aws

ユーザーガイド

AWS CodeDeploy



API バージョン 2014-10-06

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

AWS CodeDeploy: ユーザーガイド

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

Amazon の商標とトレードドレスは、Amazon 以外の製品またはサービスとの関連において、顧客に混乱を招いたり、Amazon の名誉または信用を毀損するような方法で使用することはできません。Amazon が所有していないその他のすべての商標は Amazon との提携、関連、支援関係の有無にかかわらず、それら該当する所有者の資産です。

Table of Contents

What is CodeDeploy? 1
の利点 AWS CodeDeploy 2
CodeDeploy コンピューティングプラットフォームの概要 3
CodeDeploy デプロイタイプの概要 10
インプレースデプロイの概要 12
Blue/Green デプロイの概要 13
ご意見をお待ちしております 18
プライマリコンポーネント 18
アプリケーション 18
コンピューティングプラットフォーム 19
Deployment configuration 20
デプロイグループ 20
デプロイタイプ 20
デプロイタイプ 22
リビジョン 22
サービスロール 22
ターゲットリビジョン 23
他のコンポーネント 23
デプロイ 23
AWS Lambda コンピューティングプラットフォームでのデプロイ 24
Amazon ECS コンピューティングプラットフォームのデプロイ 27
EC2/オンプレミスコンピューティングプラットフォームの Blue/Green デプロイ 39
アプリケーション仕様ファイル 46
Amazon ECS コンピューティングプラットフォーム上の AppSpec ファイル 47
AWS Lambda コンピューティングプラットフォーム上の AppSpec ファイル 47
EC2 オンプレミスコンピューティングプラットフォーム上の AppSpec ファイル 48
CodeDeploy エージェントが AppSpec ファイルを使用する方法 48
開始方法 50
ステップ 1: セットアップ 50
にサインアップする AWS アカウント 50
ステップ 2: サービスロールを作成する 51
サービスロールの作成 (コンソール) 53
サービスロールの作成 (CLI) 55
サービスロール ARN の取得 (コンソール) 58

サービスロール ARN の取得 (CLI) 59
ステップ 3: CodeDeploy ユーザーのアクセス許可を制限する 59
ステップ 4: IAM インスタンスプロファイルを作成する 63
Amazon EC2 インスタンス(CLI)の IAM インスタンスプロファイルを作成する 64
Amazon EC2 インスタンス(コンソール)の IAM インスタンスプロファイルを作成する 68
製品およびサービスの統合 72
他の AWS サービスとの統合 72
Amazon EC2 Auto Scaling 80
Elastic Load Balancing 88
パートナーの製品とサービスとの統合 93
GitHub 98
コミュニティから統合の例 102
ブログ記事 103
チュートリアル 104
チュートリアル: WordPress を Windows 以外のインスタンスへデプロイする 104
ステップ 1: Amazon EC2 インスタンスを起動する 105
ステップ 2: ソース コンテンツを設定する 108
ステップ 3: Amazon S3 にアプリケーションをアップロードする 113
ステップ 4: アプリケーションをデプロイする 118
ステップ 5: アプリケーションを更新して再デプロイする 125
ステップ 6: クリーンアップする 129
チュートリアル: Windows Server インスタンスへの HelloWorld アプリケーションのデプロイ 132
ステップ 1: Amazon EC2 インスタンスを起動する 133
ステップ 2: ソース コンテンツを設定する 135
ステップ 3: Amazon S3 にアプリケーションをアップロードする 139
手順 4: アプリケーションをデプロイする 144
ステップ 5: アプリケーションを更新して再デプロイする 149
ステップ 6: クリーンアップする 152
チュートリアル: オンプレミスインスタンスへアプリケーションをデプロイ 156
前提条件 157
ステップ 1: オンプレミスインスタンスを設定する 157
ステップ 2: サンプルのアプリケーションリビジョンを作成する 157
ステップ 3: アプリケーションリビジョンをバンドルし、Amazon S3 にアップロードする . 162
ステップ 4: アプリケーションリビジョンをデプロイする 163
ステップ 5: デプロイを確認する 163

ステップ 6: リソースをクリーンアップする 163
チュートリアル: Auto Scaling グループへデプロイします。 166
前提条件 166
ステップ 1: Auto Scaling グループを作成して設定します。 167
ステップ 2: Auto Scaling グループにアプリケーションをデプロイする 174
ステップ 3: 結果の確認 184
ステップ 4: Auto Scaling グループの Amazon EC2 インスタンスの数を増やす 186
ステップ 5: 結果を再度確認します 187
ステップ 6: クリーンアップする 190
チュートリアル: GitHub からアプリケーションをデプロイします。 192
前提条件 193
ステップ 1: GitHub アカウントを設定します。 193
ステップ 2: GitHub リポジトリを作成します。 193
ステップ 3: GitHub リポジトリにサンプルアプリケーションをアップロードします。 196
ステップ 4: インスタンスをプロビジョニングします。 201
ステップ 5: アプリケーションおよびデプロイグループを作成します。 202
ステップ 6: アプリケーションをインスタンスにデプロイします。 204
ステップ 7: デプロイをモニタリングおよび確認します。 208
ステップ 8: クリーンアップする 209
チュートリアル: Amazon ECS へアプリケーションをデプロイする 211
前提条件 213
ステップ 1: Amazon ECS アプリケーションを更新する 214
ステップ 2: AppSpec ファイルを作成します。 215
ステップ 3: CodeDeploy コンソールを使用してアプリケーションをデプロイする 216
ステップ 4: クリーンアップする 221
チュートリアル: 検証テストを使用して Amazon ECS サービスをデプロイする 221
前提条件 224
ステップ 1: テストリスナーを作成する 224
ステップ 2: Amazon ECS アプリケーションを更新する 225
ステップ 3: ライフサイクルフック Lambda 関数を作成する 225
ステップ 4: AppSpec ファイルを更新する 228
ステップ 5: CodeDeploy コンソールを使用して Amazon ECS サービスをデプロイする 229
ステップ 6: CloudWatch Logs で Lambda フック関数の出力を表示する 231
ステップ 7: クリーンアップする 232
チュートリアル: SAM を使用して Lambda AWS 関数をデプロイする 233
前提条件 234

ステップ 1: インフラストラクチャをセットアップする 234
ステップ 2: Lambda 関数を更新します 249
ステップ 3: 更新された Lambda 関数をデプロイします。 252
ステップ 4: デプロイの結果を表示する 254
ステップ 5: クリーンアップ 257
CodeDeploy エージェントの使用 259
CodeDeploy エージェントで対応するオペレーティングシステム 260
対応する Amazon EC2 AMI オペレーティングシステム 260
サポートされているオンプレミスオペレーションシステム 261
CodeDeploy エージェントの通信プロトコルとポート 261
CodeDeploy エージェントのバージョン履歴 262
CodeDeploy プロセスの管理 281
アプリケーションリビジョンとログファイルのクリーンアップ 282
CodeDeploy エージェントでインストールされるファイル 283
CodeDeploy エージェントのオペレーションの管理 286
CodeDeploy エージェントが実行されていることの確認 287
CodeDeploy エージェントのバージョンを特定します。 290
CodeDeploy エージェントをインストールする 293
CodeDeploy エージェントを更新する 307
CodeDeploy エージェントをアンインストールする 309
CodeDeploy エージェントログを CloudWatch に送信する 311
インスタンスの使用 316
Amazon EC2 インスタンスとオンプレミスインスタンスの比較 316
CodeDeploy のためのインスタンスタスク 318
CodeDeploy デプロイのインスタンスのタグ付け 319
例 1: 単一タググループ、単一タグ 320
例 2: 単一タググループ、複数タグ 321
例 3: 複数タググループ、複数タグ 323
例 4: 複数タググループ、複数タグ 325
Amazon EC2 インスタンスの使用 329
CodeDeploy のための Amazon EC2 インスタンスを作成します。 329
Amazon EC2 インスタンスを作成する (CloudFormation テンプレート) 336
Amazon EC2 インスタンスの設定 347
オンプレミスインスタンスの使用 351
オンプレミスインスタンスを設定するための前提条件 352
オンプレミスインスタンスの登録 354

オンプレミスインスタンスオペレーションの管理 ... 391
インスタンスの詳細の表示 ... 398
インスタンスの詳細の表示 (コンソール) ... 399
インスタンスの詳細の表示 (CLI) ... 400
インスタンスの状態 ... 400
ヘルスステータス ... 401
正常なインスタンスの最小数について ... 402
アベイラビリティーゾーンあたりの正常なインスタンスの最小数について ... 406
デプロイ設定の使用 ... 409
EC2/オンプレミスコンピューティングプラットフォームのデプロイ設定 ... 409
事前定義されたデプロイ設定 ... 409
Amazon ECS コンピューティングプラットフォームのデプロイ設定 ... 414
Amazon ECS の事前定義されたデプロイ設定 ... 414
CloudFormation Blue/Green デプロイのデプロイ設定 (Amazon ECS) ... 415
AWS Lambda コンピューティングプラットフォームでのデプロイ設定 ... 416
Lambda の事前定義されたデプロイ設定 ... 416
トピック ... 417
デプロイ設定を作成する ... 417
デプロイ設定を作成する (コンソール) ... 418
デプロイ設定を作成する (AWS CLI) ... 420
デプロイ設定の詳細の表示 ... 421
デプロイ設定の詳細の表示 (コンソール) ... 422
デプロイ設定の表示 (CLI) ... 422
デプロイ設定を削除する ... 423
アプリケーションを用いた作業 ... 424
アプリケーションの作成 ... 425
インプレースデプロイ (コンソール) 用のアプリケーションを作成 ... 427
Blue/Green デプロイ (コンソール) のアプリケーションを作成します。 ... 430
Amazon ECS サービスデプロイ用のアプリケーションを作成 (コンソール) ... 435
AWS Lambda 関数デプロイ用のアプリケーションを作成する (コンソール) ... 437
アプリケーションの作成 (CLI) ... 438
アプリケーションの詳細を表示する ... 439
アプリケーションの詳細を表示する (コンソール) ... 439
アプリケーションの詳細を表示する (CLI) ... 439
通知ルールの作成 ... 440
アプリケーションの名前を変更する ... 443

アプリケーションの削除 ... 443
アプリケーションの削除 (コンソール) ... 444
アプリケーションの削除 (AWS CLI) ... 444
デプロイグループの使用 ... 445
Amazon ECS コンピューティングプラットフォームのデプロイでのデプロイグループ ... 445
AWS Lambda コンピューティングプラットフォームデプロイのデプロイグループ ... 445
EC2 オンプレミスコンピューティングプラットフォームのデプロイでのデプロイグループ ... 445
トピック ... 446
デプロイグループの作成 ... 447
インプレースデプロイ用のデプロイグループを作成する (コンソール) ... 447
EC2/オンプレミス Blue/Green デプロイ用のデプロイグループを作成する (コンソール) ... 451
Amazon ECS デプロイ用のデプロイグループを作成する (コンソール) ... 457
CodeDeploy Amazon EC2 デプロイ用の Elastic Load Balancing でロードバランサーをセットアップする ... 458
CodeDeploy Amazon ECS デプロイ用のロードバランサー、ターゲットグループ、リスナーをセットアップする ... 460
デプロイグループの作成 (CLI) ... 465
デプロイグループの詳細の表示 ... 466
デプロイグループの詳細の表示 (コンソール) ... 467
デプロイグループの詳細の表示 (CLI) ... 467
デプロイグループの設定を変更する ... 468
デプロイグループの設定を変更する (コンソール) ... 468
デプロイグループの設定を変更する (CLI) ... 469
デプロイグループの詳細オプションの設定 ... 470
デプロイグループを削除する ... 474
デプロイグループを削除する (コンソール) ... 474
デプロイグループを削除する (CLI) ... 475
アプリケーションリビジョンの操作 ... 476
リビジョンの計画を立てる ... 476
AppSpec ファイルを追加する ... 477
Amazon ECS デプロイ用の AppSpec ファイルを追加する ... 477
AWS Lambda デプロイ用の AppSpec ファイルを追加する ... 481
EC2/オンプレミスデプロイに AppSpec ファイルを追加する ... 483
レポジトリタイプの選択 ... 487
リビジョンをプッシュする ... 490
を使用してリビジョンをプッシュする AWS CLI ... 492

アプリケーションリビジョンの詳細の表示 ... 494
アプリケーションリビジョンの詳細の表示 (コンソール) ... 495
アプリケーションリビジョンの詳細の表示 (CLI) ... 495
アプリケーションリビジョンの登録 ... 497
CodeDeploy で Amazon S3 にリビジョンを登録する (CLI) ... 497
CodeDeploy による GitHub でのリビジョンの登録 (CLI) ... 498
デプロイでの作業 ... 500
デプロイを作成する ... 501
デプロイの前提条件 ... 502
Amazon ECS コンピューティングプラットフォームのデプロイの作成 (コンソール) ... 505
AWS Lambda コンピューティングプラットフォームのデプロイを作成する (コンソール) ... 507
EC2/オンプレミスコンピューティングプラットフォームのデプロイ作成 (コンソール) ... 509
Amazon ECS コンピューティングプラットフォームのデプロイの作成 (CLI) ... 514
AWS Lambda コンピューティングプラットフォームのデプロイを作成する (CLI) ... 515
EC2/ オンプレミスコンピューティングプラットフォームのデプロイ作成 (CLI) ... 517
を使用して Amazon ECS ブルー/グリーンデプロイを作成する CloudFormation ... 521
デプロイの詳細の表示 ... 524
デプロイの詳細の表示 (コンソール) ... 525
デプロイの詳細の表示 (CLI) ... 526
デプロイのログデータの表示 ... 526
Amazon CloudWatch コンソールでのログファイルのデータの表示 ... 527
インスタンスでのログファイルの表示 ... 527
デプロイの停止 ... 530
デプロイ (コンソール) の停止 ... 531
デプロイ (CLI) の停止 ... 532
デプロイを使用した再デプロイおよびロールバック ... 532
自動ロールバック ... 533
手動ロールバック ... 533
ロールバックおよび再デプロイのワークフロー ... 533
既存のコンテンツでのロールバック動作 ... 535
別の AWS アカウントにアプリケーションをデプロイする ... 539
ステップ 1: いずれかのアカウントで S3 バケットを作成する ... 539
ステップ 2: Amazon S3 バケットへのアクセス許可を本番稼働用アカウントの インスタンスプロファイルに付与する ... 540
ステップ 3: 本番稼働用アカウントでリソースとクロスアカウントロールを作成する ... 541
ステップ 4: Amazon S3 バケットにアプリケーションリビジョンをアップロードする ... 542

ステップ 5: クロスアカウントロールを引き受け、アプリケーションをデプロイする 543
ローカルマシンのデプロイパッケージの検証 543
前提条件 544
ローカルのデプロイを作成する。 546
例 550
デプロイのモニタリング 552
自動ツール 553
手動ツール 554
Amazon CloudWatch ツールを使用したデプロイのモニタリング 555
CloudWatch アラームを使用したデプロイのモニタリング 555
Amazon CloudWatch Events を使用したデプロイのモニタリング 557
を使用したデプロイのモニタリング AWS CloudTrail 560
CloudTrail での CodeDeploy 情報 560
CodeDeploy ログファイルエントリの理解 561
Amazon SNS イベント通知を使用したデプロイのモニタリング 562
サービスロールへの Amazon SNS アクセス許可の付与 564
CodeDeploy イベントのトリガーを作成 565
デプロイグループのトリガーの編集 572
デプロイグループからトリガーを削除 574
トリガーの JSON データの形式 575
セキュリティ 578
データ保護 578
ネットワーク間のトラフィックのプライバシー 579
保管中の暗号化 580
転送中の暗号化 580
暗号化キーの管理 581
ID とアクセス管理 581
オーディエンス 581
アイデンティティを使用した認証 582
ポリシーを使用したアクセス管理 583
AWS CodeDeploy が IAM と連携する方法 584
AWS CodeDeploy の マネージド (事前定義) ポリシー 589
AWS マネージドポリシーへの CodeDeploy の更新 598
アイデンティティベースのポリシーの例 602
トラブルシューティング 609
CodeDeploy アクセス許可 リファレンス 611

サービス間での不分別な代理処理の防止 620
インシデント対応 622
CodeDeploy とのすべてのインタラクションの監査 622
アラートと障害管理 622
コンプライアンス検証 623
耐障害性 624
インフラストラクチャセキュリティ 624
リファレンス 625
AppSpec ファイルのリファレンス 625
Amazon ECS コンピューティングプラットフォームの AppSpec ファイル 626
AWS Lambda コンピューティングプラットフォーム上の AppSpec ファイル 626
EC2/オンプレミスコンピューティングプラットフォームの AppSpec ファイル 626
AppSpec ファイル構造 627
AppSpec ファイルの例 673
AppSpec ファイルの間隔 679
AppSpec ファイルの検証とファイルの場所 680
エージェント設定リファレンス 681
関連トピック 693
CloudFormation テンプレートリファレンス 694
Amazon Virtual Private Cloud で CodeDeploy を使用 697
可用性 698
CodeDeploy 用の VPC エンドポイントを作成する 701
CodeDeploy エージェントと IAM 許可を設定する 702
リソース キットのリファレンス 703
リージョン別リソースキットバケット名 703
リソースキットの内容 706
リソースキットのファイルのリストの表示 708
リソースキットのファイルのダウンロード 710
クォータ 713
トラブルシューティング 720
一般的なトラブルシューティングの問題 720
一般的なトラブルシューティングのチェックリスト 721
CodeDeploy デプロイリソースは一部の AWS リージョンでのみサポートされています 722
このガイドの手順が CodeDeploy コンソールと一致しない 723
必要な IAM ロールを取得できない 723

何らかのテキストエディタを使用して AppSpec ファイルとシェルスクリプトを作成すると、デプロイが失敗する場合がある 724
macOS の Finder を使用してアプリケーションリビジョンをバンドルすると、デプロイが失敗することがある 725
EC2/オンプレミスのデプロイに関する問題のトラブルシューティング 725
CodeDeploy プラグイン CommandPoller の認証情報不足エラー 726
デプロイが失敗し、PKCS7 署名付きメッセージの検証に失敗しました」というメッセージが表示される 727
同じデプロイ先のインスタンスに対する同じファイルのデプロイや再デプロイは失敗し、「指定したファイルはこの場所に既に存在するため、デプロイに失敗しました」というエラーが返されます。 727
ファイルパスが長いと、「そのようなファイルまたはディレクトリはありません」というエラーが発生します 729
長時間実行されているプロセスにより、デプロイが失敗することがある 730
AllowTraffic ライフサイクルイベントが失敗し、デプロイログにエラーが報告されない場合のトラブルシューティング 732
失敗した ApplicationStop、BeforeBlockTraffic、または AfterBlockTraffic デプロイライフサイクルイベントのトラブルシューティング 733
「不明なエラー: 読み取り用に開いていません」で失敗した DownloadBundle デプロイライフサイクルイベントのトラブルシューティング 734
スキップされたすべてのライフサイクルイベントのエラーをトラブルシューティングする 735
Windows PowerShell スクリプトで、デフォルトで 64 ビットバージョンの Windows PowerShell スクリプトを使用できない 737
Amazon ECS のデプロイに関する問題のトラブルシューティング 738
置き換えタスクセットを待っている間にタイムアウトが発生します 739
通知が継続されるのを待っている間のタイムアウトの発生 739
IAM ロールに十分なアクセス許可がありません 740
ステータスコールバックを待っている間に、デプロイがタイムアウトした 740
1 つ以上のライフサイクルイベントの検証機能が失敗したため、デプロイが失敗しました . 741
「プライマリタスクセットのターゲットグループはリスナーの後ろにある必要があります」というエラーのため、ELB を更新できませんでした 741
Auto Scaling を使用するとデプロイが失敗することがあります 742
ALB のみが段階的なトラフィックルーティングをサポートしているため、デプロイグループの作成/更新時には、代わりに AllAtOnce トラフィックルーティングを使用してください 743

デプロイが成功しても、置き換えタスクセットは Elastic Load Balancing のヘルスチェックに失敗し、アプリケーションがダウンしています 744
1 つのデプロイグループに複数のロードバランサーをアタッチできますか? 745
ロードバランサーなしで CodeDeploy のブルー/グリーンデプロイを実行できますか? 745
デプロイ中に Amazon ECS サービスを新しい情報で更新する方法を教えてください。 745
AWS Lambda デプロイの問題をトラブルシューティングする 746
AWS Lambda ロールバックが設定されていない Lambda デプロイを手動で停止すると、デプロイが失敗する 746
デプロイグループの問題のトラブルシューティング 747
デプロイグループの一部としてインスタンスにタグを付けても、アプリケーションが自動的に新しいインスタンスにデプロイされない 747
インスタンスの問題のトラブルシューティング 747
タグは正しく設定する必要がある 748
AWS CodeDeploy エージェントはインスタンスにインストールして実行する必要があります 748
デプロイ中にインスタンスを削除した場合、デプロイは最大 1 時間は失敗しません。 748
ログファイルの分析によるインスタンスでのデプロイの失敗の調査 749
誤って削除した場合は、新しい CodeDeploy ログファイルを作成します。 749
InvalidSignatureException – Signature expired: [time] is now earlier than [time]」デプロイエラーのトラブルシューティング 749
GitHub トークンの問題のトラブルシューティング 750
GitHub OAuth トークンが無効です 750
GitHub OAuth トークンの最大数を超えました 750
Amazon EC2 Auto Scaling の問題のトラブルシューティング 751
Amazon EC2 Auto Scaling の一般的なトラブルシューティング 751
CodeDeployRole は、次の AWS サービスでオペレーションを実行するアクセス許可を付与しません: AmazonAutoScaling」エラー 752
Amazon EC2 Auto Scaling グループのインスタンスのプロビジョニングと終了が繰り返されてリビジョンをデプロイできない 753
Amazon EC2 Auto Scaling インスタンスを終了または再起動すると、デプロイが失敗する場合がある 754
複数のデプロイグループを 1 つの Amazon EC2 Auto Scaling グループに関連付けることは避ける 755
Amazon EC2 Auto Scaling グループの EC2 インスタンスが起動に失敗し、「ハートビートのタイムアウト」というエラーが表示される 755

Amazon EC2 Auto Scaling ライフサイクルフックの不一致により、Amazon EC2 Auto Scaling グループへの自動デプロイが停止または失敗することがある 758
「デプロイグループのインスタンスが見つからないため、デプロイに失敗しました」というエラー 759
エラーコード 768
関連トピック 773
リソース 774
リファレンスガイドとサポートリソース 774
サンプル 774
ブログ 774
AWS ソフトウェア開発キットとツール 774
ドキュメント履歴 776
以前の更新 796
AWS 用語集 819
..... dcccxx

CodeDeploy とは

CodeDeploy は、Amazon EC2 インスタンスやオンプレミスインスタンス、サーバーレス Lambda 関数、または Amazon ECS サービスに対するアプリケーションのデプロイを自動化するデプロイメントサービスです。

以下のような、ほぼ無制限の多様なアプリケーションコンテンツをデプロイできます。

- コード
- サーバーレス AWS Lambda 関数
- ウェブファイルおよび設定ファイル
- Executables
- パッケージ
- スクリプト
- マルチメディアファイル

CodeDeploy では、サーバーで実行され、Amazon S3 バケット、GitHub リポジトリ、または Bitbucket リポジトリに保存されているアプリケーションコンテンツをデプロイできます。CodeDeploy では、サーバーレス Lambda 関数をデプロイすることもできます。既存のコードを変更することなく CodeDeploy を使用できます。

CodeDeploy を使用すると、以下を容易に行うことができます。

- 新機能の迅速なリリース。
- AWS Lambda 関数のバージョンを更新します。
- アプリケーションのデプロイメント中のダウンタイム回避。
- アプリケーションの更新に伴う繁雑さを処理。エラーの発生しやすい手動デプロイに伴うリスクの多くを回避できます。

