージョンごとアカウントごとに、Fargate 起動タイプを使用するタスクの数が20 から50 に引き上げられました。 • Fargate 起動タイプを使用するタスクのパブリック IP アドレスの数が 20 から50 に引き上げられました。 詳細については、「Amazon ECS の Service Quotas」を参照してください。	2018 年 10 月 31 日

変更	説明	日付
AWS Fargateリージョンの拡大	AWS Fargate with Amazon ECSは欧州 (ロンドン) リージョンに拡大されました。 詳細については、「Amazon ECS の AWS Fargate 用のアーキテクト」を参照してください。	2018 年 10 月 26 日
Amazon ECS に最適化された AMI 2 AMI のリリース	Amazon ECS は、2 つのバリアントでサービスに最適化された Linux AMI を提供しています。最新の推奨バージョンは、x に基づいています。Amazon ECS は Amazon Linux AMI に基づいている AMI も提供しますが、ワークロードを Amazon Linux 2 のバリアントに移行することをお勧めします。Amazon Linux AMI のサポートは 2020 年 6 月 30 日までには終了します。 詳細については、「Amazon ECS に最適化された Linux AMI」を参照してください。	2018 年 10 月 18 日
Amazon ECS タスクメタデータエンドポイントバージョン 3	Amazon ECS コンテナエージェントの 1.21.0 バージョン以降では、エージェントはタスクの各コンテナに、環境変数 `ECS_CONTAINER_METADATA_URI` を挿入します。タスクメタデータのバージョン 3 エンドポイントをクエリするとき、さまざまなタスクメタデータおよび Docker 統計を、Amazon ECS コンテナエージェントによって指定される HTTP エンドポイントで `awsvpc` ネットワークモードを使用するタスクで利用できます。詳細については、「Amazon ECS メタデータを使用したワークロードのモニタリング」を参照してください。	2018 年 10 月 18 日

変更	説明	日付
Amazon ECS サービス検出リージョンの拡大	Amazon ECS サービス検出は、サポート範囲が拡張され、カナダ (中部)、南米 (サンパウロ)、アジアパシフィック (ソウル)、アジアパシフィック (ムンバイ)、および欧州 (パリ) の各リージョンで利用できます。 詳細については、「サービス検出を使用して Amazon ECS サービスを DNS 名で接続する」を参照してください。	2018 年 9 月 27 日
コンテナ定義に追加の Docker フラグのサポートを追加する	Amazon ECS ではコンテナ定義に次の Docker フラグのサポートを導入しました。 • システムコントロール • インタラクティブ • 擬似ターミナル	2018 年 9 月 17 日
AWS Fargateタスク を使用した Amazon ECS のプライベートレジストリ認証のサポート	Amazon ECSはAWS Secrets Managerを使用したプライベートレジストリ認証を使用した Fargate タスクのサポートが導入されました。この機能では、認証情報を安全に保管してコンテナの定義で参照できるため、タスクでプライベートイメージを使用することができます。 詳細については、「Amazon ECS での AWS 以外のコンテナイメージの使用」を参照してください。	2018 年 9 月 10 日

変更	説明	日付
Amazon ECS サービス検出リージョンの拡大	Amazon ECSサービス検出は、サポート範囲が拡張され、アジアパシフィック (シンガポール)、アジアパシフィック (シドニー)、アジアパシフィック (東京)、欧州 (フランクフルト)、欧州 (ロンドン) の各リージョンで利用できます。 詳細については、「サービス検出を使用して Amazon ECS サービスを DNS 名で接続する」を参照してください。	2018 年 8 月 30 日
Fargateタスクのサポートでスケジュールされたタスク	Amazon ECS ではFargate 起動タイプのスケジュールされたタスクのサポートが導入されました。	2018 年 8 月 28 日
AWS Secrets Managerサポートを使用したプライベートレジストリ認証	Amazon ECS では、AWS Secrets Managerを使用したプライベートレジストリ認証のサポートが導入されました。この機能では、認証情報を安全に保管してコンテナの定義で参照できるため、タスクでプライベートイメージを使用することができます。 詳細については、「Amazon ECS での AWS 以外のコンテナイメージの使用」を参照してください。	2018 年 8 月 16 日
Docker ボリュームのサポートが追加されました	Amazon ECS では、Docker ボリュームのサポートが導入されました。 詳細については、「Amazon ECS タスクのストレージオプション」を参照してください。	2018 年 8 月 9 日
AWS Fargateリージョンの拡大	AWS Amazon ECS でのFargate の使用は、欧州 (フランクフルト)、アジアパシフィック (シンガポール)、アジアパシフィック (シドニー) の各リージョンに拡張されました。 詳細については、「Amazon ECS の AWS Fargate 用のアーキテクト」を参照してください。	2018 年 7 月 19 日