サービスはインフラストラクチャに合わせてスケールするため、1 つのインスタンスまたは数千のインスタンスに簡単にデプロイできます。

CodeDeploy は、設定管理、ソース管理、継続的統合、継続的配信、および継続的デプロイのための様々なシステムで動作します。詳細については、「製品の統合」を参照してください。

また、CodeDeploy コンソールでは、リポジトリ、ビルドプロジェクト、デプロイアプリケーション、パイプラインなどのリソースをすばやく検索することもできます。[Go to resource (リソースに移動)] または / キーを押して、リソースの名前を入力します。一致するものはすべてリストに表示されます。検索では大文字と小文字が区別されません。リソースを表示する権限がある場合のみ表示されます。詳細については、「の ID とアクセスの管理 AWS CodeDeploy」を参照してください。

トピック

- の利点 AWS CodeDeploy
- CodeDeploy コンピューティングプラットフォームの概要
- CodeDeploy デプロイタイプの概要
- ご意見をお待ちしております
- CodeDeploy プライマリコンポーネント
- CodeDeploy デプロイ
- CodeDeploy アプリケーション仕様 (AppSpec) ファイル

の利点 AWS CodeDeploy

CodeDeploy には、次の利点があります。

- サーバー、サーバーレス、コンテナアプリケーション。CodeDeploy では、従来のアプリケーションと、サーバーレス AWS Lambda 関数バージョンまたは Amazon ECS アプリケーションをデプロイするアプリケーションの両方をデプロイできます。
- 自動デプロイ CodeDeploy は、開発、テスト、本番稼働環境にまたがるアプリケーションのデプロイを完全に自動化します。CodeDeployは、インフラストラクチャに合わせてスケールするため、1 つのインスタンスまたは数千のインスタンスにデプロイできます。
- ダウンタイムの最小化。アプリケーションで EC2/オンプレミスコンピューティングプラットフォームを使用している場合、CodeDeploy は、アプリケーションの可用性を最大化します。インプレースデプロイの場合、CodeDeploy は複数の Amazon EC2 インスタンスにまたがってローリング更新を実行します。更新のために、一度にオフラインにするインスタンスの数を指定できます。Blue/Green デプロイの間、最新アプリケーションのリビジョンは、置き換え先インスタンスにインストールされます。トラフィックは、選択するとすぐに、または新しい環境のテストが完了した時点で、これらのインスタンスに再ルーティングされます。どちらのデプロイタイプでも、CodeDeploy は設定したルールに従ってアプリケーションの状態を追跡します。
- 停止してロールバック。エラーが発生した場合、自動または手動でデプロイを停止してロールバックできます。

- コントロールの一元化。CodeDeploy コンソールまたは AWS CLIでデプロイを起動し、そのステータスを追跡できます。各アプリケーションのリビジョンがいつデプロイされ、どの Amazon EC2 インスタンスがリストされているかを示すレポートを受け取ります。
- 使い方が簡単。CodeDeploy は、プラットフォームに依存せず、すべてのアプリケーションで動作します。セットアップコードは簡単に再利用できます。セットアップコードを簡単に再利用できます。また、CodeDeploy はソフトウェアリリースプロセスまたは継続的な配信ツールチェーンと統合できます。
- 同時デプロイ。EC2/オンプレミスコンピューティングプラットフォームを使用する複数のアプリケーションがある場合、CodeDeploy は同じインスタンスのセットでこれらを同時にデプロイできます。

CodeDeploy コンピューティングプラットフォームの概要

CodeDeploy では、アプリケーションを 3 つのコンピューティングプラットフォームにデプロイできます。

- EC2/オンプレミス: Amazon EC2 クラウドインスタンス、オンプレミスサーバー、またはその両方とすることができる物理サーバーのインスタンスを記述します。EC2/オンプレミスコンピューティングプラットフォームを使用して作成されたアプリケーションは、実行可能ファイル、設定ファイル、イメージなどで構成できます。

  EC2/オンプレミス コンピューティングプラットフォームを使用するデプロイでは、インプレイスまたは Blue/Green デプロイタイプを使用して、トラフィックをインスタンスに振り分ける方法を管理できます。詳細については、「CodeDeploy デプロイタイプの概要」を参照してください。
- AWS Lambda: 更新されたバージョンの Lambda 関数で構成されるアプリケーションをデプロイするために使用されます。 は、高可用性コンピューティング構造で構成されるサーバーレスコンピューティング環境で Lambda 関数 AWS Lambda を管理します。コンピューティングリソースの管理はすべて、 によって実行されます AWS Lambda。詳細については、「サーバーレスコンピューティングとアプリケーション」を参照してください。 AWS Lambda および Lambda 関数の詳細については、「」を参照してくださいAWS Lambda。

  Canary 設定、線形設定、または一度にすべての設定を選択して、デプロイ時に更新済み Lambda 関数バージョンにトラフィックを移行する方法を管理できます。
- Amazon ECS：Amazon ECS にコンテナ化されたアプリケーションをタスクセットとしてデプロイするために使用します。CodeDeploy は、アプリケーションの更新バージョンを新しい置き換えタスクセットとしてインストールすることで、blue/Green のデプロイを実行します。CodeDeploy

は、元のアプリケーションタスクセットからの本稼働トラフィックを置き換えタスクセットに再ルーティングします。デプロイが正常に完了すると、元のタスクセットは終了します。Amazon ECS の詳細については、「Amazon Elastic Container Service」を参照してください。

Canary 設定、線形設定、または一度にすべての設定を選択して、デプロイ時に更新済みタスクセットにトラフィックを移行する方法を管理できます。

> **Note**
>
> Amazon ECS blue/Green デプロイは、CodeDeploy と CloudFormationの両方でサポートされています。これらのデプロイの詳細については、以降のセクションで説明します。

次の表に、CodeDeploy コンポーネントが各コンピューティングプラットフォームで使用される様子を示します。詳細については、以下を参照してください。

- CodeDeploy でのデプロイグループの使用
- CodeDeploy でのデプロイグループの使用
- CodeDeploy でデプロイ設定を使用する
- CodeDeploy のアプリケーションリビジョンの操作
- CodeDeploy 内でアプリケーションを用いて作業をする

CodeDeploy コンポーネント	EC2/オンプレミス	AWS Lambda	Amazon ECS
デプロイグループ	一連のインスタンスにリビジョンをデプロイします。	可用性の高いコンピューティングインフラストラクチャで、サーバーレス Lambda 関数の新しいバージョンをデプロイします。	タスクセットとしてデプロイする、コンテナ化されたアプリケーション、デプロイされたアプリケーションへの

CodeDeploy コンポーネント	EC2/オンプレミス	AWS Lambda	Amazon ECS
			トラフィック提供に使用される本稼働およびオプションのテストリスナー、トラフィックを再ルーティングし、デプロイされたアプリケーションの元のタスクセットを終了するタイミング、オプションのトリガー、アラーム、およびロールバック設定で Amazon ECS サービスを指定します。

CodeDeploy コンポーネント	EC2/オンプレミス	AWS Lambda	Amazon ECS
デプロイメント	アプリケーションと AppSpec ファイルから構成される新しいリビジョンをデプロイします。AppSpec は、アプリケーションをデプロイグループのインスタンスにデプロイする方法を指定します。	Lambda 関数の 1 つのバージョンから、同じ関数の新しいバージョンに本稼働トラフィックを移行させます。AppSpec は、デプロイする Lambda 関数のバージョンを指定します。	Amazon ECS コンテナ化されたアプリケーションの更新バージョンを、新しい置き換えタスクセットとしてデプロイします。CodeDeploy は、タスクセットからの本稼働トラフィックを置き換えタスクセットに再ルーティングします。デプロイ完了時に、元のタスクセットは削除されます。

CodeDeploy コンポーネント	EC2/オンプレミス	AWS Lambda	Amazon ECS
Deployment configuration	デプロイの速度と、デプロイ中にいつでも使用できる必要のある正常なインスタンスの最小数を決定する設定。	更新された Lambda 関数バージョンにトラフィックを移行する方法を決定する設定。	更新された Amazon ECS タスクセットにトラフィックを移行する方法を決定する設定。

CodeDeploy コンポーネント	EC2/オンプレミス	AWS Lambda	Amazon ECS
リビジョン	AppSpec ファイルとアプリケーションファイルの組み合わせ (例: 実行ファイル、設定ファイル)。	AppSpec ファイルは Lambda 関数と、ライフサイクルイベントフック中に検証テストを実行可能な Lambda 関数のデプロイを指定します。	AppSpec ファイル以下を指定する。 • デプロイするためのコンテナー化されたアプリケーションを含む、Amazon ECS サービスの Amazon ECS タスク定義。 • 最新のアプリケーションがデプロイされているコンテナ。 • 本稼働トラフィックが再ルーティングされ

CodeDeploy コンポーネント	EC2/オンプレミス	AWS Lambda	Amazon ECS
			るコンテナのポート。 • オプションのネットワーク構成設定と Lambda 関数は、デプロイライフサイクルイベントフック中に検証テストを実行できる。

CodeDeploy コンポーネント	EC2/オンプレミス	AWS Lambda	Amazon ECS
アプリケーション	デプロイグループおよびリビジョンのコレクション。EC2/オンプレミスアプリケーションは、EC2/オンプレミスコンピューティングプラットフォームを使用します。	デプロイグループおよびリビジョンのコレクション。 AWS Lambda デプロイに使用されるアプリケーションは、サーバーレス Lambda AWS コンピューティングプラットフォームを使用します。	デプロイグループおよびリビジョンのコレクション。Amazon ECS デプロイに使用されるアプリケーションは、Amazon ECS コンピューティングプラットフォームを使用します。

CodeDeploy デプロイタイプの概要

CodeDeploy には、2 つのデプロイタイプのオプションがあります。

- インプレイスデプロイ: デプロイグループの各インスタンス上のアプリケーションが停止され、最新のアプリケーションリビジョンがインストールされて、新バージョンのアプリケーションが開始され検証されます。ロードバランサーを使用し、デプロイ中はインスタンスが登録解除され、デプロイ完了後にサービスに復元されるようにできます。EC2 オンプレミスコンピューティングプラットフォームを使用するデプロイのみが、インプレイスデプロイを使用できます。インプレイスデプロイの詳細については、「インプレースデプロイの概要」を参照してください。

> **Note**
>
> AWS Lambda および Amazon ECS デプロイでは、インプレースデプロイタイプを使用できません。

- Blue/Green デプロイ: デプロイの動作は、使用するコンピューティングプラットフォームにより異なります。
  - EC2 オンプレミスコンピューティングプラットフォームの Blue/Green: 以下のステップを使用して、デプロイグループのインスタンス (元の環境) がインスタンスの別のセット (置き換え先環境) に置き換えられます。
    - 置き換え先の環境のインスタンスがプロビジョニングされます。
    - 最新のアプリケーションリビジョンは、置き換え先インスタンスにインストールされます。
    - オプションの待機時間は、アプリケーションのテストやシステム検証などのアクティビティに対して発生します。
    - 置き換え先環境のインスタンスは、1 つまたは複数の Elastic Load Balancing ロードバランサーに登録され、トラフィックは、それらに再ルーティングされます。元の環境のインスタンスは、登録が解除され、終了するか、他の使用のために実行することができます。

    > **Note**
    >
    > EC2/オンプレミスのコンピューティングプラットフォームを使用する場合は、blue/green デプロイが Amazon EC2 インスタンスでのみ機能することに注意してください。

  - AWS Lambda または Amazon ECS コンピューティングプラットフォームの Blue/Green: トラフィックは、Canary、線形、または all-at-onceのデプロイ設定に従って増分でシフトされます。
  - 経由のブルー/グリーンデプロイ CloudFormation: スタック CloudFormation の更新の一環として、トラフィックは現在のリソースから更新されたリソースに移行されます。現時点では、ECS blue/green デプロイのみがサポートされています。

ブルー/グリーンデプロイの詳細については、「Blue/Green デプロイの概要」を参照してください。

> **Note**
>
> CodeDeploy エージェントを使用すると、アプリケーション、デプロイグループ、または AWS アカウントを必要とせずに、サインインしているインスタンスでデプロイを実行できます。詳細については、「CodeDeploy エージェントを使用してローカルマシン上のデプロイパッケージを検証する」を参照してください。

トピック

- インプレースデプロイの概要
- Blue/Green デプロイの概要

インプレースデプロイの概要

> **Note**
>
> AWS Lambda および Amazon ECS デプロイでは、インプレースデプロイタイプを使用できません。

インプレイスデプロイの仕組みは次のとおりです。

1. 最初に、ローカルの開発用マシンまたは同様の環境でデプロイ可能なコンテンツを作成し、アプリケーション特定ファイル (AppSpecファイル) を追加します。AppSpec ファイルは CodeDeploy に固有です。CodeDeploy で実行するデプロイアクションを定義します。デプロイ可能なコンテンツおよび AppSpec ファイルをアーカイブファイルにバンドルし、Amazon S3 バケットまたは GitHub リポジトリにアップロードします。このアーカイブファイルは、アプリケーションリビジョン (または単にリビジョン) と呼ばれます。
2. 次に、リビジョンを取得する Amazon S3 バケットまたは GitHub リポジトリ、コンテンツをデプロイする Amazon EC2 インスタンスのセットなど、デプロイに関する情報を CodeDeploy に提供します。CodeDeploy は Amazon EC2 インスタンスのセットを デプロイグループ で呼び出します。デプロイグループには、個別にタグ付けされた Amazon EC2 インスタンス、Amazon EC2 Auto Scaling グループ内の Amazon EC2 インスタンス、またはその両方が含まれます。

   デプロイグループにデプロイする新しいアプリケーションリビジョンを正常にアップロードするたびに、そのバンドルはデプロイグループのターゲットリビジョンとして設定されます。つま

り、現在デプロイ対象となっているアプリケーションリビジョンがターゲットリビジョンです。これは、自動デプロイにプルされるリビジョンでもあります。

3. 次に、各インスタンスの CodeDeploy エージェントが CodeDeploy をポーリングして、指定された Amazon S3 バケットまたは GitHub リポジトリから何をいつ取得するかを決定します。
4. 最後に、各インスタンスの CodeDeploy エージェントは、Amazon S3 バケットや GitHub リポジトリからターゲットリビジョンを取得し、AppSpec ファイルの手順を使用して、コンテンツをインスタンスにデプロイします。

CodeDeploy は、デプロイステータス、デプロイ設定パラメータ、インスタンスの状態を取得できるように、デプロイのレコードを保持します。

Blue/Green デプロイの概要

Blue/Green デプロイは、アプリケーションの新しいバージョンの変更による中断を最小限に抑えながら、アプリケーションの更新に使用されます。CodeDeploy は、本番トラフィックを再ルーティングする前に、古いバージョンと一緒に新しいアプリケーションバージョンをプロビジョニングします。

- AWS Lambda: トラフィックは、Lambda 関数の 1 つのバージョンから、同じ Lambda 関数の新しいバージョンに移行されます。
- Amazon ECS: Amazon ECS サービスのタスクセットから、同じ Amazon ECS サービスの最新の置き換えタスクセットにトラフィックが移行します。
- EC2/オンプレミス: 元の環境内の、あるインスタンスセットから、インスタンスの置き換え先のセットにトラフィックが移行します。

すべての AWS Lambda および Amazon ECS デプロイは Blue/Green です。EC2/オンプレミス デプロイは、インプレースまたは Blue/Green です。Blue/Green デプロイには、インプレースデプロイと比べて多くのメリットがあります。

- トラフィックを再ルーティングするだけで、アプリケーションを新しい置き換え先環境にインストールしてテストし、本稼働環境へデプロイできます。
- EC2/オンプレミスコンピューティングプラットフォーム を使用している場合、最新バージョンのアプリケーションに切り替えることで、より迅速になり、信頼性が高まります。これは、トラフィックが終了していない限り、トラフィックを元のインスタンスにルーティングできるためです。インプレースデプロイでは、以前のバージョンのアプリケーションを再デプロイすることによってバージョンをロールバックする必要があります。

- EC2/オンプレミスコンピューティングプラットフォーム を使用している場合、新しいインスタンスは Blue/Green デプロイ向けにプロビジョニングされ、最新のサーバー設定が反映されます。これにより、長時間実行するインスタンスで発生する問題を回避できます。
- AWS Lambda コンピューティングプラットフォームを使用している場合は、トラフィックを元の AWS Lambda 関数バージョンから新しい Lambda AWS 関数バージョンに移行する方法を制御します。
- Amazon ECS コンピューティングプラットフォームを使用している場合は、元のタスクセットから新しいタスクセットにトラフィックを移行する方法を制御します。

を使用した Blue/Green デプロイ CloudFormation では、次のいずれかの方法を使用できます。

- CloudFormation デプロイ用の テンプレート: CloudFormation テンプレートを使用してデプロイを設定すると、デプロイは CloudFormation 更新によってトリガーされます。リソースを変更してテンプレートの変更をアップロードすると、 のスタックの更新によって新しいデプロイ CloudFormation が開始されます。 CloudFormation テンプレートで使用できるリソースのリストについては、「」を参照してくださいCloudFormation CodeDeploy リファレンスの テンプレート。
- によるブルー/グリーンデプロイ CloudFormation: を使用して CloudFormation 、スタックの更新を通じてブルー/グリーンデプロイを管理できます。スタックテンプレート内で、トラフィックルーティングと安定化設定を指定するだけでなく、Blue ソースと Green リソースの両方を定義します。次に、スタックの更新中に選択したリソースを更新すると、 は必要なすべてのグリーンリソース CloudFormation を生成し、指定されたトラフィックルーティングパラメータに基づいてトラフィックをシフトし、ブルーリソースを削除します。詳細については、[CloudFormationユーザーガイド] の [AWS CloudFormation を使用した CodeDeploy による Perform Amazon ECS Blue/Green デプロイの実行] を参照してください。

> **Note**
>
> Amazon ECS Blue/Green デプロイでのみサポートされます。

Blue/Green デプロイを設定する方法は、使用するコンピューティングプラットフォームによって異なります。

AWS Lambda または Amazon ECS コンピューティングプラットフォームでのブルー/グリーンデプロイ

AWS Lambda または Amazon ECS コンピューティングプラットフォームを使用している場合は、トラフィックを元の AWS Lambda 関数または Amazon ECS タスクセットから新しい関数またはタスクセットに移行する方法を指定する必要があります。トラフィックを移行する方法を指定するには、以下のいずれかのデプロイ設定を指定する必要があります。

- canary
- 線形
- 一度にすべて

Canary 設定、線形設定、または一度にすべてのデプロイ設定でトラフィックを移行する方法については、「Deployment configuration」を参照してください。

Lambda デプロイ設定の詳細については、「AWS Lambda コンピューティングプラットフォームでのデプロイ設定」を参照してください。

Amazon ECS デプロイ設定の詳細については、「Amazon ECS コンピューティングプラットフォームのデプロイ設定」を参照してください。

EC2/オンプレミスコンピューティングプラットフォームの Blue/Green デプロイ

> **Note**
>
> EC2/オンプレミスコンピューティングプラットフォームでの Blue/Green デプロイには、Amazon EC2 インスタンスを使用する必要があります。オンプレミスインスタンスは Blue/Green デプロイタイプではサポートされません。

EC2/オンプレミスコンピューティングプラットフォームを使用している場合は、次が適用されます。

Amazon EC2 タグまたは Amazon EC2 Auto Scaling グループを識別する 1 つ以上の Amazon EC2 インスタンスが必要です。インスタンスは、以下の条件を満たす必要があります。

- 各 Amazon EC2 インスタンスには、適切な IAM インスタンスプロファイルが添付されている必要があります。

- 各インスタンスで CodeDeploy エージェントをインストールして実行する必要があります。

> **Note**
>
> 通常は、元の環境のインスタンスでアプリケーションリビジョンも実行しますが、これは Blue/Green デプロイの要件ではありません。

Blue/Green デプロイで使用されるデプロイグループを作成するときは、置き換え先環境の指定方法を選択できます。

既存の Amazon EC2 Auto Scaling グループのコピー: Blue/Green デプロイでは、CodeDeploy は、デプロイ中に置き換え先環境用のインスタンスを作成します。このオプションを使用すると、CodeDeploy は、指定した Amazon EC2 Auto Scaling グループを置き換え先環境のテンプレートとして使用します。これには、同じ数の実行中のインスタンスと他の多くの設定オプションが含まれます。

手動でインスタンスを選択: Amazon EC2 インスタンスタグ、Amazon EC2 Auto Scaling グループ名、またはその両方を使用して、置き換え先として含めるインスタンスを指定できます。このオプションを選択した場合、デプロイを作成するまで置き換え先環境のインスタンスを指定する必要はありません。

処理の流れ

1. 元の環境となるインスタンスや Amazon EC2 Auto Scaling グループは既にあります。Blue/Green デプロイを初めて実行するときは、通常、インプレースデプロイで既に使用されているインスタンスを使用します。
2. 既存の CodeDeploy アプリケーションでは、Blue/Green デプロイグループを作成し、インプレースデプロイに必要なオプションに加えて、次を指定します。
   - ブルー/グリーンデプロイプロセス中に元の環境から置き換え先環境にトラフィックをルーティングするロードバランサー。
   - 置き換え先環境にトラフィックを直ちに再ルーティングするか、手動で再ルーティングするまで待つかどうか。
   - トラフィックが置き換え先インスタンスにルーティングされるレート。
   - 置き換え元インスタンスを削除するか引き続き実行するかどうか。
3. このデプロイグループのデプロイを作成するときには、次の処理が行われます。

a. Amazon EC2 Auto Scaling グループをコピーすることを選択した場合、インスタンスは置き換え先環境にプロビジョニングされます。

b. デプロイに指定したアプリケーションリビジョンは、置き換え先インスタンスにインストールされます。

c. デプロイグループの設定で待機時間を指定した場合、デプロイは一時停止します。これは置き換え先環境のテストおよび確認を実行できる時間です。待機時間が終了する前にトラフィックを手動で再ルーティングしない場合、デプロイは停止します。

d. 置き換え先環境のインスタンスは Elastic Load Balancing ロードバランサーに登録され、これらのインスタンスに対してトラフィックのルーティングが開始されます。

e. 元の環境のインスタンスは、終了するか引き続き実行するか、デプロイグループの指定に従って登録が解除され、処理されます。

によるブルー/グリーンデプロイ CloudFormation

テンプレートを使用してリソースをモデリングすることで、 CloudFormation CodeDeploy のブルー/グリーンデプロイを管理できます。

CloudFormation テンプレートを使用して Blue/Green リソースをモデル化するときは、タスクセット CloudFormation を更新するスタック更新を で作成します。本稼働トラフィックは、すべて一度に、線形デプロイとベイク処理時間を使用して、または Canary デプロイを使用してサービスの元のタスクセットから置き換えタスクセットにシフトします。スタックの更新により、CodeDeploy でデプロイが開始されます。CodeDeploy でデプロイのステータスと履歴を表示できますが、それ以外の場合は CloudFormation 、テンプレートの外部で CodeDeploy リソースを作成または管理しません。

> **Note**
>
> を通じてブルー/グリーンデプロイを行う場合 CloudFormation、CodeDeploy アプリケーションまたはデプロイグループを作成しません。

この方法では、Amazon ECS Blue/Green デプロイのみサポートされます。によるブルー/グリーンデプロイの詳細については CloudFormation、「」を参照してください[を使用して Amazon ECS ブルー/グリーンデプロイを作成する CloudFormation]。