変更	説明	日付
Amazon ECSサービススケジューラ戦略の追加	Amazon ECSにサービススケジューラ戦略の概念を導入しました。 利用できる 2 つのサービススケジューラ戦略があります。 • REPLICA - レプリカスケジュール戦略では、クラスター全体で必要数のタスクを配置して維持します。デフォルトでは、サービススケジューラによってタスクはアベイラビリティーゾーン間に分散されます。タスク配置の戦略と制約を使用すると、タスク配置の決定をカスタマイズできます。詳細については、「レプリカスケジューリング戦略」を参照してください。 • DAEMON - デーモンのスケジュール戦略では、指定したすべてのタスク配置制約を満たすクラスター内のアクティブなコンテナインスタンスごとに、1 つのタスクのみをデプロイします。この戦略を使用する場合、タスクの必要数や配置戦略、サービスの自動スケーリングポリシーを指定する必要はありません。詳細については、「デーモンスケジューリング戦略」を参照してください。 Note Fargate タスクは DAEMON スケジュール戦略をサポートしていません。 .	2018 年 12 月 6 日

変更	説明	日付
Amazon ECS コンテナエージェント v1.18.0	新しいバージョンの Amazon ECS コンテナエージェントがリリースされ、以下の機能が追加されました。 • `ECS_IMAGE_PULL_BEHAVIOR` パラメータを使用したコンテナエージェントイメージのプル動作のカスタマイズのサポートが追加されました。詳細については、「Amazon ECS コンテナエージェントの設定」を参照してください。 詳細については、amazon-ecs-agent github を参照してください。	2018 年 5 月 24 日
Service Discovery の設定時に `bridge` および `host` ネットワークモードのサポートが追加されました。	`bridge` または `host` ネットワークモードを指定するタスク定義を使用して Amazon ECS サービスのサービス検出を設定するためのサポートが追加されました。詳細については、「サービス検出を使用して Amazon ECS サービスを DNS 名で接続する」を参照してください。	2018 年 5 月 22 日
Amazon ECS に最適化された AMI のその他のメタデータパラメータのサポートが追加されました	Amazon ECS に最適化された AMI の ID、イメージ名、オペレーティングシステム、コンテナエージェントバージョン、およびランタイムバージョンをプログラムで取得できるサブパラメータが追加されました。Systems Manager Parameter Store API を使用してメタデータをクエリします。詳細については、「Amazon ECS に最適化された Linux AMI メタデータを取得する」を参照してください。	2018 年 5 月 9 日
AWS Fargateリージョンの拡大	AWS Fargate with Amazon ECS は、米国東部 (オハイオ)、米国西部 (オレゴン)、欧州西部 (アイルランド) リージョンに拡大されます。 詳細については、「Amazon ECS の AWS Fargate 用のアーキテクト」を参照してください。	2018 年 4 月 26 日

変更	説明	日付
Amazon ECS に最適化された AMI メタデータの取得	Systems Manager パラメータストア API を使用して、Amazon ECS に最適化された AMI メタデータをプログラムで取得する機能を追加しました。詳細については、「Amazon ECS に最適化された Linux AMI メタデータを取得する」を参照してください。	2018 年 4 月 10 日
AWS Fargate プラットフォームバージョン	以下を含む新しくリリースされたAWS Fargateプラットフォームバージョン: • Amazon ECS メタデータを使用したワークロードのモニタリング のサポートを追加しました。 • ヘルスチェック のサポートを追加しました。 • サービス検出を使用して Amazon ECS サービスを DNS 名で接続する のサポートを追加しました。 詳細については、「Amazon ECS 向け Fargate プラットフォームバージョン」を参照してください。	2018 年 3 月 26 日
Amazon ECSサービス検出	Amazon ECS サービス検出をサポートする Route 53 との統合を追加しました。詳細については、「サービス検出を使用して Amazon ECS サービスを DNS 名で接続する」を参照してください。	2018 年 3 月 22 日
Docker shm-size および tmpfs のサポート	Amazon ECS タスク定義に Docker shm-size および tmpfs パラメータのサポートが追加されていること。 更新された ECS CLI 構文の詳細については、「Linux パラメータ」を参照してください。	2018 年 3 月 20 日
コンテナヘルスチェック	コンテナ定義に追加された Docker ヘルスチェックのサポート。詳細については、「ヘルスチェック」を参照してください。	2018 年 3 月 8 日

変更	説明	日付
AWS Fargate	AWS Fargate を使用する Amazon ECS の概要が追加されました。詳細については、「Amazon ECS の AWS Fargate 用のアーキテクト」を参照してください。	2018 年 2 月 22 日
Amazon ECSタスクメタデータエンドポイント	Amazon ECSコンテナエージェントのバージョン 1.17.0 以降、Amazon ECSコンテナエージェントによって提供される HTTP エンドポイントで awsvpc ネットワークモードを使用するタスクで、さまざまなタスクメタデータおよび Docker 統計を使用できます。詳細については、「Amazon ECS メタデータを使用したワークロードのモニタリング」を参照してください。	2018 年 2 月 8 日
ターゲット追跡ポリシーを使用したAmazon ECS Service Auto Scaling	Amazon ECS コンソールにターゲット追跡ポリシーを使用した ECS Service Auto Scaling のサポートが追加されました。詳細については、「ターゲットメトリクスを使用して Amazon ECS サービスをスケールする」を参照してください。 ECS の初回実行ウィザードの前段階チュートリアルのステップスケーリングが削除されていること。これはターゲット追跡用の新しいチュートリアルに置き換えられます。	2018 年 2 月 8 日
Docker 17.09 のサポート	Docker 17.09 のサポートを追加しました。詳細については、「Amazon ECS に最適化された Linux AMI」を参照してください。	2018 年 1 月 18 日
新しいサービススケジューラの動作	起動に失敗するサービスタスクの動作に関する情報を更新しました。サービスタスクが連続して失敗する場合にトリガーされる新しいサービスイベントメッセージについて文書化しました。	2018 年 1 月 11 日

変更	説明	日付
Elastic Load Balancing のヘルスチェックの初期化待機期間	ヘルスチェックの待機期間を指定する機能を追加しました。	2017 年 12 月 27 日
タスクレベルの CPU とメモリ	タスク定義でタスクレベルの CPU とメモリを指定するためのサポートを追加しました。詳細については、「TaskDefinition」を参照してください。	2017 年 12 月 12 日
タスク実行ロール	Amazon ECSコンテナエージェントはユーザーに代わって Amazon ECS API アクションを呼び出すため、エージェントがユーザーに属していることをサービスに伝えるために、IAM ポリシーおよびロールが必要です。次のアクションがタスク実行ロールの対象になっています。 • コンテナイメージをプルするためのAmazon ECR の呼び出し • コンテナアプリケーションログを保存するための CloudWatch の呼び出し 詳細については、「Amazon ECS タスク実行IAM ロール」を参照してください。	2017 年 12 月 7 日
Windows コンテナが GA をサポート	Windows Server 2016 コンテナのサポートが追加されました。詳細については、「Amazon ECS に最適化された AMI バリアント」を参照してください。	2017 年 5 月 12 日
AWS Fargateの一般提供	Fargate 起動タイプを使用したAmazon ECS サービスの起動のサポートを追加しました。詳細については、「Amazon ECS のソリューションの構築」を参照してください。	2017 年 11 月 29 日
Amazon ECS の名前の変更	Amazon Elastic Container Service の名前が変更されました (旧 Amazon EC2 Container Service)。	2017 年 11 月 21 日

変更	説明	日付
タスクネットワーク	awsvpc ネットワークモードで利用できるタスクネットワーキング機能により、Amazon EC2 インスタンスと同じネットワーキングプロパティが Amazon ECS タスクに提供されます。タスク定義で awsvpc ネットワークモードを使用すると、このタスク定義から起動されるすべてのタスクが、独自の Elastic Network Interface、プライマリプライベート IP アドレス、および内部 DNS ホスト名を取得します。タスクネットワーキング機能により、コンテナネットワークを簡素化できるだけでなく、コンテナ化されたアプリケーションの相互通信や、コンテナ化されたアプリケーションと VPC 内のその他のサービスとの通信に対する統制力を強化できます。詳細については、「EC2 の Amazon ECS タスクネットワークオプション」を参照してください。	2017 年 11 月 14 日
Amazon ECS コンテナメタデータ	Amazon ECS コンテナが、Docker コンテナ、イメージ ID、ネットワーク設定、Amazon ARN などのメタデータにアクセスできるようになりました。詳細については、「Amazon ECS コンテナメタデータファイル」を参照してください。	2017 年 11 月 2 日
Docker 17.06 のサポート	Docker 17.06 のサポートを追加しました。詳細については、「Amazon ECS に最適化された Linux AMI」を参照してください。	2017 年 11 月 2 日
Docker フラグ device および init のサポート	`LinuxParameters` パラメータ (`devices` および `initProcessEnabled`) を使用した、タスク定義での Docker の device および init 機能のサポートを追加しました。詳細については、LinuxParameters を参照してください。	2017 年 11 月 2 日