ご意見をお待ちしております

ご意見をお待ちしております。お問い合わせの場合には、CodeDeploy フォーラム をご利用ください。

トピック

- Primary Components
- Deployments
- Application Specification Files

CodeDeploy プライマリコンポーネント

サービスの使用をスタートする前に、CodeDeploy デプロイプロセスの主なコンポーネントを理解しておく必要があります。

トピック

- アプリケーション
- コンピューティングプラットフォーム
- Deployment configuration
- デプロイグループ
- デプロイタイプ
- IAM インスタンスプロファイル
- リビジョン
- サービスロール
- ターゲットリビジョン
- 他のコンポーネント

アプリケーション

アプリケーションは、デプロイしたいアプリケーションを一意に識別する名前です。CodeDeploy はこの名前をコンテナとして使用して、デプロイ中にリビジョン、デプロイ設定、およびデプロイグループの正しい組み合わせが参照されるようにします。

コンピューティングプラットフォーム

コンピューティングプラットフォームは、CodeDeploy がアプリケーションをデプロイするプラットフォームです。コンピューティングプラットフォームは 3 つあります。

- EC2/オンプレミス: Amazon EC2 クラウドインスタンス、オンプレミスサーバー、またはその両方とすることができる物理サーバーのインスタンスを記述します。EC2/オンプレミスコンピューティングプラットフォームを使用して作成されたアプリケーションは、実行可能ファイル、設定ファイル、イメージなどで構成できます。

  EC2/オンプレミス コンピューティングプラットフォームを使用するデプロイでは、インプレイスまたは Blue/Green デプロイタイプを使用して、トラフィックをインスタンスに振り分ける方法を管理できます。詳細については、「CodeDeploy デプロイタイプの概要」を参照してください。

- AWS Lambda: Lambda 関数の更新バージョンで構成されるアプリケーションをデプロイするために使用されます。 は、高可用性コンピューティング構造で構成されるサーバーレスコンピューティング環境で Lambda 関数 AWS Lambda を管理します。コンピューティングリソースの管理はすべて、 によって実行されます AWS Lambda。詳細については、「サーバーレスコンピューティングとアプリケーション」を参照してください。 AWS Lambda および Lambda 関数の詳細については、「」を参照してくださいAWS Lambda。

  Canary 設定、線形設定、または一度にすべての設定を選択して、デプロイ時に更新済み Lambda 関数バージョンにトラフィックを移行する方法を管理できます。

- Amazon ECS：Amazon ECS にコンテナ化されたアプリケーションをタスクセットとしてデプロイするために使用します。CodeDeploy は、アプリケーションの更新バージョンを新しい置き換えタスクセットとしてインストールすることで、blue/Green のデプロイを実行します。CodeDeploy は、元のアプリケーションタスクセットからの本稼働トラフィックを置き換えタスクセットに再ルーティングします。デプロイが正常に完了すると、元のタスクセットは終了します。Amazon ECS の詳細については、「Amazon Elastic Container Service」を参照してください。

  Canary 設定、線形設定、または一度にすべての設定を選択して、デプロイ時に更新済みタスクセットにトラフィックを移行する方法を管理できます。

> **Note**
>
> Amazon ECS Blue/Green デプロイは、CodeDeploy と CloudFormationの両方でサポートされています。これらのデプロイの詳細については、以降のセクションで説明します。

Deployment configuration

デプロイ設定は、デプロイ時に CodeDeploy が使用する一連のデプロイのルール、デプロイの成功条件、デプロイの失敗条件です。デプロイで EC2/オンプレミスコンピューティングプラットフォームを使用している場合、デプロイの正常なインスタンスの最小数を指定できます。デプロイで AWS Lambda または Amazon ECS コンピューティングプラットフォームを使用している場合は、更新された Lambda 関数または ECS タスクセットにトラフィックをルーティングする方法を指定できます。

EC2/オンプレミスコンピューティングプラットフォームを使用したデプロイに対する正常なホストの最小数の指定については、「正常なインスタンスの最小数について」を参照してください。

以下のデプロイ設定では、Lambda または ECS コンピューティングプラットフォームを使用するデプロイの間にトラフィックをルーティングする方法を指定します。

- Canary: トラフィックは 2 つの増分で移行されます。事前定義された canary オプションから選択できます。これらのオプションでは、更新された Lambda 関数または ECS タスクセットに、2 回目の増分で移行される前に最初の増分および間隔 (分単位) で移行される、トラフィックの割合 (%) が指定されています。
- Linear: トラフィックは等しい増分で移行され、増分間の間隔 (分) も同じです。増分ごとに移行するトラフィックの割合 (%) と、増分間の間隔 (分) を指定する、事前定義済み線形オプションから選択できます。
- 一度にすべて: すべてのトラフィックを元の Lambda 関数または ECS タスクセットから更新された関数またはタスクセットに同時に移行します。

デプロイグループ

デプロイグループは、個々のインスタンスのセットです。デプロイグループには、個別にタグ付けされた Amazon EC2 インスタンス、Amazon EC2 Auto Scaling グループ内の Amazon EC2 インスタンス、またはその両方が含まれます。Amazon EC2 インスタンスタグの詳細については、「コンソールでのタグの操作」を参照してください。オンプレミスインスタンスの詳細については、「Working with On-Premises Instances」を参照してください。Amazon EC2 Auto Scaling の情報に関しては、「CodeDeploy と Amazon EC2 Auto Scaling の統合」を参照してください。

デプロイタイプ

デプロイタイプは、デプロイグループ内のインスタンスで最新のアプリケーションリビジョンを使用できるようにするために使用される方法です。次の 2 種類のデプロイタイプがあります。

- インプレイスデプロイ: デプロイグループの各インスタンス上のアプリケーションが停止され、最新のアプリケーションリビジョンがインストールされて、新バージョンのアプリケーションが開始され検証されます。ロードバランサーを使用し、デプロイ中はインスタンスが登録解除され、デプロイ完了後にサービスに復元されるようにできます。EC2 オンプレミスコンピューティングプラットフォームを使用するデプロイのみが、インプレイスデプロイを使用できます。インプレイスデプロイの詳細については、「インプレースデプロイの概要」を参照してください。
- Blue/Green デプロイ: デプロイの動作は、使用するコンピューティングプラットフォームにより異なります。
  - EC2 オンプレミスコンピューティングプラットフォームの Blue/Green: 以下のステップを使用して、デプロイグループのインスタンス (元の環境) がインスタンスの別のセット (置き換え先環境) に置き換えられます。
    - 置き換え先の環境のインスタンスがプロビジョニングされます。
    - 最新のアプリケーションリビジョンは、置き換え先インスタンスにインストールされます。
    - オプションの待機時間は、アプリケーションのテストやシステム検証などのアクティビティに対して発生します。
    - 置き換え先環境のインスタンスは、1 つまたは複数の Elastic Load Balancing ロードバランサーに登録され、トラフィックは、それらに再ルーティングされます。元の環境のインスタンスは、登録が解除され、終了するか、他の使用のために実行することができます。

    > **Note**
    >
    > EC2/オンプレミスのコンピューティングプラットフォームを使用する場合は、blue/green デプロイが Amazon EC2 インスタンスでのみ機能することに注意してください。

  - AWS Lambda または Amazon ECS コンピューティングプラットフォームの Blue/Green: トラフィックは、Canary、線形、または all-at-onceのデプロイ設定に従って増分でシフトされます。
  - 経由のブルー/グリーンデプロイ CloudFormation: スタック CloudFormation の更新の一環として、トラフィックは現在のリソースから更新されたリソースに移行されます。現時点では、ECS blue/green デプロイのみがサポートされています。

  ブルー/グリーンデプロイの詳細については、「Blue/Green デプロイの概要」を参照してください。

> **Note**
>
> Amazon ECS blue/Green デプロイは、CodeDeploy と CloudFormationの両方でサポートされています。これらのデプロイの詳細については、以降のセクションで説明します。

IAM インスタンスプロファイル

IAM インスタンスプロファイルは、Amazon EC2 インスタンス にアタッチする IAM ロールです。このプロファイルには、アプリケーションが保存される Amazon S3 バケットまたは GitHub リポジトリへのアクセスに必要な権限が含まれます。詳細については、「ステップ 4: Amazon EC2 インスタンス用の IAM インスタンスプロファイルを作成する」を参照してください。

リビジョン

リビジョンは、アプリケーションのバージョンです。 AWS Lambda デプロイリビジョンは、デプロイする Lambda 関数に関する情報を指定する YAML 形式または JSON 形式のファイルです。EC2/オンプレミスデプロイリビジョンは、ソースコンテンツ (ソースコード、ウェブページ、実行可能ファイル、デプロイスクリプト) とアプリケーション仕様ファイル (AppSpec ファイル) を含むアーカイブファイルです。 AWS Lambda リビジョンは Amazon S3 バケットに保存できます。EC2/オンプレミスのリビジョンは Amazon S3 バケットまたは GitHub リポジトリに格納されます。Amazon S3 では、リビジョンの Amazon S3 オブジェクトキー、ETag、バージョン、またはその両方により、リビジョンが一意に識別されます。GitHub では、コミット ID により、リビジョンが一意に識別されます。

サービスロール

サービスロールは、 AWS リソースにアクセスできるように AWS サービスにアクセス許可を付与する IAM ロールです。サービスロールにアタッチするポリシーによって、サービスがアクセスできる AWS リソースと、それらのリソースで実行できるアクションが決まります。CodeDeploy では、サービスロールは、以下の目的で使用されます。

- インスタンスに適用されているタグやインスタンスに関連付けられている Amazon EC2 Auto Scaling グループ名を読み取ることができます。これにより、CodeDeploy はアプリケーションをデプロイできるインスタンスを識別できます。
- インスタンス、Amazon EC2 Auto Scaling グループ、および Elastic Load Balancing ロードバランサーでオペレーションを実行するには。

- 指定したデプロイまたはインスタンスイベントが発生したときに通知を送信できるように、Amazon SNS トピックに情報を公開すること。
- CloudWatch アラームに関する情報を取得して、デプロイのアラームモニタリングを設定します。

詳細については、「ステップ 2: CodeDeployのサービスのロールを作成する」を参照してください。

ターゲットリビジョン

ターゲットリビジョンは、リポジトリにアップロードし、デプロイグループ内のインスタンスにデプロイしたいアプリケーションリビジョンの最新バージョンです。つまり、現在デプロイの対象としているアプリケーションリビジョン。これは、自動デプロイにプルされるリビジョンでもあります。

他のコンポーネント

CodeDeploy ワークフローの他のコンポーネントの詳細については、次のトピックを参照してください。

- CodeDeploy リポジトリタイプを選択する
- Deployments
- Application Specification Files
- Instance Health
- CodeDeploy エージェントの使用
- Working with On-Premises Instances

CodeDeploy デプロイ

このトピックでは、CodeDeploy のデプロイのコンポーネントおよびワークフローについて説明します。デプロイプロセスは、デプロイに使用するコンピューティングプラットフォームまたはデプロイ方法 (Lambda、Amazon ECS、EC2/オンプレミス、または 経由 AWS CloudFormation) によって異なります。

トピック

- AWS Lambda コンピューティングプラットフォームでのデプロイ
- Amazon ECS コンピューティングプラットフォームのデプロイ
- EC2/オンプレミスコンピューティングプラットフォームの Blue/Green デプロイ

AWS Lambda コンピューティングプラットフォームでのデプロイ

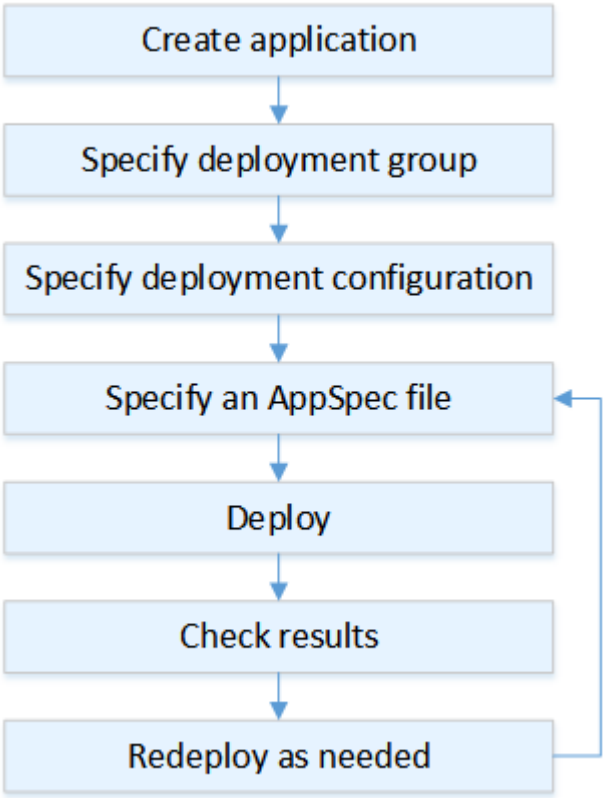
このトピックでは、 AWS Lambda コンピューティングプラットフォームを使用する CodeDeploy デプロイのコンポーネントとワークフローについて説明します。

トピック

- AWS Lambda コンピューティングプラットフォームでのデプロイワークフロー
- アプリケーションリビジョンのアップロード
- アプリケーションとデプロイグループの作成
- アプリケーションリビジョンのデプロイ
- アプリケーションの更新
- 停止、失敗したデプロイ
- デプロイと再デプロイのロールバック

AWS Lambda コンピューティングプラットフォームでのデプロイワークフロー

次の図は、新規および更新された AWS Lambda 関数のデプロイの主要なステップを示しています。



ステップには以下が含まれます。

1. アプリケーションを作成し、デプロイするアプリケーションリビジョンを一意に識別する名前を付けます。Lambda 関数をデプロイするには、 AWS Lambda コンピューティングプラットフォームは、アプリケーションを作成するときに使用します。CodeDeploy は、この名前を使用して、デプロイグループ、デプロイ設定、アプリケーションリビジョンなど、正しいデプロイコンポーネントを参照していることを確認します。詳細については、「CodeDeploy でアプリケーションを作成する」を参照してください。
2. デプロイグループを設定するには、デプロイグループの名前を指定します。
3. デプロイ設定を選択して、トラフィックを元の AWS Lambda 関数バージョンから新しい Lambda 関数バージョンに移行する方法を指定します。詳細については、「View Deployment Configuration Details」を参照してください。
4. アプリケーション仕様ファイル(AppSpec ファイル) を Amazon S3 にアップロードします。AppSpec ファイルは、デプロイを検証するために使用される Lambda 関数のバージョンと Lambda 関数を指定します。AppSpec ファイルを作成しない場合は、YAML または JSON を使用して、コンソールで直接 Lambda 関数バージョンと Lambda デプロイ検証用の関数を指定できます。詳細については、「CodeDeploy のアプリケーションリビジョンの操作」を参照してください。
5. アプリケーションリビジョンをデプロイグループにデプロイします。 は、指定した Lambda 関数リビジョンを AWS CodeDeploy デプロイします。トラフィックを Lambda 関数リビジョンに移行するには、アプリケーション作成時に選択した AppSpec ファイルのデプロイを使用します。詳細については、「CodeDeploy でデプロイを作成する」を参照してください。
6. デプロイの結果を確認します。詳細については、「CodeDeploy でのデプロイモニタリング」を参照してください。

アプリケーションリビジョンのアップロード

Amazon S3 に AppSpec ファイルを配置するか、コンソールまたは AWS CLIに直接入力します。詳細については、「Application Specification Files」を参照してください。

アプリケーションとデプロイグループの作成

AWS Lambda コンピューティングプラットフォーム上の CodeDeploy デプロイグループは、1 つ以上の AppSpec ファイルのコレクションを識別します。AppSpec ファイルごとに 1 つの Lambda 関数バージョンをデプロイできます。デプロイグループは、今後のデプロイのために、アラームおよびロールバックの設定などの設定オプションのセットを定義します。

アプリケーションリビジョンのデプロイ

これで、AppSpec ファイル で指定した関数リビジョンをデプロイグループにデプロイする準備ができました。CodeDeploy コンソールまたは create-deployment コマンドを使用できます。デプロイを制御するために指定できるパラメータ (リビジョン、デプロイグループ、デプロイ設定など) があります。

アプリケーションの更新

アプリケーションを更新し、CodeDeploy コンソールを使用するか、create-deployment コマンドを呼び出してリビジョンをプッシュできます。

停止、失敗したデプロイ

デプロイを停止するには、CodeDeploy コンソールまたはstop-deploymentコマンドを使用できます。デプロイを停止しようとする場合、次の 3 つのうち 1 つのことが発生します。

- デプロイは停止し、オペレーションは成功というステータスを返す。この場合、停止したデプロイに対してそれ以上デプロイライフサイクルイベントは実行されません。
- デプロイは即時に停止せず、オペレーションは保留中というステータスを返す。この場合、一部のデプロイライフサイクルイベントは、デプロイグループでまだ実行中である可能性があります。保留中のオペレーションが完了すると、デプロイを停止するためのそれ以降の呼び出しは、成功というステータスを返します。
- デプロイは停止できず、オペレーションはエラーを返す。詳細については、 AWS CodeDeploy API リファレンスのErrorInformation」と「一般的なエラー」を参照してください。

失敗したデプロイでは、停止されたデプロイのように、一部のデプロイライフサイクルイベントが実行済みになる場合があります。デプロイが失敗した理由を調べるには、CodeDeploy コンソールを使用するか、失敗したデプロイのログファイルデータを分析することができます。詳細については、「アプリケーションリビジョンとログファイルのクリーンアップ」および「CodeDeploy EC2/オンプレミスデプロイのログデータの表示」を参照してください。

デプロイと再デプロイのロールバック

CodeDeploy は新しいデプロイとして、以前にデプロイされたリビジョンを再デプロイすることによって、ロールバックを実装します。

デプロイが失敗した、アラームのモニタリングしきい値に一致したなど、特定の条件が満たされた場合に、自動的にデプロイをロールバックするようグループデプロイを設定できます。個別のデプロイで、デプロイグループに指定されたロールバック設定をオーバーライドすることもできます。

以前のデプロイされたバージョンを手動で再デプロイして、失敗したデプロイをロールバックすることもできます。

いずれの場合でも、新しいデプロイまたはロールバックされたデプロイには独自のデプロイ ID が割り当てられます。CodeDeploy コンソールで表示できるデプロイの一覧には、どれが自動デプロイの結果であるかが示されます。

詳細については、「CodeDeploy を使用した再デプロイおよびデプロイのロールバック」を参照してください。

Amazon ECS コンピューティングプラットフォームのデプロイ

このトピックでは、Amazon ECS コンピューティングプラットフォームを使用する CodeDeploy のデプロイのコンポーネントおよびワークフローについて説明します。

トピック

- Amazon ECS デプロイを開始する前に
- Amazon ECS コンピューティングプラットフォームのデプロイワークフロー (高レベル)
- Amazon ECS デプロイ中の処理で起こっていること
- アプリケーションリビジョンのアップロード
- アプリケーションとデプロイグループの作成
- アプリケーションリビジョンのデプロイ
- アプリケーションの更新
- 停止、失敗したデプロイ
- デプロイと再デプロイのロールバック
- による Amazon ECS ブルー/グリーンデプロイ AWS CloudFormation

Amazon ECS デプロイを開始する前に

Amazon ECS アプリケーションのデプロイを開始する前に、次の準備が完了している必要があります。デプロイグループを作成するときに指定される要件と、AppSpec ファイルで指定される要件があります。

要件	指定される場所
Amazon ECS クラスター	デプロイグループ
Amazon ECS サービス	デプロイグループ
Application Load Balancer および Network Load Balancer	デプロイグループ
本稼働リスナー	デプロイグループ
テストリスナー (オプション)	デプロイグループ
2 つのターゲットグループ	デプロイグループ
Amazon ECS タスク定義	AppSpec ファイル
コンテナ名	AppSpec ファイル
コンテナポート	AppSpec ファイル

Amazon ECS クラスター

Amazon ECSクラスターは、タスクまたはサービスの論理グループです。CodeDeploy アプリケーションのデプロイグループを作成するときに、Amazon ECS サービスを含む Amazon ECS クラスターを指定します。詳細については、「Amazon Elastic Container Service ユーザーガイド」の「Amazon ECS のクラスター」を参照してください。

Amazon ECS サービス

Amazon ECS サービスは、Amazon ECS クラスター内のタスク定義で指定されたインスタンスを維持し、実行します。Amazon ECS サービスが CodeDeploy で有効になっている必要があります。デフォルトでは、Amazon ECS サービスは、Amazon ECS デプロイで有効になっています。デプロイグループを作成するときは、Amazon ECS クラスター内の Amazon ECS サービスをデプロイすることを選択します。詳細については、「Amazon Elastic Container Service ユーザーガイド」の「Amazon ECS のクラスター」を参照してください。

Application Load Balancer および Network Load Balancer

Elastic Load Balancing は、Amazon ECS デプロイで更新する Amazon ECS サービスで使用する必要があります。Application Load Balancer または Network Load Balancer を作成できます。

動的ポートマッピング、パスベースのルーティング、優先ルールなどの機能を利用できるように、 Application Load Balancer をお勧めします。CodeDeploy アプリケーションのデプロイグループを作成するときに、ロードバランサーを指定します。詳細については、「Amazon Elastic Container Service ユーザーガイド」の CodeDeploy Amazon ECS デプロイ用のロードバランサー、ターゲットグループ、リスナーをセットアップする と「ロードバランサーの作成」を参照してください。

1 つまたは 2 つのリスナー

ロードバランサーは、リスナーを使用してターゲットグループにトラフィックをルーティングします。本稼働リスナーが 1 つ必要です。検証テストの実行中、置き換えタスクセットにトラフィックをルーティングする、2 番目のオプションのテストリスナーを指定できます。デプロイグループを作成するときに、一方または両方のリスナーを指定します。Amazon ECS コンソールを使用して Amazon ECS サービスを作成すると、リスナーが作成されます。詳細については、「Elastic Load Balancing ユーザーガイド」の「Application Load Balancer のリスナー」そして「Amazon Elastic Container Service ユーザーガイド」の「サービスの作成」を参照してください。

2 つの Amazon ECS ターゲットグループ

ターゲットグループは、登録済みターゲットにトラフィックをルーティングするために使用されます。Amazon ECS デプロイには 2 つのターゲットグループが必要です。1 つは Amazon ECS アプリケーションの元のタスクセット用、もう 1 つはその置き換えタスクセット用です。デプロイ中、CodeDeploy は置き換えタスクセットを作成し、元のタスクセットから新しいタスクセットにトラフィックを再ルーティングします。CodeDeploy アプリケーションのデプロイグループを作成するときに、ターゲットグループを指定します。

デプロイ中、CodeDeploy は、ステータス `PRIMARY` (これが元のタスクセット) を持つ、Amazon ECS サービスのタスクセットに関連付けられているターゲットグループを決定し、それに一方のターゲットグループを関連付けます。さらに、もう一方のターゲットグループを置き換えタスクセットと関連付けます。別のデプロイを行う場合、現在のデプロイの元のタスクセットに関連付けられているターゲットグループは、次のデプロイの置き換えタスクセットに関連付けられています。詳細については、「Elastic Load Balancingのユーザーガイド」の「Application Load Balancer のターゲットグループ」を参照してください。

Amazon ECS タスク定義

Amazon ECS アプリケーションを含んだ Docker コンテナを実行するには、タスク定義 が必要です。CodeDeploy アプリケーションの AppSpec ファイルで、タスク定義の ARN を指定します。詳細については、「Amazon Elastic Container Service ユーザーガイド」と Amazon ECS デプロ

イ用の AppSpec の「resources」セクション の「Amazon ECS のタスクロール」を参照してください。

Amazon ECS アプリケーション用のコンテナ

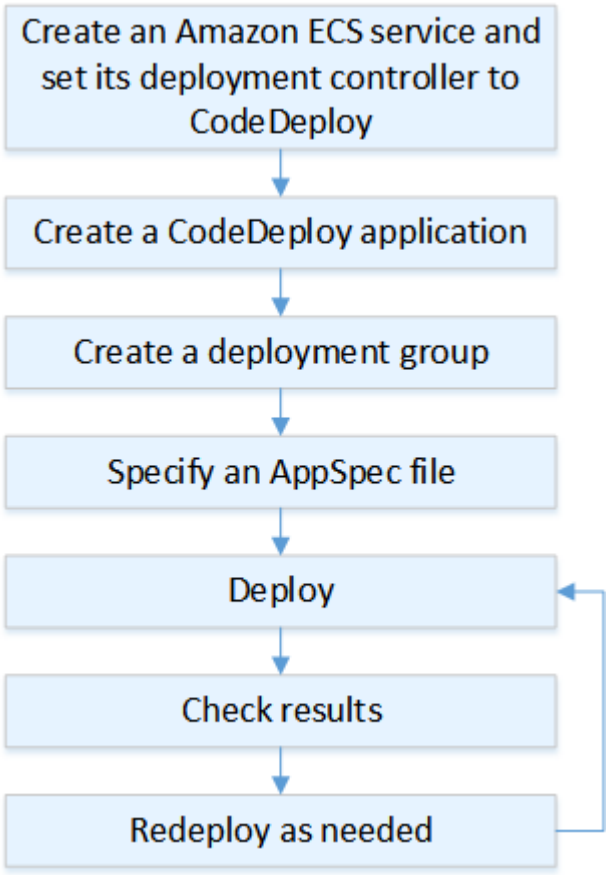
Docker コンテナは、コードとその依存関係をパッケージ化してアプリケーションを実行できるようにするソフトウェアのユニットです。コンテナはアプリケーションを分離して、さまざまなコンピューティング環境で実行できるようにします。ロードバランサーは、Amazon ECS アプリケーションのタスクセットのコンテナにトラフィックをルーティングします。CodeDeploy アプリケーションの AppSpec ファイルで、コンテナの名前を指定します。AppSpec ファイル で指定したコンテナは、Amazon ECS タスク定義で指定したコンテナのいずれかである必要があります。詳細については、「Amazon Elastic Container Service ユーザーガイド」と Amazon ECS デプロイ用の AppSpec の「resources」セクション の「Amazon Elastic Container Service とは」を参照してください。

置き換えタスクセット用のポート

Amazon ECS デプロイ中に、ロードバランサーが CodeDeploy アプリケーションの AppSpec ファイルで指定されたコンテナ上の ポート にトラフィックを指定します。CodeDeploy アプリケーションの AppSpec ファイル でポートを指定します。詳細については、「Amazon ECS デプロイ用の AppSpec の「resources」セクション」を参照してください。

Amazon ECS コンピューティングプラットフォームのデプロイワークフロー (高レベル)

次の図表は、更新された Amazon ECS サービスのデプロイの主要なステップを示しています。



ステップには以下が含まれます。

1. デプロイする内容を一意に表す名前を指定して、 AWS CodeDeploy アプリケーションを作成します。Amazon ECS アプリケーションをデプロイするには、 AWS CodeDeploy アプリケーションで Amazon ECS コンピューティングプラットフォームを選択します。CodeDeploy は、デプロイ中にアプリケーションを使用して、デプロイグループ、ターゲットグループ、リスナー、トラフィックの再ルーティング動作、およびアプリケーションリビジョンなどの正しいデプロイコンポーネント参照します。詳細については、「CodeDeploy でアプリケーションを作成する」を参照してください。
2. デプロイグループをセットアップするには、以下を指定します。
   - デプロイグループ名。
   - お客様の Amazon ECS クラスターとサービス名 Amazon ECS サービスのデプロイコントローラーは、CodeDeploy に設定する必要があります。
   - 本稼働リスナー、オプションのテストリスナー、およびターゲットグループは、デプロイ中に使用されます。

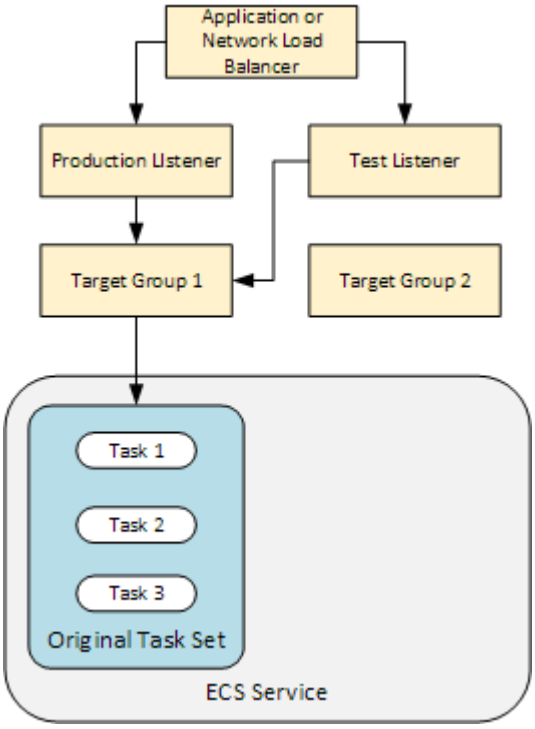
- 本稼働トラフィックを Amazon ECS サービスの置き換え Amazon ECS タスクセットに再ルーティングするタイミングや、Amazon ECS サービスの元の Amazon ECS タスクセットを終了するタイミングなどのデプロイ設定。
- トリガー、アラーム、ロールバック動作などのオプション設定。

3. アプリケーション仕様ファイル (AppSpec ファイル) を指定します。Amazon S3 にアップロードしたり、YAML または JSON 形式でコンソールに入力したり、 AWS CLI または SDK で指定したりできます。AppSpec ファイル は、デプロイの Amazon ECS タスク定義、トラフィックをルーティングするコンテナ名とポートマッピング、およびデプロイのライフサイクルフックの後で実行される Lambda 関数を指定するために使用されます。コンテナ名は、Amazon ECS タスク定義内のコンテナである必要があります。詳細については、「CodeDeploy のアプリケーションリビジョンの操作」を参照してください。
4. アプリケーションリビジョンをデプロイします。 AWS CodeDeploy rerout は、Amazon ECS サービスのタスクセットの元のバージョンから新しい置き換えタスクセットにトラフィックをデプロイします。デプロイグループで指定されたターゲットグループは、元のタスクセットと置き換えタスクセットにトラフィックを提供するために使用されます。デプロイが完了すると、元のタスクセットは削除されます。トラフィックが再ルーティングされる前に、テストトラフィックを置き換えバージョンに提供するためのオプションのテストリスナーを指定できます。詳細については、「CodeDeploy でデプロイを作成する」を参照してください。
5. デプロイの結果を確認します。詳細については、「CodeDeploy でのデプロイモニタリング」を参照してください。

Amazon ECS デプロイ中の処理で起こっていること

テストリスナーを使用して Amazon ECS デプロイを開始する前に、そのコンポーネントを設定する必要があります。詳細については、「Amazon ECS デプロイを開始する前に」を参照してください。

次の図表は、Amazon ECS デプロイを開始する準備ができたときのこれらのコンポーネント間の関係を示しています。



デプロイが開始されたら、デプロイライフサイクルイベントが一度に 1 つずつ実行され始めます。ライフサイクルイベントの中には、AppSpec ファイル で指定されている Lambda 関数のみを実行するフックがあります。次の表のデプロイのライフサイクルイベントは、実行された順序で一覧表示されています。詳細については、「Amazon ECS のデプロイ向けの AppSpec の「hooks」セクション」を参照してください。

ライフサイクルイベント	ライフサイクルイベントアクション
`BeforeInstall` (Lambda 関数のフック)	Lambda 関数を実行します。
インストール	代替タスクの設定を行います。
`AfterInstall` (Lambda 関数のフック)	Lambda 関数を実行します。
AllowTestTraffic	テストリスナーからターゲットグループ 2 にトラフィックをルーティングします。
`AfterAllowTestTraffic` (Lambda 関数のフック)	Lambda 関数を実行します。
`BeforeAllowTraffic` (Lambda 関数のフック)	Lambda 関数を実行します。

ライフサイクルイベント	ライフサイクルイベントアクション
AllowTraffic	本稼働リスナーからターゲットグループ 2 にトラフィックをルーティングします。
`AfterAllowTraffic`	Lambda 関数を実行します。

> **Note**
>
> フックの Lambda 関数はオプションです。

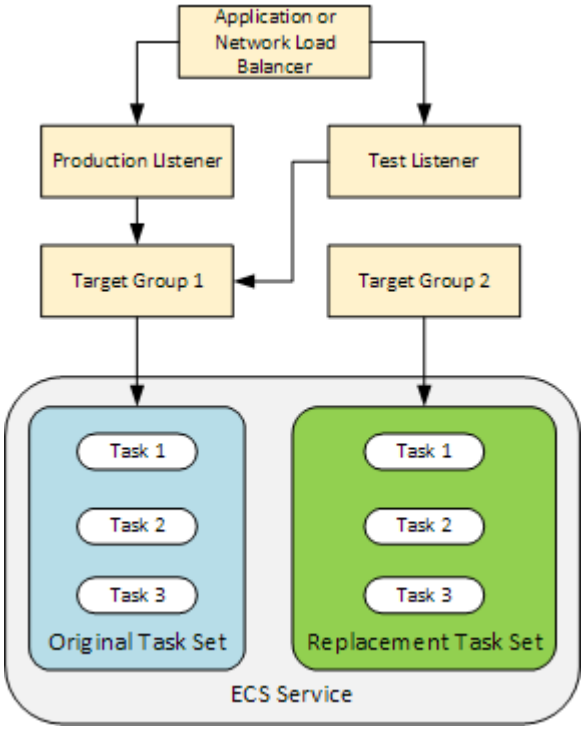
1. AppSpec ファイルの `BeforeInstall` フックで指定された Lambda 関数を実行します。
2. `Install` ライフサイクルイベント中:

   a. 代替タスクセットが Amazon ECS サービスで作成されます。

   b. 更新後のコンテナ化されたアプリケーションは、置き換えタスクセットにインストールされます。

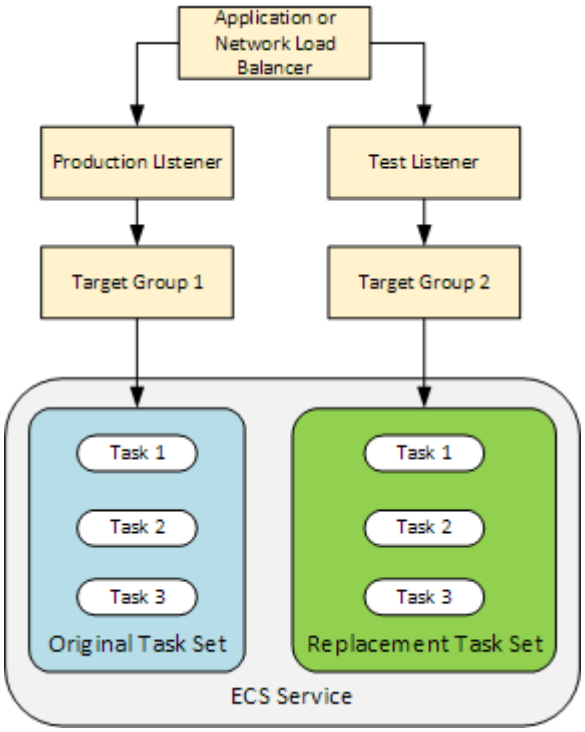
   c. 2 番目のターゲットグループは置き換えタスクセットに関連付けられています。

   この図は、新しい置き換えタスクセットを含むデプロイコンポーネントを示しています。コンテナ化されたアプリケーションはこのタスクセット内にあります。タスクセットは 3 つのタスクで構成されています。(アプリケーションには任意の数のタスクを含めることができます。) 2 番目のターゲットグループが置き換えタスクセットに関連付けられました。



3. AppSpec ファイルの `AfterInstall` フックで指定された Lambda 関数を実行します。

4. `AllowTestTraffic` イベントが呼び出されます。このライフサイクルイベントの間、テストリスナーは、更新されたコンテナ化アプリケーションにトラフィックをルーティングします。



5. AppSpec ファイルの `AfterAllowTestTraffic` フックで指定された Lambda 関数を実行します。Lambda 関数は、テストトラフィックを使用してデプロイを検証します。たとえば、Lambda 関数はテストリスナーにトラフィックを送信し、置き換えタスクセットのメトリクスを追跡できます。ロールバックが設定されている場合は、Lambda 関数内の検証テストが失敗したときにロールバックをトリガーする CloudWatch アラームを設定できます。

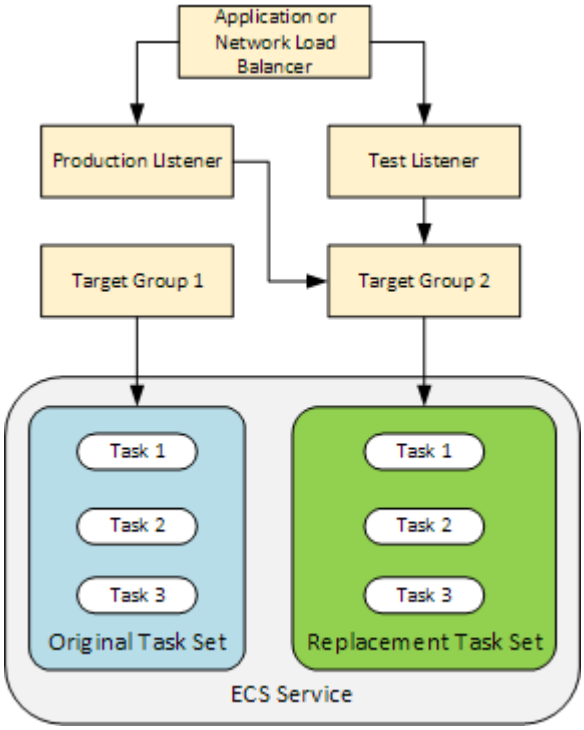
   検証テストが完了したら、次のいずれかが発生します。

   - 検証が失敗し、ロールバックが設定されている場合、デプロイステータスは `Failed` とマークされ、コンポーネントはデプロイが開始されたときの状態に戻ります。
   - 検証が失敗し、ロールバックが設定されていない場合、デプロイステータスは `Failed` とマークされ、コンポーネントは現在の状態のまま変わりません。
   - 検証が正常に完了すると、デプロイは引き続き `BeforeAllowTraffic` に進みます。

   詳細については、CodeDeploy での CloudWatch アラームを使用したデプロイのモニタリング、自動ロールバック、および デプロイグループの詳細オプションの設定 を参照してください。

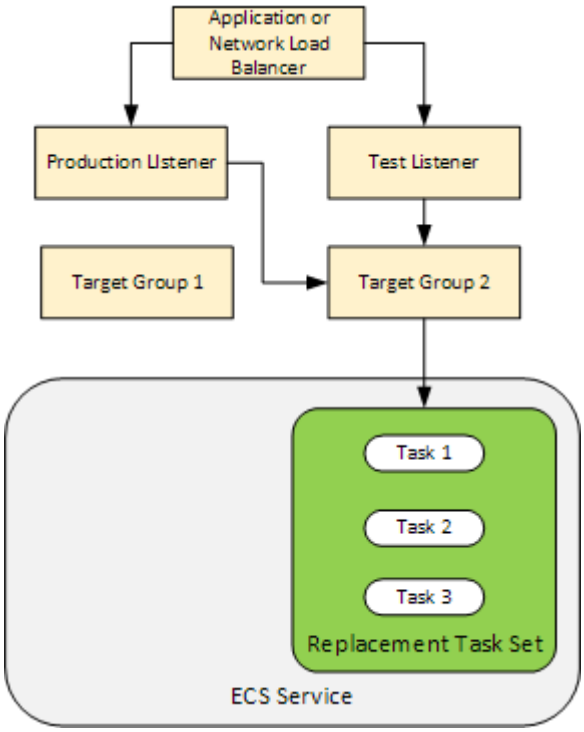
6. AppSpec ファイルの `BeforeAllowTraffic` フックで指定された Lambda 関数を実行します。

7. `AllowTraffic` イベントが呼び出されます。本稼働トラフィックは、元のタスクセットから置き換えタスクセットに再ルーティングされます。次の図は、本稼働トラフィックを受信している代替タスクセットを示しています。



8. AppSpec ファイルの `AfterAllowTraffic` フックで指定された Lambda 関数を実行します。

9. すべてのイベントが正常に完了したら、デプロイステータスは Succeeded になり、元のタスクセットは削除されます。



アプリケーションリビジョンのアップロード

Amazon S3 に AppSpec ファイルを配置するか、コンソールまたは AWS CLIに直接入力します。詳細については、「Application Specification Files」を参照してください。

アプリケーションとデプロイグループの作成

Amazon ECS コンピューティングプラットフォーム上の CodeDeploy デプロイグループは、更新された Amazon ECS アプリケーションへのトラフィックを提供するリスナー、およびデプロイ中に使用される 2 つのターゲットグループを識別します。デプロイグループは、アラームおよびロールバックの設定などの設定オプションのセットも定義します。

アプリケーションリビジョンのデプロイ

これで、デプロイグループで指定された、更新された Amazon ECS サービスをデプロイする準備が整いました。CodeDeploy コンソールまたは create-deployment コマンドを使用できます。デプロイを制御するために指定できるパラメータ (リビジョン、デプロイグループなど) があります。

アプリケーションの更新

アプリケーションを更新し、CodeDeploy コンソールを使用するか、create-deployment コマンドを呼び出してリビジョンをプッシュできます。

停止、失敗したデプロイ

デプロイを停止するには、CodeDeploy コンソールまたはstop-deploymentコマンドを使用できます。デプロイを停止しようとする場合、次の 3 つのうち 1 つのことが発生します。

- デプロイは停止し、オペレーションは成功というステータスを返す。この場合、停止したデプロイに対してそれ以上デプロイライフサイクルイベントは実行されません。
- デプロイは即時に停止せず、オペレーションは保留中というステータスを返す。この場合、一部のデプロイライフサイクルイベントは、デプロイグループでまだ実行中である可能性があります。保留中のオペレーションが完了すると、デプロイを停止するためのそれ以降の呼び出しは、成功というステータスを返します。
- デプロイは停止できず、オペレーションはエラーを返す。詳細については、 AWS CodeDeploy API リファレンスの「エラー情報」と「一般的なエラー」を参照してください。

デプロイと再デプロイのロールバック

CodeDeploy は、置き換えタスクセットから元のタスクセットにトラフィックを再ルーティングすることにより、ロールバックを実装します。

デプロイが失敗した、アラームのモニタリングしきい値に一致したなど、特定の条件が満たされた場合に、自動的にデプロイをロールバックするようグループデプロイを設定できます。個別のデプロイで、デプロイグループに指定されたロールバック設定をオーバーライドすることもできます。

以前のデプロイされたバージョンを手動で再デプロイして、失敗したデプロイをロールバックすることもできます。

いずれの場合でも、新しいデプロイまたはロールバックされたデプロイには独自のデプロイ ID が割り当てられます。CodeDeploy コンソールには、自動デプロイの結果であるデプロイの一覧が表示されます。

デプロイする場合、現在のデプロイの元のタスクセットに関連付けられているターゲットグループは、デプロイの置き換えタスクセットに関連付けられています。

詳細については、「CodeDeploy を使用した再デプロイおよびデプロイのロールバック」を参照してください。

による Amazon ECS ブルー/グリーンデプロイ AWS CloudFormation

を使用して AWS CloudFormation 、CodeDeploy を通じて Amazon ECS ブルー/グリーンデプロイを管理できます。詳細については、「を使用して Amazon ECS ブルー/グリーンデプロイを作成する CloudFormation」を参照してください。

> **Note**
>
> を使用した Amazon ECS ブルー/グリーンデプロイの管理 CloudFormation は、アジアパシフィック (大阪) リージョンでは利用できません。

EC2/オンプレミスコンピューティングプラットフォームの Blue/Green デプロイ

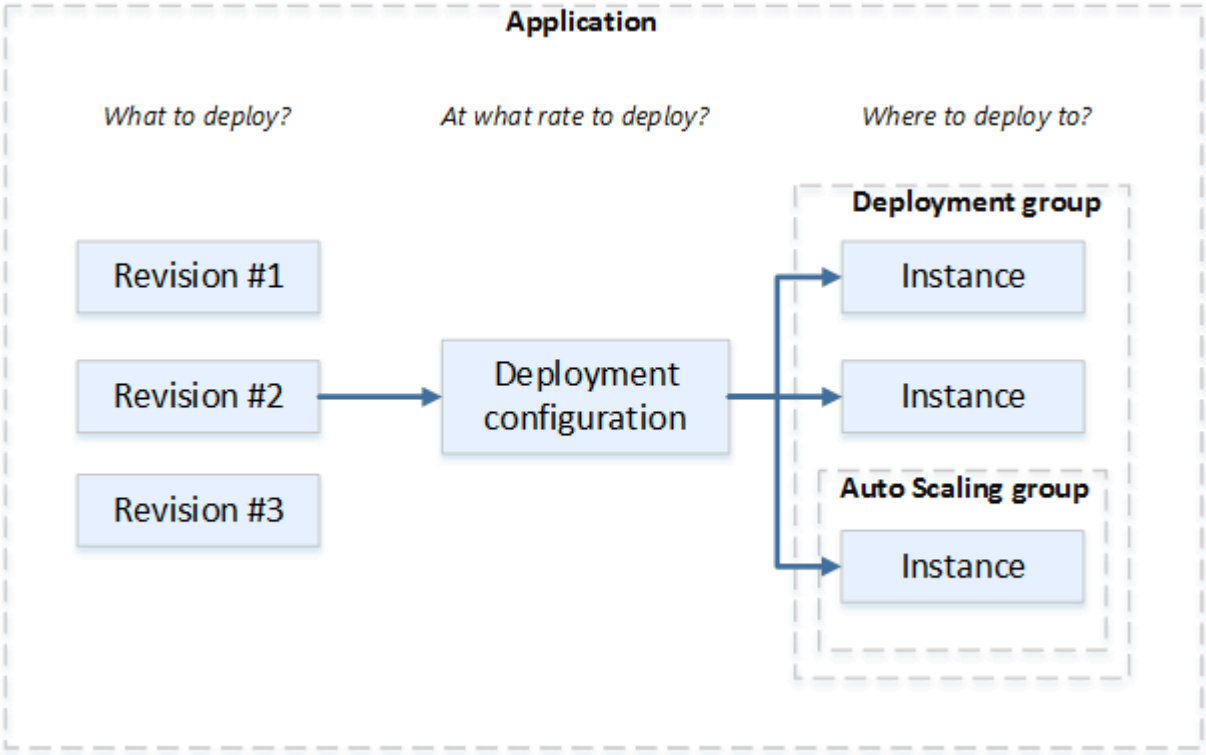
このトピックでは、EC2 オンプレミスのコンピューティングプラットフォームを使用する CodeDeploy のデプロイのコンポーネントおよびワークフローについて説明します。Blue/Green デプロイの詳細については、「Blue/Green デプロイの概要」を参照してください。

トピック

- EC2/オンプレミスコンピューティングプラットフォームのデプロイコンポーネント
- EC2/オンプレミスコンピューティングプラットフォームのデプロイワークフロー
- インスタンスの設定
- アプリケーションリビジョンのアップロード
- アプリケーションとデプロイグループの作成
- アプリケーションリビジョンのデプロイ
- アプリケーションの更新
- 停止、失敗したデプロイ
- デプロイと再デプロイのロールバック