変更	説明	日付
Docker フラグ cap-add および cap-drop のサポート	`LinuxParameters` パラメータ (`capabilities`) を使用した、タスク定義での Docker の cap-add および cap-drop 機能のサポートを追加しました。詳細については、LinuxParameters を参照してください。	2017 年 9 月 22 日
Network Load Balancer のサポート	Amazon ECS では、Amazon ECS コンソールで Network Load Balancer のサポートが追加されました。	2017 年 9 月 7 日
RunTask の上書き	タスクの実行時のタスク定義の上書きのサポートが追加されました。これにより、タスク定義の変更中に、新しいタスク定義のリビジョンを作成することなくタスクを実行できます。詳細については、「Amazon ECS タスクとしてのアプリケーションの実行」を参照してください。	2017 年 6 月 27 日
Amazon ECS のスケジュールされたタスク	cron を使用したタスクのスケジューリングのサポートを追加しました。	2017 年 6 月 7 日
Amazon ECS コンソールのスポットインスタンス	Amazon ECS コンソール内でスポットフリートコンテナインスタンスを作成するためのサポートを追加しました。詳細については、「Amazon ECS Linux コンテナインスタンスの起動」を参照してください。	2017 年 6 月 6 日
新しい Amazon ECS 最適化 AMI リリース用の Amazon SNS 通知	新しい Amazon ECS 最適化 AMI リリースに関する SNS 通知にサブスクライブする機能が追加されました。	2017 年 3 月 23 日
マイクロサービスとバッチジョブ	Amazon ECS の 2 つの一般的ユースケース (マイクロサービスおよびバッチジョブ) に関するドキュメントを追加しました。	2017 年 2 月

変更	説明	日付
コンテナインスタンスのドレイン	コンテナインスタンスのドレインのサポートを追加しました。これにより、クラスターからコンテナインスタンスを削除するメソッドが提供されます。詳細については、「Amazon ECS コンテナインスタンスをドレインする」を参照してください。	2017 年 1 月 24 日
Docker 1.12 のサポート	Docker 1.12 のサポートを追加しました。詳細については、「Amazon ECS に最適化された Linux AMI」を参照してください。	2017 年 1 月 24 日
新しいタスク配置戦略	タスク配置戦略のサポート (属性ベースの配置、ビンパック、アベイラビリティーゾーン分散、ホストごとに 1 つ) を追加しました。詳細については、「戦略を使用して Amazon ECS タスク配置を定義する」を参照してください。	2016 年 12 月 29 日
Windows コンテナのベータ版のサポート	Windows Server 2016 コンテナ (ベータ版) のサポートが追加されました。詳細については、「Amazon ECS に最適化された AMI バリアント」を参照してください。	2016 年 12 月 20 日
Blox OSS のサポート	Blox OSS のサポートが追加されました。これにより、カスタムタスクスケジューラを使用できます。詳細については、「Amazon ECS でコンテナをスケジュールする」を参照してください。	2016 年 12 月 1 日
CloudWatch Events の Amazon ECS イベントストリーム	Amazon ECS が、コンテナインスタンスとタスクの状態の変更を CloudWatch Events に送信するようになりました。詳細については、「EventBridge を使用して Amazon ECS エラーへの対応を自動化する」を参照してください。	2016 年 11 月 21 日
CloudWatch Logs への Amazon ECS コンテナのロギング	awslogs ドライバーがコンテナログストリームを CloudWatch Logs に送信するためのサポートを追加しました。詳細については、「Amazon ECS ログを CloudWatch に送信する」を参照してください。	2016 年 9 月 12 日