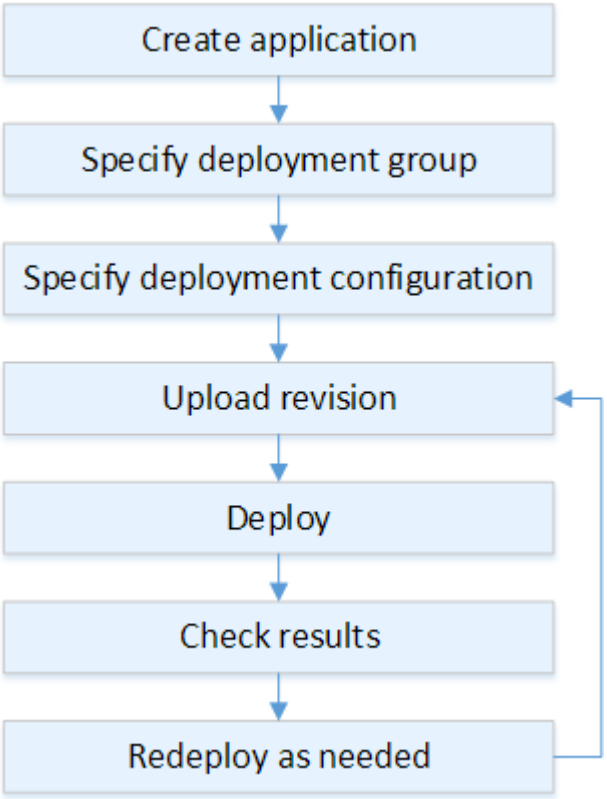
EC2/オンプレミスコンピューティングプラットフォームのデプロイコンポーネント

以下の図表は、EC2/オンプレミスコンピューティングプラットフォームの CodeDeploy デプロイのコンポーネントを示します。



EC2/オンプレミスコンピューティングプラットフォームのデプロイワークフロー

次の図は、アプリケーションリビジョンのデプロイの主要なステップを示しています。



ステップには以下が含まれます。

1. アプリケーションを作成し、デプロイするアプリケーションリビジョンとアプリケーションのコンピューティングプラットフォームを一意に識別する名前を付けます。CodeDeploy は、この名前を使用して、デプロイグループ、デプロイ設定、アプリケーションリビジョンなど、正しいデプロイコンポーネントを参照していることを確認します。詳細については、「CodeDeploy でアプリケーションを作成する」を参照してください。
2. アプリケーションリビジョンをデプロイするデプロイタイプとインスタンスを指定して、デプロイグループをセットアップします。インプレースデプロイでは、最新のアプリケーションリビジョンでインスタンスを更新します。Blue/Green デプロイはロードバランサーでデプロイグループ用の代替セットを登録し、元のインスタンスを登録解除します。

   インスタンス、Amazon EC2 Auto Scaling グループ名、または両方に適用するタグを指定できます。

   デプロイグループのタグのグループを指定すると、CodeDeploy は指定されたタグの少なくとも 1 つが適用されたインスタンスにデプロイします。2 つ以上のタググループを指定した場合、CodeDeploy はそれぞれのタググループの条件を満たすインスタンスにのみデプロイします。詳細については、「Tagging Instances for Deployments」を参照してください。

いずれの場合も、インスタンスはデプロイで使用するよう設定されていること (つまり、タグが付いているか、Amazon EC2 Auto Scaling グループに所属していること)、および CodeDeploy エージェントをインストールして実行していることが必要です。

CloudFormation テンプレートを使用して、Amazon Linux または Windows Server に基づいて Amazon EC2 インスタンスをすばやくセットアップできます。また、スタンドアロン CodeDeploy エージェントも提供しています。これにより、Amazon Linux、Ubuntu Server、Red Hat Enterprise Linux (RHEL)、または Windows Server インスタンスにインストールできます。詳細については、「CodeDeploy でデプロイグループを作成する」を参照してください。

また、以下のオプションを指定することもできます。

- Amazon SNS の通知 成功イベントや失敗イベントなど、指定されたイベントがデプロイとインスタンスで発生したときに、Amazon SNS トピックの受信者に通知を送信するトリガーを作成します。詳細については、「Monitoring Deployments with Amazon SNS Event Notifications」を参照してください。
- アラームベースのデプロイ管理。CloudWatch で設定したしきい値を超えるか下回ったときに、デプロイを停止する Amazon CloudWatch アラームモニタリングを実装します。
- 自動デプロイロールバック。デプロイが失敗するか、アラームのしきい値に一致したときに、以前の既知の正常なリビジョンに自動的にロールバックするようデプロイを設定します。

3. アプリケーションのリビジョンを同時にデプロイする必要があるインスタンスの数と、デプロイの成功と失敗の条件を示すために、デプロイ構成を指定します。詳細については、「View Deployment Configuration Details」を参照してください。
4. アプリケーションリビジョンを Amazon S3 または GitHub にアップロードします。デプロイするファイルおよびデプロイ中に実行するスクリプトに加えて、アプリケーション 仕様ファイル (AppSpec ファイル) を含める必要があります。このファイルには、ファイルを各インスタンスにコピーする場所や、デプロイスクリプトを実行するタイミングなど、デプロイの手順が含まれています。詳細については、「CodeDeploy のアプリケーションリビジョンの操作」を参照してください。
5. デプロイグループにアプリケーションリビジョンをデプロイします。デプロイグループの各インスタンスの CodeDeploy エージェントは、Amazon S3 または GitHub からインスタンスにアプリケーションリビジョンをコピーします。次に、CodeDeploy エージェントは、リビジョンをバンドル解除し、AppSpec ファイル を使用してファイルを指定された場所にコピーして、デプロイスクリプトを実行します。詳細については、「CodeDeploy でデプロイを作成する」を参照してください。

6. デプロイの結果を確認します。詳細については、「CodeDeploy でのデプロイモニタリング」を参照してください。
7. リビジョンをデプロイします。ソースコンテンツのバグを修正する、別の順序でデプロイスクリプトを実行する、または失敗したデプロイに対応する必要がある場合に、この作業を行います。これを行うには、変更したソースコンテンツ、デプロイスクリプト、および AppSpec ファイルを新しいリビジョンを再バンドルし、このリビジョンを Amazon S3 バケットまたは GitHub リポジトリにアップロードします。次に、新しいリビジョンで同じデプロイグループに新しいデプロイを実行します。詳細については、「CodeDeploy でデプロイを作成する」を参照してください。

インスタンスの設定

アプリケーションリビジョンを初めてデプロイする前に、インスタンスを設定する必要があります。アプリケーションリビジョンで 3 つの本番稼働用サーバーと 2 つのバックアップサーバーが必要な場合、5 つのインスタンスを起動または使用します。

インスタンスを手動でプロビジョニングするには:

1. CodeDeploy エージェントをインスタンスにインストールする CodeDeploy エージェントは、Amazon Linux、Ubuntu サーバー、RHEL、および Windows Server インスタンスにインストールできます。
2. タグを使用してデプロイグループのインスタンスを識別する場合は、タグ付けを有効にします。CodeDeploy は、CodeDeploy デプロイグループにインスタンスを識別し、グループ化するタグを使用します。入門チュートリアルでは両方を使用しましたが、キーまたは値を使用して、デプロイグループのタグを定義できます。
3. IAM インスタンスプロファイルをアタッチして、Amazon EC2 インスタンスを起動します。CodeDeploy エージェントでインスタンスの ID を検証するには、Amazon EC2 インスタンスを起動する際に IAM インスタンスプロファイルをアタッチする必要があります。
4. サービスロールを作成します。CodeDeploy が AWS アカウントのタグを拡張できるように、サービスアクセスを提供します。

最初のデプロイでは、 CloudFormation テンプレートがこれらをすべて実行します。これにより、CodeDeploy エージェントがインストール済みの Amazon Linux または Windows Serverをベースに、一つの新規Amazon EC2 インスタンスを作成および設定することができます。詳細については、「CodeDeploy のためにインスタンスを用いた操作」を参照してください。

> **Note**
>
> Blue/Green デプロイでは、置き換え先環境用の既存のインスタンスを使用するか、デプロイプロセスの一部として CodeDeploy で新しいインスタンスをプロビジョニングするか選択できます。

アプリケーションリビジョンのアップロード

アプリケーションのソースコンテンツフォルダ構造で、ルートフォルダの下に AppSpec ファイル を配置します。詳細については、「Application Specification Files」を参照してください。

zip、tar、または圧縮された tar などのアーカイブファイル形式にアプリケーションのソースコンテンツフォルダ構造をバンドルします。アーカイブファイル (リビジョン) を Amazon S3 バケットまたは GitHub リポジトリにアップロードします。

> **Note**
>
> tar および圧縮 tar アーカイブファイル形式 (.tar および .tar.gz) は、Windows Server インスタンスではサポートされていません。

アプリケーションとデプロイグループの作成

CodeDeploy デプロイグループはタグ、Amazon EC2 Auto Scaling グループ名、または両方に基づいてインスタンスのコレクションを識別します。複数のアプリケーションリビジョンを同じインスタンスにデプロイできます。1 つのアプリケーションリビジョンを複数のインスタンスにデプロイできます

たとえば、3 つの本番稼働用サーバーに「Prod」というタグを追加し、2 つのバックアップサーバーに「Backup」というタグを追加できます。これら 2 つのタグを使用して、CodeDeploy アプリケーションで 2 つの異なるデプロイグループを作成し、デプロイにどちらのサーバーのセットを参加させるか (または両方を参加させるか) 選択することができます。

デプロイグループの複数のタググループを使用して、デプロイするインスタンスのセットを減らすことができます。詳細については、「Tagging Instances for Deployments」を参照してください。

アプリケーションリビジョンのデプロイ

これで、Amazon S3 または GitHub からデプロイグループにアプリケーションリビジョンをデプロイできる状態になりました。CodeDeploy コンソールまたは create-deployment コマンドを使用できます。デプロイを制御するために指定できるパラメータ (リビジョン、デプロイグループ、デプロイ設定など) があります。

アプリケーションの更新

アプリケーションを更新し、CodeDeploy コンソールを使用するか、create-deployment コマンドを呼び出してリビジョンをプッシュできます。

停止、失敗したデプロイ

デプロイを停止するには、CodeDeploy コンソールまたはstop-deploymentコマンドを使用できます。デプロイを停止しようとする場合、次の 3 つのうち 1 つのことが発生します。

- デプロイは停止し、オペレーションは成功というステータスを返す。この場合、停止したデプロイに対してそれ以上デプロイライフサイクルイベントは実行されません。デプロイグループで一部のファイルは既にコピーされ、一部のスクリプトは実行され、1 つ以上のインスタンスが実行されている可能性があります。
- デプロイは即時に停止せず、オペレーションは保留中というステータスを返す。この場合、一部のデプロイライフサイクルイベントは、デプロイグループでまだ実行中である可能性があります。デプロイグループで一部のファイルは既にコピーされ、一部のスクリプトは実行され、1 つ以上のインスタンスが実行されている可能性があります。保留中のオペレーションが完了すると、デプロイを停止するためのそれ以降の呼び出しは、成功というステータスを返します。
- デプロイは停止できず、オペレーションはエラーを返す。詳細については、 AWS CodeDeploy API リファレンスのErrorInformation」と「一般的なエラー」を参照してください。

失敗したデプロイでは、停止されたデプロイのように、デプロイグループの 1 つ以上のインスタンスで一部のデプロイライフサイクルイベントが実行済みになる場合があります。デプロイが失敗した理由を調べるには、CodeDeploy コンソールを使用して、get-deployment-instance コマンドを呼び出すか、失敗したデプロイのログファイルデータを分析することができます。詳細については、「アプリケーションリビジョンとログファイルのクリーンアップ」および「CodeDeploy EC2/オンプレミスデプロイのログデータの表示」を参照してください。

デプロイと再デプロイのロールバック

CodeDeploy は新しいデプロイとして、以前にデプロイされたリビジョンを再デプロイすることによって、ロールバックを実装します。

デプロイが失敗した、アラームのモニタリングしきい値に一致したなど、特定の条件が満たされた場合に、自動的にデプロイをロールバックするようグループデプロイを設定できます。個別のデプロイで、デプロイグループに指定されたロールバック設定をオーバーライドすることもできます。

以前のデプロイされたバージョンを手動で再デプロイして、失敗したデプロイをロールバックすることもできます。

いずれの場合でも、新しいデプロイまたはロールバックされたデプロイには独自のデプロイ ID が割り当てられます。CodeDeploy コンソールで表示できるデプロイの一覧には、どれが自動デプロイの結果であるかが示されます。

詳細については、「CodeDeploy を使用した再デプロイおよびデプロイのロールバック」を参照してください。

CodeDeploy アプリケーション仕様 (AppSpec) ファイル

アプリケーション仕様ファイル (AppSpec ファイル) は、YAML 形式、または JSON 形式のファイルであり、CodeDeploy に固有のものです。AppSpec ファイルは、ファイルで定義された一連のライフサイクルイベントフックとして、各デプロイを管理するために使用されます。

正しい形式のAppSpec ファイルを作成する方法については、「CodeDeploy AppSpec ファイルのリファレンス」を参照してください。

トピック

- Amazon ECS コンピューティングプラットフォーム上の AppSpec ファイル
- AWS Lambda コンピューティングプラットフォーム上の AppSpec ファイル
- EC2 オンプレミスコンピューティングプラットフォーム上の AppSpec ファイル
- CodeDeploy エージェントが AppSpec ファイルを使用する方法

Amazon ECS コンピューティングプラットフォーム上の AppSpec ファイル

アプリケーションで Amazon ECS コンピューティングプラットフォームを使用している場合は、AppSpec ファイルを YAML または JSON 形式にすることができます。また、コンソールのエディタに直接入力することもできます。AppSpec ファイルは、以下を指定するために使用されます。

- Amazon ECS サービスの名称と、新しいタスクセットにトラフィックを送信するために使用されるコンテナの名称およびポート。
- 検証テストとして使用される関数。

Lambda 関数は、デプロイライフサイクルイベント後に検証を実行できます。詳細については、Amazon ECS のデプロイ向けの AppSpec の「hooks」セクション、Amazon ECS デプロイ向けの AppSpec ファイル構造、および Amazon ECS デプロイの AppSpec ファイルの例 を参照してください。

AWS Lambda コンピューティングプラットフォーム上の AppSpec ファイル

アプリケーションで AWS Lambda コンピューティングプラットフォームを使用している場合、AppSpec ファイルは YAML または JSON のいずれかでフォーマットできます。また、コンソールのエディタに直接入力することもできます。AppSpec ファイルは、以下を指定するために使用されます。

- デプロイする AWS Lambda 関数のバージョン。
- 検証テストとして使用される関数。

Lambda 関数は、デプロイライフサイクルイベント後に検証を実行できます。詳細については、「AWS Lambda デプロイの AppSpec 'hooks' セクション」を参照してください。

EC2 オンプレミスコンピューティングプラットフォーム上の AppSpec ファイル

アプリケーションで EC2 オンプレミスコンピューティングプラットフォームを使用している場合、AppSpec ファイルは必ず YAML 形式になります。AppSpec ファイルは、以下を目的として使用されます。

- アプリケーションリビジョンのソースファイルを、インスタンスの宛先にマッピングします。
- デプロイされたファイルのカスタムアクセス権限を指定する。
- デプロイプロセスのさまざまなフェーズにおいて、各インスタンスで実行するスクリプトを指定する。

個別のデプロイライフサイクルイベントのうちの多くは、発生後にインスタンス上でスクリプトの実行が可能となっています。CodeDeploy はファイルで指定されているスクリプトのみを実行しますが、これらのスクリプトは、インスタンスの他のスクリプトを呼び出すことができます。インスタンスで実行しているオペレーティングシステムでサポートされている限り、どのタイプのスクリプトでも実行できます。詳細については、「EC2/オンプレミスのデプロイ向けの AppSpec の「hooks」セクション」を参照してください。

CodeDeploy エージェントが AppSpec ファイルを使用する方法

デプロイ中、CodeDeploy エージェントは、現在のイベントの名前を AppSpec ファイル内の フック セクションで検索できます。イベントが見つからない場合、CodeDeploy エージェントは次のステップに進みます。イベントが見つかった場合、CodeDeploy エージェントは実行するスクリプトのリストを取得します。スクリプトはファイルに表示された順序に沿って実行されます。各スクリプトのステータスはインスタンス上の CodeDeploy エージェントログファイルに記録されます。

スクリプトが正常に実行すると、終了コード 0 (ゼロ) を返します。

> **Note**
>
> CodeDeploy エージェントは、 AWS Lambda または Amazon ECS デプロイでは使用されません。

Install イベント中、CodeDeploy エージェントは AppSpec ファイルの files セクション内に定義されているマッピングを使用して、リビジョンからインスタンスにコピーするフォルダまたはファイルを判断します。

オペレーティングシステムにインストールされている CodeDeploy エージェントが AppSpec ファイルに記載されているものと一致しない場合、デプロイは失敗します。

CodeDeploy エージェントログファイルの詳細については、CodeDeploy エージェントの使用 を参照してください。

CodeDeploy の開始方法

トピック

- ステップ 1: セットアップ
- ステップ 2: CodeDeployのサービスのロールを作成する
- ステップ 3: CodeDeploy ユーザーのアクセス許可を制限する
- ステップ 4: Amazon EC2 インスタンス用の IAM インスタンスプロファイルを作成する

ステップ 1: セットアップ

AWS CodeDeploy を初めて使用する場合は、セットアップ手順を完了する必要があります。この手順では、 AWS アカウント (まだ持っていない場合) と管理ユーザーを作成します。

このガイドでは、管理ユーザーは CodeDeploy 管理ユーザーと呼ばれます。

にサインアップする AWS アカウント

の使用を開始するには AWS、 が必要です AWS アカウント。サインアップするには 2 つの方法があります AWS。アカウント設定とコンテンツの保存場所を完全に制御する場合、または HIPAA や FedRAMP などの規制対象のワークロードがある場合は、「Sign up for AWS (advanced)」を使用します。新規ユーザーで、既に所有しているログインを使用してサインイン AWS し、 の選択で事前設定されたデフォルトを使用して AWS 環境を作成する場合はAWS リージョン、サインアップ AWSを使用します。

> **Important**
>
> CodeDeploy 管理ユーザーをワークフォース ID (IAM Identity Center で管理されるユーザー) として AWS CLIで設定することを強くお勧めします。このガイドの手順の多くは、 を使用して設定を実行していること AWS CLI を前提としています。

> **Important**
>
> を設定すると AWS CLI、 AWS リージョンを指定するように求められる場合があります。「AWS 全般のリファレンス」の「リージョンエンドポイント」に記載されているサポートされているリージョンから一つを選びます。

ステップ 2: CodeDeployのサービスのロールを作成する

では AWS、サービスロールを使用して AWS サービスにアクセス許可を付与し、 AWS リソースにアクセスできるようにします。サービスロールにアタッチするポリシーによって、どの リソースにサービスがアクセスできるか、およびそれらのリソースで何ができるかが決まります。

CodeDeploy に作成するサービスロールで、コンピューティングプラットフォームに必要なアクセス許可がを付与する必要があります。 複数のコンピューティングプラットフォームにデプロイした場合、それぞれにサービスロールを 1 つずつ作成します。アクセス許可を追加するには、次の AWS 指定されたポリシーを 1 つ以上アタッチします。

EC2/オンプレミスのデプロイには、**AWSCodeDeployRole** ポリシーをアタッチします。これは、サービスロールで以下を実行するためのアクセス許可を提供します。

- インスタンスのタグを読み取る、または Amazon EC2 Auto Scaling グループ名により Amazon EC2 インスタンスを識別します。
- Amazon EC2 Auto Scaling グループ、ライフサイクルフック、スケーリングポリシーの読み取り、作成、更新、削除を行います。
- Amazon SNS トピックに情報を公開します。
- CloudWatch アラームに関する情報を取得します。
- Elastic Load Balancing を読み、更新します。

> **Note**
>
> 起動テンプレートを使用して Auto Scaling グループを作成する場合は、次のアクセス許可を追加する必要があります。
>
> - `ec2:RunInstances`
> - `ec2:CreateTags`
> - `iam:PassRole`
>
> 詳細については、「Amazon EC2 Auto Scaling ユーザーガイド」、「Auto Scaling グループの起動テンプレートの作成」、および「起動テンプレートサポート」にはの「ステップ 2: サービスロールを作成する」を参照してください。

Amazon ECS のデプロイの場合、サポートサービスへのフルアクセスが必要な場合は、**AWSCodeDeployRoleForECS** ポリシーをアタッチします。これは、サービスロールで以下を実行するためのアクセス許可を提供します。

- Amazon ECS タスクセットを読んで、更新、削除します。
- Elastic Load Balancing ターゲットグループ、リスナー、ルールを更新します。
- AWS Lambda 関数を呼び出します。
- Amazon S3 バケットのリビジョンファイルにアクセスします。
- CloudWatch アラームに関する情報を取得します。
- Amazon SNS トピックに情報を公開します。

Amazon ECS のデプロイの場合、サポートサービスへの制限付きのアクセスが必要な場合は、**AWSCodeDeployRoleForECSLimited** ポリシーをアタッチします。これは、サービスロールで以下を実行するためのアクセス許可を提供します。

- Amazon ECS タスクセットを読んで、更新、削除します。
- CloudWatch アラームに関する情報を取得します。
- Amazon SNS トピックに情報を公開します。

AWS Lambda デプロイでは、Amazon SNS への発行を許可する場合は、**AWSCodeDeployRoleForLambda**ポリシーをアタッチします。これは、サービスロールで以下を実行するためのアクセス許可を提供します。

- AWS Lambda 関数と エイリアスの読み取り、更新、呼び出しを行います。
- Amazon S3 バケットのリビジョンファイルにアクセスします。
- CloudWatch アラームに関する情報を取得します。
- Amazon SNS トピックに情報を公開します。

AWS Lambda デプロイの場合、Amazon SNS へのアクセスを制限する場合は、**AWSCodeDeployRoleForLambdaLimited**ポリシーをアタッチします。これは、サービスロールで以下を実行するためのアクセス許可を提供します。

- AWS Lambda 関数と エイリアスの読み取り、更新、呼び出しを行います。
- Amazon S3 バケットのリビジョンファイルにアクセスします。
- CloudWatch アラームに関する情報を取得します。

また、サービスロールの設定の一環として、アクセス許可を付与するエンドポイントを指定するために信頼関係を更新します。

サービスロールは、IAM コンソール、 AWS CLI、または IAM APIs を使用して作成できます。

トピック

- サービスロールの作成 (コンソール)
- サービスロールの作成 (CLI)
- サービスロール ARN の取得 (コンソール)
- サービスロール ARN の取得 (CLI)

サービスロールの作成 (コンソール)

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. ナビゲーションペインで ロール を選択してから、ロールを作成する を選択します。
3. AWS サービスを選択し、「ユースケース」のドロップダウンリストから [CodeDeploy] を選択します。
4. ユースケースを選択します。

   - EC2/オンプレミスのデプロイ CodeDeploy を選択
   - AWS Lambda デプロイの場合は、CodeDeploy for Lambda を選択します。
   - Amazon ECS デプロイの場合は、CodeDeploy-ECS を選択
5. [次へ] を選択します。
6. 「アクセス許可を追加」ページに、そのユースケースに適したアクセス許可ポリシーが表示されます。[次へ] を選択します。
7. [名前、確認、および作成] ページで、[ロール名] にサービスロールの名前 (例えば、**`CodeDeployServiceRole`**) を入力し、[ロールを作成] を選択します。

   このサービスロールの説明を、[Role description] ボックスに入力することもできます。
8. 現在サポートされているすべてのエンドポイントへのアクセス許可をサービスロールに付与する場合は、この手順を終了します。

   このサービスロールから一部のエンドポイントへのアクセスを制限するには、この手順の残りのステップに進みます。
9. ロールの一覧で、作成したロール (`CodeDeployServiceRole`) を検索し、選択します。
10. [信頼関係] タブを選択します。

11. [Edit trust policy] (信頼ポリシーを編集) を選択します。

次のポリシーでは、サポートされているすべてのエンドポイントにアクセスする権限をサービスロールに付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "codedeploy.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