変更	説明	日付
動的ポートの Elastic Load Balancing をサポートする Amazon ECS サービス	リスナーあたり複数の instance:port の組み合わせをサポートするためのロードバランサーのサポートを追加しました。これにより、コンテナの柔軟性が高まります。Docker でコンテナのホストポートを動的に定義し、ECS スケジューラでロードバランサーを使用して instance:port を登録できます。詳細については、「ロードバランサーを使用して Amazon ECS サービストラフィックを分散する」を参照してください。	2016 年 8 月 11 日
Amazon ECS タスク用の IAM ロール	タスクへの IAM ロールの関連付けのサポートが追加されました。これにより、コンテナインスタンス全体に対して 1 つのロールを使用するのではなく、コンテナに対するより詳細なアクセス許可が提供されます。詳細については、「Amazon ECS タスクの IAM ロール」を参照してください。	2016 年 7 月 13 日
Docker 1.11 のサポート	Docker 1.11 のサポートを追加しました。詳細については、「Amazon ECS に最適化された Linux AMI」を参照してください。	2016 年 5 月 31 日
タスクの自動スケーリング	Amazon ECS で、サービスによって実行されるタスクの自動的なスケーリングのサポートが追加されました。詳細については、「Amazon ECS サービスを自動的にスケールする」を参照してください。	2016 年 5 月 18 日
タスクファミリーでのタスク定義のフィルタリング	タスク定義ファミリーに基づいてタスク定義のリストをフィルタリングするサポートが追加されました。詳細については、「ListTaskDefinitions」を参照してください。	2016 年 5 月 17 日
Docker コンテナおよび Amazon ECS エージェントのログ記録	Amazon ECS で、問題のトラブルシューティングを簡単にするため、コンテナインスタンスから ECS エージェントと Docker コンテナログを CloudWatch Logs に送信する機能が追加されました。	2016 年 5 月 5 日

変更	説明	日付
ECS 最適化 AMI が Amazon Linux 2016.03 をサポートするようになりました。	ECS 最適化 AMI では Amazon Linux 2016.03 のサポートを追加しました。詳細については、「Amazon ECS に最適化された Linux AMI」を参照してください。	2016 年 4 月 5 日
Docker 1.9 のサポート	Docker 1.9 のサポートを追加しました。詳細については、「Amazon ECS に最適化された Linux AMI」を参照してください。	2015 年 12 月 22 日
クラスターの CPU およびメモリ予約に関する CloudWatch メトリクス	Amazon ECS で、CPU およびメモリ予約に関するカスタム CloudWatch メトリクスが追加されました。	2015 年 12 月 22 日
新しい Amazon ECS の初回実行のエクスペリエンス	Amazon ECS コンソールの初回実行のエクスペリエンスで、ゼロクリックのロール作成が追加されました。	2015 年 11 月 23 日
複数のアベイラビリティーゾーンにわたるタスクの配置	Amazon ECS サービススケジューラで、複数のアベイラビリティーゾーンにわたるタスク配置のサポートが追加されました。	2015 年 10 月 8 日
Amazon ECS クラスターとサービスの CloudWatch メトリクス	Amazon ECS で、クラスターの各コンテナインスタンス、サービス、タスク定義ファミリーの CPU およびメモリ使用率に関するカスタム CloudWatch メトリクスが追加されました。これらの新しいメトリクスを使用すると、Auto Scaling グループを使用してクラスターでコンテナインスタンスをスケールするか、カスタム CloudWatch アラームを作成できます。	2015 年 8 月 17 日
UDP ポートのサポート	タスク定義で UDP ポートのサポートが追加されました。	2015 年 7 月 7 日
環境変数の上書き	deregisterTaskDefinition と環境変数による runTask の上書きのサポートが追加されました。	2015 年 6 月 18 日

変更	説明	日付
自動化された Amazon ECS エージェントの更新	コンテナインスタンスで実行されている ECS エージェントバージョンを表示する機能を追加しました。AWS マネジメントコンソール、AWS CLI、および SDK から ECS エージェントを更新することもできます。	2015 年 6 月 11 日
Amazon ECS サービススケジューラと Elastic Load Balancing の統合	サービスを定義し、そのサービスを Elastic Load Balancing ロードバランサーに関連付ける機能を追加しました。	2015 年 4 月 9 日
Amazon ECS GA	Amazon ECS は米国東部 (バージニア北部)、米国西部 (オレゴン)、アジアパシフィック (東京)、欧州 (アイルランド) の各リージョンで一般提供されます。	2015 年 4 月 9 日