サポートされているエンドポイントの一部にのみアクセスする権限をサービスロールに付与するには、信頼ポリシーテキストボックスの内容を以下のポリシーに置き換えます。アクセスを除外するエンドポイントの行を削除し、[ポリシーの更新] を選択します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": [

                    "codedeploy.us-east-1.amazonaws.com",
                    "codedeploy.us-east-2.amazonaws.com",
                    "codedeploy.us-west-1.amazonaws.com",
                    "codedeploy.us-west-2.amazonaws.com",
                    "codedeploy.ca-central-1.amazonaws.com",
                    "codedeploy.ap-east-1.amazonaws.com",
                    "codedeploy.ap-northeast-1.amazonaws.com",
                    "codedeploy.ap-northeast-2.amazonaws.com",
```

```
                    "codedeploy.ap-northeast-3.amazonaws.com",
                    "codedeploy.ap-southeast-1.amazonaws.com",
                    "codedeploy.ap-southeast-2.amazonaws.com",
                    "codedeploy.ap-southeast-3.amazonaws.com",
                    "codedeploy.ap-southeast-4.amazonaws.com",
                    "codedeploy.ap-south-1.amazonaws.com",
                    "codedeploy.ap-south-2.amazonaws.com",
                    "codedeploy.ca-central-1.amazonaws.com",
                    "codedeploy.eu-west-1.amazonaws.com",
                    "codedeploy.eu-west-2.amazonaws.com",
                    "codedeploy.eu-west-3.amazonaws.com",
                    "codedeploy.eu-central-1.amazonaws.com",
                    "codedeploy.eu-central-2.amazonaws.com",
                    "codedeploy.eu-north-1.amazonaws.com",
                    "codedeploy.eu-south-1.amazonaws.com",
                    "codedeploy.eu-south-2.amazonaws.com",
                    "codedeploy.il-central-1.amazonaws.com",
                    "codedeploy.me-central-1.amazonaws.com",
                    "codedeploy.me-south-1.amazonaws.com",
                    "codedeploy.sa-east-1.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

サービスロールの作成の詳細については、IAM ユーザーガイドの「AWS サービスにアクセス許可を委任するロールの作成」を参照してください。

サービスロールの作成 (CLI)

1. 開発マシンで、たとえば、CodeDeployDemo-Trust.json という名前のテキストファイルを作成します。このファイルは、CodeDeploy がユーザーの代理操作の実行を許可するのに使用されます。

   次のいずれかを行います。

   - サポートされているすべての AWS リージョンへのアクセスを許可するには、次のコンテンツを ファイルに保存します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "codedeploy.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

- サポートされている一部のリージョンのみへのアクセスを許可するには、ファイルに次のコンテンツを入力し、アクセスを除外するリージョンの行を削除します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": [

                    "codedeploy.us-east-1.amazonaws.com",
                    "codedeploy.us-east-2.amazonaws.com",
                    "codedeploy.us-west-1.amazonaws.com",
                    "codedeploy.us-west-2.amazonaws.com",
                    "codedeploy.ca-central-1.amazonaws.com",
                    "codedeploy.ap-east-1.amazonaws.com",
                    "codedeploy.ap-northeast-1.amazonaws.com",
                    "codedeploy.ap-northeast-2.amazonaws.com",
                    "codedeploy.ap-northeast-3.amazonaws.com",
                    "codedeploy.ap-southeast-1.amazonaws.com",
                    "codedeploy.ap-southeast-2.amazonaws.com",
                    "codedeploy.ap-southeast-3.amazonaws.com",
                    "codedeploy.ap-southeast-4.amazonaws.com",
                    "codedeploy.ap-south-1.amazonaws.com",
```

```
                    "codedeploy.ap-south-2.amazonaws.com",
                    "codedeploy.ca-central-1.amazonaws.com",
                    "codedeploy.eu-west-1.amazonaws.com",
                    "codedeploy.eu-west-2.amazonaws.com",
                    "codedeploy.eu-west-3.amazonaws.com",
                    "codedeploy.eu-central-1.amazonaws.com",
                    "codedeploy.eu-central-2.amazonaws.com",
                    "codedeploy.eu-north-1.amazonaws.com",
                    "codedeploy.eu-south-1.amazonaws.com",
                    "codedeploy.eu-south-2.amazonaws.com",
                    "codedeploy.il-central-1.amazonaws.com",
                    "codedeploy.me-central-1.amazonaws.com",
                    "codedeploy.me-south-1.amazonaws.com",
                    "codedeploy.sa-east-1.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

> **Note**
>
> リストにある最後のエンドポイントの後にコンマを使用しないでください。

2. 同じディレクトリから、create-role コマンドを呼び出し、先ほど作成したテキストファイルの情報に基づく **CodeDeployServiceRole** という名前のサービスロールを作成します。

```
aws iam create-role --role-name CodeDeployServiceRole --assume-role-policy-document
 file://CodeDeployDemo-Trust.json
```

> **Important**
>
> ファイル名の前に必ず `file://` を含めてください。このコマンドでは必須です。

コマンドの出力で、`Arn` オブジェクトの下にある `Role` エントリの値を書きとめておきます。これは、後でデプロイグループを作成する際に必要になります。値を忘れた場合は、[サービスロール ARN の取得 (CLI)] の手順に従います。

3. 使用する管理ポリシーは、コンピューティングプラットフォーム によって異なります。

- EC2/オンプレミスコンピューティングプラットフォームにデプロイする場合は、以下を行います。

  attach-role-policy コマンドを呼び出し、**CodeDeployServiceRole** という名前のサービスロールに対して、**AWSCodeDeployRole** という名前の IAM マネージドポリシーに基づくアクセス許可を付与します。例えば、次のようになります。

  ```
  aws iam attach-role-policy --role-name CodeDeployServiceRole --policy-arn
   arn:aws:iam::aws:policy/service-role/AWSCodeDeployRole
  ```

- Lambda AWS コンピューティングプラットフォームへのデプロイの場合:

  attach-role-policy コマンドを呼び出し、**CodeDeployServiceRole** という名前のサービスロールに対して、**AWSCodeDeployRoleForLambda** または **AWSCodeDeployRoleForLambdaLimited** という名前の IAM マネージドポリシーに基づくアクセス許可を付与します。例:

  ```
  aws iam attach-role-policy --role-name CodeDeployServiceRole --policy-arn
   arn:aws:iam::aws:policy/service-role/AWSCodeDeployRoleForLambda
  ```

- Amazon ECS コンピューティングプラットフォームへのデプロイの場合。

  attach-role-policy コマンドを呼び出し、**CodeDeployServiceRole** という名前のサービスロールに対して、**AWSCodeDeployRoleForECS** または **AWSCodeDeployRoleForECSLimited** という名前の IAM マネージドポリシーに基づくアクセス許可を付与します。例:

  ```
  aws iam attach-role-policy --role-name CodeDeployServiceRole --policy-arn
   arn:aws:iam::aws:policy/AWSCodeDeployRoleForECS
  ```

サービスロールの作成の詳細については、IAM ユーザーガイドの「 AWS サービスのロールの作成」を参照してください。

サービスロール ARN の取得 (コンソール)

IAM コンソールを使用してサービスロールの ARN を取得するには:

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。

2. ナビゲーションペインで Roles (ロール) を選択してください。
3. [フィルタ] テキストボックスに、「**CodeDeployServiceRole**」と入力して、Enter キーを押します。
4. [CodeDeployServiceRole] を選択します。
5. [ロールの ARN] フィールドの値を書き留めます。

サービスロール ARN の取得 (CLI)

を使用してサービスロールの ARN AWS CLI を取得するには、 という名前のサービスロールに対して get-role コマンドを呼び出します**CodeDeployServiceRole**。

```
aws iam get-role --role-name CodeDeployServiceRole --query "Role.Arn" --output text
```

返される値はサービスロールの ARN です。

ステップ 3: CodeDeploy ユーザーのアクセス許可を制限する

セキュリティ上の理由から、ステップ 1: セットアップ で作成した管理ユーザーのアクセス許可を、CodeDeploy でのデプロイの作成と管理に必要なアクセス許可のみに制限することをお勧めします。

以下の一連の手順を使用して、CodeDeploy 管理ユーザーのアクセス許可を制限します。

[開始する前に]

- ステップ 1: セットアップ の手順に従って IAM Identity Center で CodeDeploy 管理ユーザーを作成したことを確認します。

アクセス権限セットを作成するには

このアクセス許可セットは後で CodeDeploy 管理ユーザーに割り当てます。

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/singlesignon/ で AWS IAM アイデンティティセンター コンソールを開きます。
2. ナビゲーションペインで [アクセス許可セット] を選択し、[アクセス許可セットの作成] を選択します。

3. [カスタム許可セット] を選択します。
4. [次へ] を選択します。
5. [インラインポリシー] を選択します。
6. サンプルコードを削除します。
7. 以下のポリシーを追加します。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "CodeDeployAccessPolicy",
            "Effect": "Allow",
            "Action": [
                "autoscaling:*",
                "codedeploy:*",
                "ec2:*",
                "lambda:*",
                "ecs:*",
                "elasticloadbalancing:*",
                "iam:AddRoleToInstanceProfile",
                "iam:AttachRolePolicy",
                "iam:CreateInstanceProfile",
                "iam:CreateRole",
                "iam:DeleteInstanceProfile",
                "iam:DeleteRole",
                "iam:DeleteRolePolicy",
                "iam:GetInstanceProfile",
                "iam:GetRole",
                "iam:GetRolePolicy",
                "iam:ListInstanceProfilesForRole",
                "iam:ListRolePolicies",
                "iam:ListRoles",
                "iam:PutRolePolicy",
                "iam:RemoveRoleFromInstanceProfile",
                "s3:*",
                "ssm:*"
            ],
            "Resource": "*"
        },
```

```
        {
            "Sid": "CodeDeployRolePolicy",
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "arn:aws:iam::111122223333:role/
CodeDeployServiceRole"
        }
    ]
}
```

このポリシーで、*arn:aws:iam::account-ID:role/CodeDeployServiceRole* を ステップ 2: CodeDeployのサービスのロールを作成する で作成した CodeDeploy サービスロールの ARN 値を置換します。ARN 値は、IAM コンソールのサービスロールの詳細ページにあります。

前述のポリシーにより AWS Lambda コンピューティングプラットフォーム、EC2/オンプレミスコンピューティングプラットフォーム、および Amazon ECS コンピューティングプラットフォームにアプリケーションをデプロイできます。

このドキュメントで提供されている CloudFormation テンプレートを使用して、CodeDeploy と互換性のある Amazon EC2 インスタンスを起動できます。 CloudFormation テンプレートを使用してアプリケーション、デプロイグループ、またはデプロイ設定を作成するには、次のように CodeDeploy 管理ユーザーのアクセス許可ポリシーに `cloudformation:*` アクセス許可を追加して、 CloudFormationおよび CloudFormation が依存する AWS サービスとアクションへのアクセスを提供する必要があります。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudformation:*"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

8. [次へ] を選択します。
9. 「アクセス許可セット名」に、次のように入力します。

```
CodeDeployUserPermissionSet
```

10. [次へ] を選択します。
11. [確認と作成] ページで情報を確認し、[グループの作成] を選択します。

許可セットを CodeDeploy 管理ユーザーに割り当てるには

1. ナビゲーションペインで を選択しAWS アカウント、現在サインイン AWS アカウント しているの横にあるチェックボックスをオンにします。
2. [ユーザーまたはグループの割り当て] ボタンを選択します。
3. [ユーザー] タブを選択します。
4. CodeDeploy 管理ユーザーの横にあるチェックボックスをオンにします。
5. [次へ] を選択します。
6. [CodeDeployUserPermissionSet] のチェックボックスをオンにします。
7. [次へ] を選択します。
8. 情報を確認し、[送信] を選択します。

   これで、CodeDeploy 管理ユーザー と を CodeDeployUserPermissionSetに割り当て AWS アカウント、それらをバインドしました。

CodeDeploy 管理ユーザーとしてサインアウトしてサインインし直すには

1. サインアウトする前に、CodeDeploy 管理ユーザーの AWS アクセスポータル URL とユーザー名、ワンタイムパスワードがあることを確認してください。

   > **Note**
   >
   > この情報がない場合は、IAM Identity Center の CodeDeploy 管理ユーザー詳細ページに移動し、[パスワードをリセットする]、[ワンタイムパスワードの生成 [...]] を選択して、もう一度 [パスワードをリセットする] と、画面に情報が表示されます。

2. からサインアウトします AWS。
3. AWS アクセスポータル URL をブラウザのアドレスバーに貼り付けます。
4. CodeDeploy 管理ユーザーとしてサインインする。

   画面に AWS アカウント ボックスが表示されます。
5. AWS アカウント を選択し、CodeDeploy 管理ユーザーと許可セットを割り当てた AWS アカウント の名前を選択します。
6. `CodeDeployUserPermissionSet` の横にある [管理コンソール] を選択します。

   AWS マネジメントコンソール が表示されます。これで、制限付きのアクセス許可を持つ CodeDeploy 管理ユーザーとしてサインインしました。このユーザーとして CodeDeploy 関連の操作のみを実行できるようになりました。

ステップ 4: Amazon EC2 インスタンス用の IAM インスタンスプロファイルを作成する

> **Note**
>
> Amazon ECS または AWS Lambda コンピューティングプラットフォーム を使用している場合は、このステップをスキップします。

Amazon EC2 インスタンスには、アプリケーションが保存される Amazon S3 バケットまたは GitHub レポジトリへのアクセス許可が必要です。CodeDeploy と互換性のある Amazon EC2 インスタンスを起動するには、追加の IAM ロールである インスタンスプロファイル を作成する必要があります。以下の手順では、Amazon EC2 インスタンスにアタッチする IAM インスタンスプロファイルを作成する方法を示します。このロールでは、アプリケーションが保存される Amazon S3 バケットまたは GitHub リポジトリへのアクセス許可が CodeDeploy エージェントに付与されます。

IAM インスタンスプロファイルは AWS CLI、、IAM コンソール、または IAM APIs を使用して作成できます。

> **Note**
>
> IAM インスタンスプロファイルは、起動時の Amazon EC2 インスタンスまたは以前に起動したインスタンスにアタッチできます。詳細については、「インスタンスプロファイル」を参照してください。

トピック

- Amazon EC2 インスタンス(CLI)の IAM インスタンスプロファイルを作成する
- Amazon EC2 インスタンス(コンソール)の IAM インスタンスプロファイルを作成する

Amazon EC2 インスタンス(CLI)の IAM インスタンスプロファイルを作成する

以下のステップでは、「CodeDeploy の開始方法」にある最初の 3 つの手順の指示に従っていることを前提としています。

1. 開発マシンで、CodeDeployDemo-EC2-Trust.json という名前のテキストファイルを作成します。Amazon EC2 によるユーザーの代理操作の実行を許可するには、次の内容を貼り付けます。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": "ec2.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "ec2.cn-north-1.amazonaws.com",
                    "ec2.cn-northwest-1.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

2. 同じディレクトリで、CodeDeployDemo-EC2-Permissions.json という名前のテキストファイルを作成します。以下の内容を貼り付けます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:Get*",
                "s3:List*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> このポリシーを、Amazon EC2 インスタンスがアクセスする必要のある Amazon S3 バケットにのみ制限することをお勧めします。CodeDeploy エージェントを含む Amazon

S3 バケットへのアクセスを必ず許可してください。そうしない場合、CodeDeploy エージェントがインスタンス上にインストールされる、または更新されるときに、エラーが発生する可能性があります。Amazon S3 中の CodeDeploy リソースキットバケットのみへの IAM インスタンスアクセスは許すが、アクセスを防止するバケットのための行を削除するには

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:Get*",
        "s3:List*"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket/*",
        "arn:aws:s3:::aws-codedeploy-us-east-2/*",
        "arn:aws:s3:::aws-codedeploy-us-east-1/*",
        "arn:aws:s3:::aws-codedeploy-us-west-1/*",
        "arn:aws:s3:::aws-codedeploy-us-west-2/*",
        "arn:aws:s3:::aws-codedeploy-ca-central-1/*",
        "arn:aws:s3:::aws-codedeploy-eu-west-1/*",
        "arn:aws:s3:::aws-codedeploy-eu-west-2/*",
        "arn:aws:s3:::aws-codedeploy-eu-west-3/*",
        "arn:aws:s3:::aws-codedeploy-eu-central-1/*",
        "arn:aws:s3:::aws-codedeploy-eu-central-2/*",
        "arn:aws:s3:::aws-codedeploy-eu-north-1/*",
        "arn:aws:s3:::aws-codedeploy-eu-south-1/*",
        "arn:aws:s3:::aws-codedeploy-eu-south-2/*",
        "arn:aws:s3:::aws-codedeploy-il-central-1/*",
        "arn:aws:s3:::aws-codedeploy-ap-east-1/*",
        "arn:aws:s3:::aws-codedeploy-ap-northeast-1/*",
        "arn:aws:s3:::aws-codedeploy-ap-northeast-2/*",
        "arn:aws:s3:::aws-codedeploy-ap-northeast-3/*",
        "arn:aws:s3:::aws-codedeploy-ap-southeast-1/*",
        "arn:aws:s3:::aws-codedeploy-ap-southeast-2/*",
        "arn:aws:s3:::aws-codedeploy-ap-southeast-3/*",
        "arn:aws:s3:::aws-codedeploy-ap-southeast-4/*",
        "arn:aws:s3:::aws-codedeploy-ap-south-1/*",
        "arn:aws:s3:::aws-codedeploy-ap-south-2/*",
        "arn:aws:s3:::aws-codedeploy-me-central-1/*",
        "arn:aws:s3:::aws-codedeploy-me-south-1/*",
```

```
        "arn:aws:s3:::aws-codedeploy-sa-east-1/*"
      ]
    }
  ]
}
```

> **Note**
>
> IAM 認可または CodeDeploy を使用する Amazon Virtual Private Cloud (VPC) エンドポイントを使用したい場合は、アクセス許可を追加する必要があります。詳細については、[Amazon Virtual Private Cloud で CodeDeploy を使う] を参照してください。

3. 同じディレクトリから、create-role コマンドを呼び出して、最初のファイルの情報に基づいて **CodeDeployDemo-EC2-Instance-Profile** という名前の IAM ロールを作成します。

> **Important**
>
> ファイル名の前に必ず `file://` を含めてください。このコマンドでは必須です。

```
aws iam create-role --role-name CodeDeployDemo-EC2-Instance-Profile --assume-role-policy-document file://CodeDeployDemo-EC2-Trust.json
```

4. 同じディレクトリから、put-role-policy コマンドを呼び出して、2 番目のファイルの情報に基づいて **CodeDeployDemo-EC2-Instance-Profile** という名前のロールアクセス許可を付与します。

> **Important**
>
> ファイル名の前に必ず `file://` を含めてください。このコマンドでは必須です。

```
aws iam put-role-policy --role-name CodeDeployDemo-EC2-Instance-Profile --policy-name CodeDeployDemo-EC2-Permissions --policy-document file://CodeDeployDemo-EC2-Permissions.json
```

5. attach-role-policy を呼び出して、SSM が CodeDeploy エージェントをインストールできるように、ロールに Amazon EC2 Systems Manager とアクセス許可を付与します。コマンドラインを使用してパブリック Amazon S3 バケットからエージェントをインストールする場合、このポリシーは必要ありません。「CodeDeploy エージェントのインストール」の詳細を確認してください。

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonSSMManagedInstanceCore --role-name CodeDeployDemo-EC2-Instance-Profile
```

6. create-instance-profile コマンドに続いて add-role-to-instance-profile コマンドを呼び出して、**CodeDeployDemo-EC2-Instance-Profile** という名前の IAM インスタンスプロファイルを作成します。インスタンスプロファイルにより、Amazon EC2 は最初に起動されたときに **CodeDeployDemo-EC2-Instance-Profile** という名前の IAM ロールを Amazon EC2 インスタンスに渡します。

```
aws iam create-instance-profile --instance-profile-name CodeDeployDemo-EC2-
Instance-Profile
aws iam add-role-to-instance-profile --instance-profile-name CodeDeployDemo-EC2-
Instance-Profile --role-name CodeDeployDemo-EC2-Instance-Profile
```

   IAM インスタンスプロファイルの名前を取得する必要がある場合は、AWS CLI リファレンスの IAM セクションで「list-instance-profiles-for-role」を参照してください。

これで、IAM インスタンスにアタッチする Amazon EC2 インスタンスプロファイルを作成しました。詳細については、Amazon EC2 ユーザーガイド の「Amazon EC2 の IAM ロール」を参照してください。

Amazon EC2 インスタンス(コンソール)の IAM インスタンスプロファイルを作成する

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. IAM コンソールのナビゲーションペインで、[Policies]、[Create policy] の順に選択します。
3. [アクセス許可の指定] ページで、[JSON] を選択します。
4. JSON サンプルコードを削除します。
5. 次のコードを貼り付けます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:Get*",
                "s3:List*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

Note

このポリシーを、Amazon EC2 インスタンスがアクセスする必要のある Amazon S3 バケットにのみ制限することをお勧めします。CodeDeploy エージェントを含む Amazon S3 バケットへのアクセスを必ず許可してください。そうしない場合、CodeDeploy エージェントがインスタンス上にインストールされる、または更新されるときに、エラーが発生する可能性があります。Amazon S3 中の CodeDeploy リソースキットバケットのみへの IAM インスタンスアクセスは許すが、アクセスを防止するバケットのための行を削除するには

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:Get*",
        "s3:List*"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket/*",
        "arn:aws:s3:::aws-codedeploy-us-east-2/*",
        "arn:aws:s3:::aws-codedeploy-us-east-1/*",
        "arn:aws:s3:::aws-codedeploy-us-west-1/*",
        "arn:aws:s3:::aws-codedeploy-us-west-2/*",
        "arn:aws:s3:::aws-codedeploy-ca-central-1/*",
```

```
        "arn:aws:s3:::aws-codedeploy-eu-west-1/*",
        "arn:aws:s3:::aws-codedeploy-eu-west-2/*",
        "arn:aws:s3:::aws-codedeploy-eu-west-3/*",
        "arn:aws:s3:::aws-codedeploy-eu-central-1/*",
        "arn:aws:s3:::aws-codedeploy-eu-central-2/*",
        "arn:aws:s3:::aws-codedeploy-eu-north-1/*",
        "arn:aws:s3:::aws-codedeploy-eu-south-1/*",
        "arn:aws:s3:::aws-codedeploy-eu-south-2/*",
        "arn:aws:s3:::aws-codedeploy-il-central-1/*",
        "arn:aws:s3:::aws-codedeploy-ap-east-1/*",
        "arn:aws:s3:::aws-codedeploy-ap-northeast-1/*",
        "arn:aws:s3:::aws-codedeploy-ap-northeast-2/*",
        "arn:aws:s3:::aws-codedeploy-ap-northeast-3/*",
        "arn:aws:s3:::aws-codedeploy-ap-southeast-1/*",
        "arn:aws:s3:::aws-codedeploy-ap-southeast-2/*",
        "arn:aws:s3:::aws-codedeploy-ap-southeast-3/*",
        "arn:aws:s3:::aws-codedeploy-ap-southeast-4/*",
        "arn:aws:s3:::aws-codedeploy-ap-south-1/*",
        "arn:aws:s3:::aws-codedeploy-ap-south-2/*",
        "arn:aws:s3:::aws-codedeploy-me-central-1/*",
        "arn:aws:s3:::aws-codedeploy-me-south-1/*",
        "arn:aws:s3:::aws-codedeploy-sa-east-1/*"
      ]
    }
  ]
}
```

> **Note**
>
> IAM 認可または CodeDeploy を使用する Amazon Virtual Private Cloud (VPC) エンドポイントを使用したい場合は、アクセス許可を追加する必要があります。詳細については、[Amazon Virtual Private Cloud で CodeDeploy を使う] を参照してください。

6. [次へ] を選択します。
7. [確認および作成] ページで、[ポリシー名] ボックスに「**`CodeDeployDemo-EC2-Permissions`**」と入力します。
8. (オプション) [説明] に、ポリシーの説明を入力します。
9. [Create policy] を選択します。
10. ナビゲーションペインで [Roles] を選択し、続いて [Create role] を選択します。

11. [ユースケース] で、[EC2] ユースケースを選択します。
12. [次へ] を選択します。
13. ポリシーのリストで、作成したポリシー (CodeDeployDemo-EC2-Permissions) の横にあるチェックボックスをオンにします。必要に応じて、検索ボックスを使用してポリシーを見つけます。
14. Systems Manager を使用して CodeDeploy エージェントをインストールまたは設定するには、[AmazonSSMManagedInstanceCore] の横にあるボックスを選択します。この AWS 管理ポリシーにより、インスタンスは Systems Manager サービスコア機能を使用できます。必要に応じて、検索ボックスを使用してポリシーを見つけます。コマンドラインを使用してパブリック Amazon S3 バケットからエージェントをインストールする場合、このポリシーは必要ありません。「CodeDeploy エージェントのインストール」の詳細を確認してください。
15. [次へ] を選択します。
16. [名前、確認、および作成] ページで、[ロール名] にサービスロールの名前 (例えば、**CodeDeployDemo-EC2-Instance-Profile**) を入力し、[ロールを作成] を選択します。

    このサービスロールの説明を、[Role description] ボックスに入力することもできます。

これで、IAM インスタンスにアタッチする Amazon EC2 インスタンスプロファイルを作成しました。詳細については、Amazon EC2 ユーザーガイド の「Amazon EC2 の IAM ロール」を参照してください。

CodeDeploy との製品とサービスの統合

デフォルトでは、CodeDeploy は多数の AWS サービスおよびパートナー製品およびサービスと統合されます。以下の情報は、使用する製品やサービスを統合するための CodeDeploy の設定に役立ちます。

- 他の AWS サービスとの統合
- パートナーの製品とサービスとの統合
- コミュニティから統合の例

他の AWS サービスとの統合

CodeDeploy は、次の AWS サービスと統合されています。

Amazon CloudWatch	Amazon CloudWatch は、 AWS クラウドリソースと、 AWSで実行するアプリケーションのモニタリングサービスです。Amazon CloudWatch を使用してメトリクスの収集と追跡、ログファイルの収集とモニタリング、アラームの設定ができます。CodeDeploy は、次の CloudWatch ツールをサポートしています。 • CloudWatch alarms は、デプロイをモニタリングし、指定したモニタリングメトリクスが CloudWatch アラームルールで指定したしきい値を超えるか下回った場合にデプロイを停止します。アラームモニタリングを使用するには、まず CloudWatch でアラームを設定します。次に、CodeDeploy でアラームをアプリケーションまたはデプロイグループに追加し、アラームがアクティブになったときに停止させます。 詳細はこちら: • CloudWatch Logs アラームの作成

- Amazon CloudWatch Events では、CodeDeploy オペレーションでのインスタンスやデプロイの状態の変更を検出し、対応できます。次に、作成したルールに基づいて CloudWatch Events はデプロイまたはインスタンスが、ルールで指定した状態になると、1 つ以上のターゲットアクションを呼び出します。

  詳細はこちら:

  - Amazon CloudWatch Events を使用したデプロイのモニタリング
- Amazon CloudWatch Logs では、インスタンスに個別にサインインせずに CodeDeploy エージェントによって作成された 3 種類のログをモニタリングします。

  詳細はこちら:

  - CloudWatch Logs コンソールで CodeDeploy ログを表示する

Amazon EC2 Auto Scaling	CodeDeploy は、Amazon EC2 Auto Scaling (Amazon EC2 オートスケーリング) をサポートします。この AWS サービスは、指定した条件に基づいて Amazon EC2 インスタンスを自動的に起動できます。次に例を示します。 • 指定された CPU 使用率の制限を超えた。 • ディスクの読み書き。 • 指定された期間のインバウンドまたはアウトバウンドのネットワークトラフィック。 必要に応じて Amazon EC2 インスタンスグループをスケールアウトできます。さらに CodeDeploy を使用してアプリケーションリビジョンを自動的にデプロイできます。Amazon EC2 Auto Scaling は、Amazon EC2 インスタンスが不要になったときに、それを終了します。 詳細はこちら: • CodeDeploy と Amazon EC2 Auto Scaling の統合 • チュートリアル: CodeDeploy を使用して、Auto Scaling グループにアプリケーションをデプロイします。 • 内部構造: CodeDeploy と Auto Scaling の統合

Amazon Elastic Container Service	CodeDeploy を使用して、Amazon ECSにコンテナ化されたアプリケーションをタスクセットとしてデプロイできます。CodeDeploy は、アプリケーションの更新バージョンを新しい置き換えタスクセットとしてインストールすることで、Blue/Green のデプロイを実行します。CodeDeploy は、元のアプリケーションタスクセットからの本稼働トラフィックを置き換えタスクセットに再ルーティングします。デプロイが正常に完了すると、元のタスクセットは終了します。Amazon ECS の詳細については、「Amazon Elastic Container Service」を参照してください。 Canary 設定、線形設定、または一度にすべての設定を選択して、デプロイ時に更新済みタスクセットにトラフィックを移行する方法を管理できます。Amazon ECS デプロイの詳細については、「Amazon ECS コンピューティングプラットフォームでのデプロイ」を参照してください。

AWS CloudTrail	CodeDeploy は AWS CloudTrail と統合します。このサービスでは、 AWS アカウントで CodeDeploy によって、または CodeDeploy の代わりに行われた API コールをキャプチャし、指定した Amazon S3 バケットにログファイルを配信します。CloudTrail は、CodeDeploy コンソール、 AWS CLIを通した CodeDeploy コマンド、または CodeDeploy API からの API 呼び出しを直接キャプチャします。CloudTrail によって収集されたデータを使用して、以下の情報を判断できます。 • CodeDeploy に行われたリクエスト。 • リクエストが行われたソース IP アドレス。 • 誰がリクエストを行ったか。 • 行われた時刻。 詳細はこちら: • Monitoring Deployments

AWS Cloud9	AWS Cloud9 は、インターネットに接続されたマシンからブラウザのみを使用してコードを記述、実行、デバッグ、デプロイするために使用できるクラウドベースのオンライン統合開発環境 (IDE) です。 AWS Cloud9 には、コードエディタ、デバッガー、ターミナル、および AWS CLI や Git などの重要なツールが含まれています。 • IDE を使用して、GitHub AWS Cloud9 リポジトリにあるコードを実行、デバッグ、構築できます。IDE Environment ウィンドウおよびエディタのタブを使用して、コードの表示、変更、保存を行うことができます。準備ができたら、 AWS Cloud9 ターミナルセッションで Git を使用してコード変更を GitHub リポジトリにプッシュし、 を使用して更新 AWS CodeDeploy をデプロイできます。GitHub AWS Cloud9 で を使用する方法の詳細については、GitHub サンプル AWS Cloud9」を参照してください。 • IDE AWS Cloud9 を使用して AWS Lambda 関数を更新できます。その後、 を使用して AWS CodeDeploy 、トラフィックを AWS Lambda 関数の新しいバージョンに移行するデプロイを作成できます。詳細については、AWS Cloud9 「統合開発環境 (IDE) での AWS Lambda 関数の使用」を参照してください。 詳細については AWS Cloud9、「 とは AWS Cloud9」および「 の開始方法 AWS Cloud9」を参照してください。

AWS CodePipeline	AWS CodePipeline は、継続的な配信プロセスでソフトウェアをリリースするために必要な手順のモデル化、可視化、および自動化に使用できる継続的な配信サービスです。 AWS CodePipeline を使用して、コード変更のたびにサービスが構築、テスト、デプロイを行うよう独自のプロセスを定義できます。例えば、ベータ、ガンマ、本番の 3 つのアプリケーションのデプロイグループがあるとします。ソースコードが変更されるたびに、各デプロイグループに 1 つずつ更新をデプロイするようにパイプラインを設定できます。 CodeDeploy を使用してデプロイ AWS CodePipeline するように を設定できます。 • Amazon EC2 インスタンス、オンプレミスインスタンス、または両方へのコード。 • Serverless AWS Lambda 関数のバージョン。 パイプラインを作成する前または [パイプラインの作成] ウィザードの段階のデプロイアクションで使用できるように、CodeDeploy アプリケーション、デプロイおよびデプロイグループを作成できます。 詳細はこちら: • DevOps 用AWS 入門ガイド - CodePipeline を CodeDeploy で使用して、CodeCommit リポジトリのソースコードを Amazon EC2 インスタンスに継続的に配信およびデプロイする方法について説明します。 • シンプルなパイプラインのチュートリアル (Amazon S3 バケット)

	• シンプルなパイプラインのチュートリアル (CodeCommit リポジトリ) • 4 段階パイプラインのチュートリアル
AWS サーバーレスアプリケーションモデル	AWS サーバーレスアプリケーションモデル (AWS SAM) は、サーバーレスアプリケーションを定義するモデルです。を拡張 CloudFormation して、サーバーレスアプリケーションに必要な関数、Amazon API Gateway APIs、Amazon DynamoDB テーブルを簡単に AWS Lambda 定義できます。 AWS SAM を既に使用している場合は、デプロイ設定を追加して CodeDeploy の使用を開始し、 AWS Lambda アプリケーションのデプロイ中にトラフィックを移行する方法を管理できます。 詳細については、「AWS サーバーレスアプリケーションモデル」を参照してください。
Elastic Load Balancing	CodeDeploy は、Elastic Load Balancing をサポートしています。このサービスでは、着信アプリケーションを複数の Amazon EC2 インスタンス間に分散します。 また、CodeDeploy のデプロイ中、インスタンスが準備完了ではない、現在デプロイ中、または環境の一部として不要になった場合は、ロードバランサーはそのインスタンスへのインターネットトラフィックのルーティングを防止します。 詳細はこちら: • Integrating CodeDeploy with Elastic Load Balancing

トピック

- CodeDeploy と Amazon EC2 Auto Scaling の統合
- CodeDeploy と Elastic Load Balancing の統合

CodeDeploy と Amazon EC2 Auto Scaling の統合

CodeDeploy は、定義した条件に従って Amazon EC2 インスタンスを自動的に起動する AWS サービスである Amazon EC2 Auto Scaling をサポートしています。これらの条件には、指定された時間間隔にわたる CPU 使用率やディスクの読み取りまたは書き込み、インバウンド/アウトバウンドのネットワークトラフィックの制限の超過が含まれます。Amazon EC2 Auto Scaling は、インスタンスが不要になったときに、それを終了します。詳細については、「Amazon EC2 Auto Scaling ユーザーガイド」の「Amazon EC2 Auto Scaling とは」を参照してください。

Amazon EC2 Auto Scaling グループの一部として新しい Amazon EC2 インスタンスが起動されると、CodeDeploy ではリビジョンを新しいインスタンスに自動的にデプロイできます。また、Elastic Load Balancing ロードバランサーに登録された Amazon EC2 Auto Scaling インスタンスを使用して、CodeDeploy でのデプロイを調整できます。詳細については、「Integrating CodeDeploy with Elastic Load Balancing」および「CodeDeploy Amazon EC2 デプロイ用の Elastic Load Balancing でロードバランサーをセットアップする」を参照してください。

> **Note**
>
> 複数のデプロイグループを 1 つの Amazon EC2 Auto Scaling グループに関連付けると、問題が発生する可能性があります。例えば、1 つのデプロイが失敗すると、インスタンスはシャットダウンを開始しますが、実行中の他のデプロイはタイムアウトに 1 時間かかる可能性があります。詳細については、「複数のデプロイグループを 1 つの Amazon EC2 Auto Scaling グループに関連付けることは避ける」そして「内部構造: CodeDeploy と Amazon EC2 Auto Scaling の統合」を参照してください。

トピック

- CodeDeploy アプリケーションを Amazon EC2 Auto Scaling グループにデプロイする
- Auto Scaling スケールインイベント中の終了デプロイの有効化
- Amazon EC2 Auto Scaling と CodeDeploy の連携
- CodeDeploy と Amazon EC2 Auto Scaling でのカスタム AMI の使用

CodeDeploy アプリケーションを Amazon EC2 Auto Scaling グループにデプロイする

Amazon EC2 Auto Scaling グループに CodeDeploy アプリケーションリビジョンをデプロイするには、次の手順を実行します。

1. Amazon EC2 Auto Scaling グループが Amazon S3 での作業を許可する IAM インスタンスプロファイルを作成または検索します。詳細については、「ステップ 4: Amazon EC2 インスタンス用の IAM インスタンスプロファイルを作成する」を参照してください。

   > **Note**
   >
   > また、CodeDeploy を使用して GitHub リポジトリから Amazon EC2 Auto Scaling グループにリビジョンをデプロイすることもできます。Amazon EC2 インスタンスで依然として IAM インスタンスプロファイルが必要な場合でも、GitHub リポジトリからデプロイするときは、プロファイルに追加のアクセス許可は不要です。

2. Amazon EC2 Auto Scaling グループを作成または使用し、起動設定またはテンプレートで、IAM インスタンスプロファイルを指定します。詳細については、「Amazon EC2 インスタンスで実行されるアプリケーションに対する IAM ロール」を参照してください。
3. Amazon EC2 Auto Scaling グループを含むデプロイグループの作成を CodeDeploy に許可するサービスロールを作成または検索します。
4. CodeDeploy でデプロイグループを作成し、Amazon EC2 Auto Scaling グループ名、サービスロール、その他いくつかのオプションを指定します。詳細については、インプレースデプロイ用のデプロイグループを作成する (コンソール)またはインプレースデプロイ用のデプロイグループを作成する (コンソール)を参照してください。
5. CodeDeploy を使用して、Amazon EC2 Auto Scaling グループを含むデプロイグループにリビジョンをデプロイします。

詳細については、「チュートリアル: CodeDeploy を使用して、Auto Scaling グループにアプリケーションをデプロイします。」を参照してください。

Auto Scaling スケールインイベント中の終了デプロイの有効化

終了デプロイは、CodeDeploy デプロイの一種で、Auto Scaling スケールインイベントが発生すると、自動的にアクティブ化されます。CodeDeploy は、Auto Scaling サービスがインスタンスを終了する直前に終了デプロイを実行します。終了デプロイ中は、CodeDeploy は何もデプロイしません。代わりに、ライフサイクルイベントを生成します。これを独自のスクリプトにフックしてカスタムの

シャットダウン機能を有効にすることができます。例えば、ApplicationStop ライフサイクルイベントを、インスタンスの終了前にアプリケーションを正常にシャットダウンするスクリプトにフックできます。

CodeDeploy が終了デプロイ中に生成するライフサイクルイベントのリストについては、「ライフサイクルイベントフックの可用性」を参照してください。

終了デプロイが何らかの理由で失敗した場合、CodeDeploy はインスタンスの終了を続行できるようにします。つまり、CodeDeploy がライフサイクルイベントの完全なセット (または一部) を最後まで実行しなかった場合でも、インスタンスはシャットダウンされます。

終了デプロイを有効にしない場合でも、スケールインイベントが発生すると、Auto Scaling サービスは Amazon EC2 インスタンスを終了しますが、CodeDeploy はライフサイクルイベントを生成しません。

> **Note**
>
> 終了デプロイを有効にするかどうかにかかわらず、CodeDeploy のデプロイ中に Auto Scaling サービスが Amazon EC2 インスタンスを終了すると、Auto Scaling サービスと CodeDeploy サービスによって生成されたライフサイクルイベント間に競合状態が発生する可能性があります。例えば、Auto Scaling サービスで生成した Terminating ライフサイクルイベントは、CodeDeploy デプロイで生成した ApplicationStart イベントをオーバーライドする場合があります。このシナリオでは、Amazon EC2 インスタンスの終了または CodeDeploy のデプロイのいずれかで障害が発生する可能性があります。

CodeDeploy で終了デプロイを実行できるようにするには

- デプロイグループを作成または更新するときに、[Auto Scaling グループに終了フックを追加] チェックボックスをオンにします。手順については、「インプレースデプロイ用のデプロイグループを作成する (コンソール)」または「EC2/オンプレミス Blue/Green デプロイ用のデプロイグループを作成する (コンソール)」を参照してください。

  このチェックボックスを有効にすると、CodeDeploy は、CodeDeploy デプロイグループの作成時または更新時に指定した Auto Scaling グループに Auto Scaling ライフサイクルフックをインストールします。このフックは終了フックと呼ばれ、終了デプロイを有効にします。

終了フックをインストールすると、次のようにスケールイン (終了) イベントが展開されます。

1. Auto Scaling サービス (または単に Auto Scaling) は、スケールアウトイベントが発生する必要があると判断すると、EC2 サービスに連絡して EC2 インスタンスを終了します。
2. EC2 サービスが EC2 インスタンスの終了を開始します。インスタンスは、`Terminating` 状態へ、その後 `Terminating:Wait` 状態へと移行します。
3. `Terminating:Wait` 中、Auto Scaling は、CodeDeploy によってインストールされた終了フックを含め、Auto Scaling グループにアタッチされたすべてのライフサイクルフックを実行します。
4. 終了フックは、CodeDeploy がポーリングする Amazon SQS キューに通知を送信します。
5. 通知を受信すると、CodeDeploy はメッセージを解析し、いくつかの検証を実行して、終了デプロイを実行します。
6. 終了デプロイの実行中、CodeDeploy は 5 分ごとに Auto Scaling にハートビートを送信し、インスタンスがまだ処理中であることを知らせます。
7. これまでのところ、EC2 インスタンスは `Terminating:Wait` 状態 (Auto Scaling グループのウォームプールを有効にしている場合は、`Warmed:Pending:Wait` 状態) のままです。
8. デプロイが完了すると、デプロイの成否にかかわらず、CodeDeploy は Auto Scaling に対して EC2 終了プロセスを `CONTINUE` するよう指示します。

Amazon EC2 Auto Scaling と CodeDeploy の連携

Auto Scaling グループを含むように CodeDeploy デプロイグループを作成または更新すると、CodeDeploy は CodeDeploy サービスロールを使用して Auto Scaling グループにアクセスし、Auto Scaling ライフサイクルフックを Auto Scaling グループにインストールします。

> **Note**
>
> Auto Scaling ライフサイクルフックは、このガイドの「AppSpec の「hooks」セクション」で説明している、CodeDeploy によって生成されるライフサイクルイベント (ライフサイクルイベントフックとも呼ばれます) とは異なります。

CodeDeploy がインストールする Auto Scaling ライフサイクルフックは次のとおりです。

- 起動フック — このフックは、Auto Scaling スケールアウトイベントが進行中であり、起動デプロイを開始する必要があることを CodeDeploy に通知します。

  起動デプロイ中、CodeDeploy は、次のことを行います。

  - アプリケーションのリビジョンを、スケールアウトされたインスタンスにデプロイします。

- デプロイの進行状況を示すライフサイクルイベントを生成します。これらのライフサイクルイベントを独自のスクリプトにフックして、カスタム起動機能を有効にすることができます。詳細については、「ライフサイクルイベントフックの可用性」の表を参照してください。

  起動フックおよび関連する起動デプロイは常に有効になっており、無効にすることはできません。

- 終了フック — このオプションのフックは、Auto Scaling スケールインイベントが進行中であり、終了デプロイを開始する必要があることを CodeDeploy に通知します。

  終了デプロイ中、CodeDeploy はインスタンスのシャットダウンの進行状況を示すライフサイクルイベントを生成します。詳細については、「Auto Scaling スケールインイベント中の終了デプロイの有効化」を参照してください。

トピック

- CodeDeploy でインストールしたライフサイクルフックは、どのように使用されますか?
- CodeDeploy による Amazon EC2 Auto Scaling グループの名前の設定方法
- カスタムライフサイクルフックイベントの実行順序
- デプロイ中のスケールアップイベント
- デプロイ中のスケールインイベント
- cfn-init AWS CloudFormation スクリプトのイベントの順序

CodeDeploy でインストールしたライフサイクルフックは、どのように使用されますか?

起動ライフサイクルフックと終了ライフサイクルフックをインストールすると、CodeDeploy の Auto Scaling グループのスケールアウトイベントとスケールインイベントでそれぞれ使用されます。

スケールアウト (起動) イベントは次のように展開されます。

1. Auto Scaling サービス (または単に Auto Scaling) は、スケールアウトイベントが発生する必要があると判断し、EC2 サービスに連絡して新しい EC2 インスタンスを起動します。
2. EC2 サービスが、新しい EC2 インスタンスを起動します。インスタンスは、`Pending` 状態へ、その後 `Pending:Wait` 状態へと移行します。
3. `Pending:Wait` 中である場合、Auto Scaling は、Auto Scaling グループにアタッチされたすべてのライフサイクルフック (CodeDeploy によってインストールされた起動フックを含む) を実行します。
4. 起動フックは、CodeDeploy がポーリングする Amazon SQS キューに通知を送信します。

5. 通知を受信すると、CodeDeploy はメッセージを解析し、いくつかの検証を実行して、起動デプロイを開始します。
6. 起動デプロイの実行中、CodeDeploy は 5 分ごとに Auto Scaling にハートビートを送信し、インスタンスがまだ処理中であることを知らせます。
7. 今までのところ、EC2 インスタンスはなお `Pending:Wait` の状態です。
8. デプロイが完了すると、デプロイが成功したか失敗したかによって、CodeDeploy は Auto Scaling に EC2 起動プロセスを `CONTINUE` または `ABANDON` するよう指示を出します。
   - CodeDeploy が `CONTINUE` を指示する場合、Auto Scaling は他のフックの完了を待機するか、インスタンスを `Pending:Proceed` 状態に、その後 `InService` 状態にします。
   - CodeDeploy が `ABANDON` を指示する場合、Auto Scaling は EC2 インスタンスを終了し、また必要な数のインスタンスを満たす必要がある場合には、[Auto Scaling の 希望容量 の設定] で定義されているように、起動手順を再開します。

スケールイン (終了) イベントは次のように展開されます。

「Auto Scaling スケールインイベント中の終了デプロイの有効化」を参照してください。

CodeDeploy による Amazon EC2 Auto Scaling グループの名前の設定方法

EC2/オンプレミスコンピューティングプラットフォームでの Blue/Green のデプロイ中に代替 (Green) 環境にインスタンスを追加するには 2 つの方法があります。

- 既存のインスタンス、または手動で作成したインスタンスを使用します。
- 指定した Amazon EC2 Auto Scaling グループの設定を使用して、新しい Amazon EC2 Auto Scaling グループにインスタンスを定義および作成します。

2 番目のオプションを選択した場合は、CodeDeploy によって Amazon EC2 Auto Scaling グループがプロビジョンされます。以下の規則を使用して、グループに名前を付けます。

```
CodeDeploy_deployment_group_name_deployment_id
```

例えば、ID が「10」のデプロイによって「alpha-deployments」というデプロイグループがデプロイされる場合、 プロビジョンされる Amazon EC2 Auto Scaling グループの名前は `CodeDeploy_alpha-deployments_10` になります。詳細については、EC2/オンプレミス Blue/Green デプロイ用のデプロイグループを作成する (コンソール) および「GreenFleetProvisioningOption」を参照してください。

カスタムライフサイクルフックイベントの実行順序

CodeDeploy がデプロイする先の Amazon EC2 Auto Scaling グループに独自のライフサイクルフックを追加できます。ただし、カスタムライフサイクルフックイベントが実行される順序は、デフォルトの CodeDeploy デプロイライフサイクルイベントに対して事前に決定できません。例えば、Amazon EC2 Auto Scaling グループに `ReadyForSoftwareInstall` というカスタムライフサイクルフックを追加すると、それが実行されるのが CodeDeploy のデフォルトデプロイライフサイクルの最初のイベントの前であるか、最後のイベントの後であるか、事前に知ることはできません。

Amazon EC2 Auto Scaling グループにカスタムライフサイクルフックを追加する方法については、[Amazon EC2 Auto Scaling ユーザーガイド] の「ライフサイクルフックの追加」を参照してください。

デプロイ中のスケールアップイベント

デプロイ中に Auto Scaling スケールアウトイベントが発生すると、新しいインスタンスが更新されて、最新のアプリケーションリビジョンではなく、前回デプロイしたアプリケーションリビジョンが反映されます。デプロイが成功すると、古いインスタンスと新しくスケールアウトされたインスタンスは異なるアプリケーションリビジョンを反映します。古いリビジョンを持つインスタンスを最新の状態にするために、CodeDeploy は (最初のデプロイの直後に) 追加のデプロイを自動的に開始し、古いインスタンスを更新します。このデフォルトの動作を変更して、古い EC2 インスタンスが古いリビジョンに残るようにする場合は、「Automatic updates to outdated instances」を参照してください。

デプロイの実行中に Amazon EC2 Auto Scaling スケールアウトプロセスを中断する場合は、CodeDeploy でのロードバランシングに使用される `common_functions.sh` スクリプトの設定を通して行うことができます。`HANDLE_PROCS=true` の場合、デプロイ中は以下の Auto Scaling イベントが自動的に中断されます。

- AZRebalance
- AlarmNotification
- ScheduledActions
- ReplaceUnhealthy

> **Important**
>
> この機能をサポートするのは、CodeDeployDefault.OneAtATime のデプロイ設定のみです。

HANDLE_PROCS=true を使用して Amazon EC2 Auto Scaling 使用時のデプロイ問題を避ける方法の詳細については、GitHub の「aws-codedeploy」で「Important notice about handling AutoScaling processes」を参照してください。

デプロイ中のスケールインイベント

Auto Scaling グループで CodeDeploy デプロイの進行中に Auto Scaling グループがスケールインを開始すると、終了プロセス (CodeDeploy 終了デプロイライフサイクルイベントを含む) と終了インスタンスの他の CodeDeploy ライフサイクルイベントとの間で競合状態が発生する可能性があります。すべての CodeDeploy ライフサイクルイベントが完了する前にインスタンスを終了すると、この特定のインスタンスへのデプロイが失敗する場合があります。また、デプロイ設定で [正常なホストの最小数] をどのように設定したかによって、CodeDeploy デプロイ全体が失敗する場合と失敗しない場合があります。

cfn-init AWS CloudFormation スクリプトのイベントの順序

cfn-init (または cloud-init) を使用して新しくプロビジョニングした Linux ベースのインスタンスでスクリプトを実行する場合、インスタンスの開始後に発生するイベントの順序を厳密に制御しないと、デプロイは失敗することがあります。

次の順序に従う必要があります。

1. 新しくプロビジョニングしたインスタンスが開始する。
2. すべての cfn-init ブートストラップスクリプトが最後まで実行する。
3. CodeDeploy エージェントが開始します。
4. 最新のアプリケーションリビジョンがインスタンスにデプロイされる。

イベントの順序を慎重に制御しないと、CodeDeploy エージェントはすべてのスクリプトの実行が終了する前にデプロイを開始する可能性があります。

イベントの順序を制御するには、以下のベストプラクティスのいずれかを使用します。

- cfn-init スクリプトを通じて CodeDeploy エージェントをインストールし、他のすべてのスクリプトの後に配置します。
- CodeDeploy エージェントをカスタム AMI に含め、cfn-init スクリプトを使用して起動します。エージェントは、他のすべてのスクリプトの後に配置します。

`cfn-init` の使用方法については、AWS CloudFormation ユーザーガイド の「cfn-init」を参照してください。

CodeDeploy と Amazon EC2 Auto Scaling でのカスタム AMI の使用

Amazon EC2 Auto Scaling グループで新しい Amazon EC2 インスタンスを起動するとき、基本 AMI を指定するには 2 つのオプションがあります。

- CodeDeploy エージェントがインストールされている基本カスタム AMI を 指定できます。エージェントが既にインストールされているため、このオプションはほかのオプションよりも迅速に新しい Amazon EC2 インスタンスを起動します。ただし、このオプションでは (特に CodeDeploy エージェントが古い場合)、Amazon EC2 インスタンスの初回デプロイが失敗する可能性が大きくなります。このオプションを選択した場合は、基本カスタム AMI の CodeDeploy エージェントを定期的に更新することをお勧めします。
- CodeDeploy エージェントがインストールされていない基本 AMI を指定できます。また、Amazon EC2 Auto Scaling グループ内で新しいインスタンスが起動されるたびにエージェントをインストールすることもできます。このオプションでは、ほかのオプションよりも新しい Amazon EC2 インスタンスの起動が遅くなりますが、インスタンスの最初のデプロイが成功する可能性は大きくなります。このオプションは、CodeDeploy エージェントの最新バージョンを使用します。

CodeDeploy と Elastic Load Balancing の統合

CodeDeploy のデプロイ中、インスタンスが準備完了ではない、現在デプロイ中、または環境の一部として不要になった場合は、ロードバランサーはそのインスタンスへのインターネットトラフィックのルーティングを防止します。ただし、ロードバランサーの正確な役割は、Blue/Green デプロイで使用されるかインプレースデプロイで使用されるかによって異なります。

> **Note**
>
> Elastic Load Balancing ロードバランサーの使用は Blue/Green デプロイでは必須、インプレースデプロイでは任意です。

Elastic Load Balancing のタイプ

Elastic Load Balancing は、CodeDeploy デプロイで使用可能な、Classic Load Balancer、Application Load Balancer、Network Load Balancer の 3 つのタイプのロードバランサーを提供します。

Classic Load Balancer

ルーティングおよび負荷分散を、トランスポートレイヤー (TCP/SSL) またはアプリケーションレイヤー (HTTP/HTTPS) のいずれかで行います。VPC がサポートされています。

> **Note**
>
> Classic Load Balancer は Amazon ECS デプロイではサポートされていません。

Application Load Balancer

ルーティングと負荷分散をアプリケーションレイヤー (HTTP/HTTPS) で行い、パスベースのルーティングをサポートしています。Virtual Private Cloud (VPC) 内の EC2 の各インスタンスまたはコンテナインスタンスのポートにリクエストをルーティングできます。

> **Note**
>
> Application Load Balancer のターゲットグループには、EC2 インスタンスでのデプロイの場合は `instance`、および Fargate デプロイの場合は `IP` のターゲットタイプがなければなりません。詳細については、「ターゲットタイプ」を参照してください。

Network Load Balancer

パケットのコンテンツからではなく、TCP パケットヘッダーから抽出されたアドレス情報に基づいて、トランスポートレイヤー (TCP/UDP Layer-4) でルーティングと負荷分散を行います。Network Load Balancer は、ロードバランサーの有効期間中、トラフィックバーストを処理し、クライアントの出典 IP を保持して、固定 IP を使用します。

Elastic Load Balancing ロードバランサーの詳細は、以下のトピックを参照してください。

- Elastic Load Balancing とは?
- 「Classic Load Balancer とは ?」
- Application Load Balancer とは?
- Network Load Balancer とは?

Blue/Green デプロイ

Elastic Load Balancing ロードバランサーの背後でインスタンストラフィックを再ルーティングすることは CodeDeploy Blue/Green デプロイの基本です。

Blue/Green デプロイの場合、ロードバランサーは、最新のアプリケーションリビジョンのデプロイ先であるデプロイグループの新しいインスタンス (置き換え先環境) に対しては、指定したルールに基づくトラフィックのルーティングを許可し、前回のアプリケーションリビジョンの実行元である古いインスタンス (元の環境) からはトラフィックをブロックします。

置き換え先環境のインスタンスが 1 つ以上のロードバランサーに登録されると、置き換え元環境のインスタンスは登録解除され、終了可能になります。

ブルー/グリーンデプロイでは、デプロイグループで 1 つ以上の Classic Load Balancer、Application Load Balancer ターゲットグループ、または Network Load Balancer ターゲットグループを指定できます。CodeDeploy コンソールまたは を使用して AWS CLI 、ロードバランサーをデプロイグループに追加します。

Blue/Green デプロイにおけるロードバランサーの使用に関する詳細については、以下のトピックを参照してください。

- CodeDeploy Amazon EC2 デプロイ用の Elastic Load Balancing でロードバランサーをセットアップする
- Blue/Green デプロイ (コンソール) のアプリケーションを作成します。
- EC2/オンプレミス Blue/Green デプロイ用のデプロイグループを作成する (コンソール)

インプレースデプロイ

インプレースデプロイ中は、ロードバランサーにより、デプロイ先のインスタンスに対するインターネットトラフィックのルーティングがブロックされ、そのインスタンスへのデプロイが完了した時点でインスタンスに対するトラフィックのルーティングが再開されます。

インプレースデプロイ中にロードバランサーが使用されないと、インターネットトラフィックはデプロイプロセス中に依然としてインスタンスにルーティングされる場合があります。その結果、お客様に表示されるウェブアプリケーションが破損していたり、不完全であったり、古いものであったりする可能性があります。インプレースデプロイで Elastic Load Balancing ロードバランサーを使用する場合、デプロイグループのインスタンスはロードバランサーから登録解除され、最新のアプリケーションリビジョンに更新されてから、デプロイが成功した後で同じデプロイグループの一部としてロードバランサーに再登録されます。CodeDeploy は、ロードバランサーのバックグラウンドでイン

スタンスが正常になるまで最大 1 時間待機します。待機期間中にロードバランサーによってインスタンスが正常とマークされていない場合、CodeDeploy はデプロイ設定に基づいて、次のインスタンスに移動するか、デプロイに失敗します。

インプレースデプロイでは、1 つ以上の Classic Load Balancer、Application Load Balancer ターゲットグループ、または Network Load Balancer ターゲットグループを指定できます。ロードバランサーをデプロイグループの設定の一部として指定できます。または CodeDeploy が提供するスクリプトを使用してロードバランサーを実装できます。

デプロイグループを使用してインプレースデプロイのロードバランサーを指定する

デプロイグループにロードバランサーを追加するには、CodeDeploy コンソールまたは を使用します AWS CLI。インプレースデプロイでロードバランサーをデプロイグループで指定する詳細については、次のトピックを参照してください。

- インプレースデプロイ (コンソール) 用のアプリケーションを作成
- インプレースデプロイ用のデプロイグループを作成する (コンソール)
- CodeDeploy Amazon EC2 デプロイ用の Elastic Load Balancing でロードバランサーをセットアップする

スクリプトを使用してインプレースデプロイのロードバランサーを指定する

次の手順のステップに従ってデプロイライフサイクルスクリプトを使用し、インプレースデプロイのロードバランシングをセットアップします。

> **Note**
>
> CodeDeployDefault.OneAtATime 設定は、スクリプトを使用してインプレイスデプロイ用のロードバランサーを設定するときのみ使用します。同時実行はサポートされておらず、CodeDeployDefault.OneAtATime 設定によりスクリプトの直列実行が確実になります。デプロイ設定の詳細については、CodeDeploy でデプロイ設定を使用する を参照してください。

GitHub の CodeDeploy サンプルリポジトリでは、CodeDeployElastic Load Balancing ロードバランサーの使用に対応できる手順とサンプルを提供します。これらのレポジトリには、開始するのに必要なすべてのコードを提供する 3 つのサンプルスクリプト、`register_with_elb.sh`、`deregister_from_elb.sh`、および `common_functions.sh`

が含まれます。これらの 3 つのスクリプトのプレースホルダーを編集して、appspec.yml ファイルからこれらのスクリプトを参照します。

Elastic Load Balancing ロードバランサーに登録された Amazon EC2 インスタンスを使用して CodeDeploy でインプレースデプロイをセットアップするには、以下を実行します。

1. インプレースデプロイで使用するロードバランサーのタイプのサンプルをダウンロードします。
   - Classic Load Balancer
   - Application Load Balancer または Network Load Balancer (同じスクリプトをいずれのタイプにも使用可能)
2. ターゲットの各 Amazon EC2 インスタンスに AWS CLI がインストールされていることを確認します。
3. ターゲットの各 Amazon EC2 インスタンスで、IAM インスタンスプロファイルに少なくとも elasticloadbalancing:* および autoscaling:* アクセス許可がアタッチされていることを確認します。
4. アプリケーションのソースコードディレクトリにデプロイライフサイクルイベントのスクリプト (register_with_elb.sh、deregister_from_elb.sh、および common_functions.sh) を含めます。
5. アプリケーションリビジョンの appspec.yml で、ApplicationStart イベント時に register_with_elb.sh スクリプトを、および ApplicationStop イベント時に deregister_from_elb.sh スクリプトを実行する CodeDeploy の説明を提供します。
6. インスタンスが Amazon EC2 Auto Scaling グループの一部である場合、このステップは省略できます。

   common_functions.sh スクリプトで:

   - Classic Load Balancer を使用している場合、ELB_LIST="" で Elastic Load Balancing ロードバランサーの名前を指定し、ファイルの他のデプロイ設定に必要な変更を加えます。
   - Application Load Balancer または Network Load Balancer を使用している場合は、TARGET_GROUP_LIST="" で Elastic Load Balancing ターゲットグループ名を指定し、ファイルの他のデプロイ設定に必要な変更を加えます。
7. アプリケーションのソースコード appspec.yml およびデプロイライフサイクルイベントのスクリプトをアプリケーションリビジョンにバンドルしてから、リビジョンをアップロードします。Amazon EC2 インスタンスにリビジョンをデプロイします。デプロイの間に、デプロイライフサイクルイベントのスクリプトは、Amazon EC2 インスタンスをロードバランサーから登

録解除して、接続がドレインするまで待機し、デプロイが完了してから Amazon EC2 インスタンスをロードバランサーに再登録します。

パートナーの製品とサービスとの統合

CodeDeploy には、次のパートナー製品とサービスとの統合が組み込まれています。

Ansible	すでに一連の Ansible プレイブックがあり、実行する場所が必要なだけの場合、Ansible と CodeDeploy のテンプレートは、いくつかのシンプルなデプロイフックによって Ansible をローカルデプロイインスタンスで使用し、プレイブックを実行できることを示します。すでにインベントリを構築して維持するためのプロセスが存在する場合、CodeDeploy エージェントのインストールと実行に使用できる Ansible モジュールもあります。 詳細はこちら: • Ansible と CodeDeploy
Atlassian - Bamboo と Bitbucket	Bamboo の CodeDeploy タスクは AppSpec ファイルを含むディレクトリを.zip ファイルに圧縮、ファイルを Amazon S3 にアップロードし、CodeDeploy アプリケーションで提供されている設定に従ってデプロイを開始します。 Atlassian Bitbucket での CodeDeploy のサポートにより、Bitbucket UI からコードを直接 Amazon EC2 インスタンスにプッシュしたり、オンデマンドで任意のデプロイグループにプッシュしたりできます。これは、Bitbucket リポジトリのコードを更新した後、手動でのデプロイプロセスを実行するために、継続的インテグレーション (CI) プラットフォーム、または

	Amazon EC2 インスタンスにサインインする必要がないことを意味します。 詳細はこちら: • Bamboo の CodeDeploy タスクの使用 • Atlassian Bitbucket による CodeDeploy のサポートの発表
Chef	AWS には、Chef と CodeDeploy を統合するための 2 つのテンプレートサンプルが用意されています。1 つは CodeDeploy エージェントをインストールして起動する Chef クックブックです。これにより CodeDeploy を使用中に Chef でのホストインフラストラクチャの管理を継続することができます。2 番目のサンプルテンプレートでは、CodeDeploy を使用して、各ノードで chef-solo を使用してクックブックとレシピの実行を調整する方法を示します。 詳細はこちら: • シェフと CodeDeploy
CircleCI	CircleCI では、自動化テスト、継続的な統合、およびデプロイのツールセットを提供します。CircleCI AWS で使用する IAM ロールをで作成し、circle.yml ファイルでデプロイパラメータを設定したら、CodeDeploy で CircleCI を使用してアプリケーションリビジョンを作成し、Amazon S3 バケットにアップロードしてから、デプロイを開始してモニタリングできます。 詳細はこちら: • CircleCI Orb を使用してアプリケーションを AWS CodeDeployにデプロイする方法

CloudBees	ビルド後のアクションとして、CloudBees DEV@cloud で利用可能な CodeDeploy Jenkins プラグインを使用できます。例えば、継続的な配信パイプラインの終了時に、サーバー群にアプリケーションリビジョンをデプロイするために使用できます。 詳細はこちら: • CodeDeploy Jenkins プラグインが DEV@cloud で利用可能に
Codeship	Codeship を使用して、CodeDeploy 経由でアプリケーションリビジョンをデプロイできます。Codeship UI を使用して、ブランチのデプロイパイプラインに CodeDeploy を追加できます。 詳細はこちら: • CodeDeploy へのデプロイ • Codeship での CodeDeploy との統合
GitHub	CodeDeploy を使用して、GitHub リポジトリからアプリケーションリビジョンをデプロイできます。そのリポジトリのソースコードが変更されるたびに、GitHub リポジトリからデプロイをトリガーすることもできます。 詳細はこちら: • GitHub と CodeDeploy との統合 • チュートリアル: CodeDeploy を使用して、GitHub からアプリケーションをデプロイします。 • CodeDeploy を使用した GitHub からの自動的なデプロイ

HashiCorp Consul	オープンソースの HashiCorp Consul ツールを使用して、CodeDeploy でアプリケーションをデプロイする時のアプリケーション環境の状態と安定性を確実にします。Consul を使用して、デプロイ中に検出されるアプリケーションを登録し、アプリケーションおよびノードをメンテナンスモードでデプロイから除外し、ターゲットのインスタンスに異常が生じた場合はデプロイを停止します。 詳細はこちら: • HashiCorp Consul を使用した CodeDeploy デプロイ
Jenkins	CodeDeploy Jenkins プラグインでは、Jenkins プロジェクトのビルド後のステップを示します。ビルドに成功すると、ワークスペースが圧縮されて Amazon S3 にアップロードされ、新しいデプロイが開始されます。 詳細はこちら: • CodeDeploy Jenkins プラグイン • CodeDeploy の Jenkins プラグインのセットアップ

Puppet Labs	AWS には、Puppet と CodeDeploy のサンプルテンプレートが用意されています。最初のテンプレートは CodeDeploy エージェントをインストールして起動する Puppet モジュールです。これにより CodeDeploy を使用中に Puppet でのホストインフラストラクチャの管理を継続することができます。2 番目のサンプルテンプレートでは、CodeDeploy を使用してモジュールとマニフェストの実行を、各ノードのマスターレス puppet で調整する方法を示します。 詳細はこちら: • Puppet と CodeDeploy
SaltStack	SaltStack のインフラストラクチャを CodeDeploy と統合できます。CodeDeploy モジュールを使用して、ミニオンズに CodeDeploy エージェントをインストールして実行できます。または、いくつかのシンプルなデプロイフックを使用して、CodeDeploy で Salt States の実行を調整できます。 詳細はこちら: • SaltStack と CodeDeploy

TeamCity	CodeDeploy Runner プラグインを使用して、TeamCity からアプリケーションを直接デプロイできます。このプラグインは、アプリケーションリビジョンを準備して Amazon S3 バケットにアップロードし、CodeDeploy アプリケーションでリビジョンを登録し、CodeDeploy のデプロイを作成します。必要に応じて、デプロイが完了するまで待つ TeamCity のビルドステップを追加します。 詳細はこちら: • CodeDeploy ランナー (ダウンロード) • CodeDeploy ランナープラグイン (ドキュメント)
Travis CI	Travis CI を設定して、ビルドが成功した後、CodeDeploy でデプロイメントをトリガーできます。 詳細はこちら: • Travis CI と CodeDeploy デプロイ

トピック

- GitHub と CodeDeploy との統合

GitHub と CodeDeploy との統合

CodeDeploy は、ウェブベースのコードホストおよび共有サービスである GitHub をサポートします。CodeDeploy は、GitHub リポジトリまたは Amazon S3 バケットに格納されているアプリケーションリビジョンをインスタンスにデプロイできます。CodeDeploy では、EC2 およびオンプレミスデプロイのみ GitHub がサポートされています。

トピック

- GitHub からの CodeDeploy リビジョンのデプロイ

- CodeDeploy を使用した GitHub 動作

GitHub からの CodeDeploy リビジョンのデプロイ

GitHub リポジトリからインスタンスへアプリケーションリビジョンをデプロイするには。

1. CodeDeploy およびデプロイする Amazon EC2 インスタンスタイプと互換性があるリビジョンを作成します。

   互換性のあるリビジョンを作成するには、CodeDeploy のリビジョンを計画する および CodeDeploy 用のアプリケーション仕様ファイルをリビジョンに追加 の手順に従います。
2. GitHub のアカウントを使用して、GitHub リポジトリにリビジョンを追加します。

   GitHub のアカウントを作成するには、「GitHub への参加」を参照してください。GitHub リポジトリを作成するには、「Repo の作成」を参照してください。
3. CodeDeploy コンソールのデプロイの作成ページまたは AWS CLI create-deployment コマンドを使用して、GitHub リポジトリから CodeDeploy デプロイで使用するように設定されたターゲットインスタンスにリビジョンをデプロイします。

   create-deployment コマンドを呼び出す場合、最初にコンソールの [デプロイの作成] ページを使用して、指定したアプリケーションの推奨 GitHub アカウントの代わりに GitHub を操作するためのアクセス許可を CodeDeploy に付与します。これを行う必要があるのは、アプリケーションごとに 1 度だけです。

   [Create deployment] ページを使用して、GitHub レポジトリからデプロイする方法については、「CodeDeploy でデプロイを作成する」を参照してください。

   create-deployment コマンドを呼び出して GitHub レポジトリからデプロイする方法については、「EC2/ オンプレミスコンピューティングプラットフォームのデプロイ作成 (CLI)」を参照してください。

   CodeDeploy デプロイで使用するためインスタンスを準備する方法については、「CodeDeploy のためにインスタンスを用いた操作」を参照してください。

詳細については、「チュートリアル: CodeDeploy を使用して、GitHub からアプリケーションをデプロイします。」を参照してください。

CodeDeploy を使用した GitHub 動作

トピック

- CodeDeploy のアプリケーションを使用した GitHub の認証
- プライベートおよびパブリックな GitHub リポジトリとの CodeDeploy のやり取り
- CodeDeploy と組織マネージド型の GitHub リポジトリのやり取り
- CodeDeploy を使用して CodePipeline から自動的にデプロイする

CodeDeploy のアプリケーションを使用した GitHub の認証

GitHub を操作するためのアクセス許可を CodeDeploy に付与すると、GitHub アカウントとアプリケーションの間の関連付けが CodeDeploy に格納されます。GitHub の他のアカウントにアプリケーションをリンクできます。また、GitHub を操作するための CodeDeploy に対するアクセス許可を取り消すこともできます。

GitHub アカウントを CodeDeploy のアプリケーションにリンクするには

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで [デプロイ] を展開し、[アプリケーション] を選択します。
3. 別の GitHub アカウントにリンクするアプリケーションを選択します。
4. アプリケーションにデプロイグループがない場合は、[デプロイグループの作成] を選択してグループを作成します。詳細については、「CodeDeploy でデプロイグループを作成する」を参照してください。次のステップで [デプロイの作成] を選択するには、デプロイグループが必要です。
5. [デプロイ] から、[デプロイの作成] を選択します。

   > **Note**
   >
   > 新しいデプロイを作成する必要はありません。これは現在 GitHub の他のアカウントをアプリケーションにリンクする唯一の方法です。

6. [デプロイ設定] の [リビジョンタイプ] で、[アプリケーションは GitHub に格納されています] を選択します。
7. 次のいずれかを行います。
   - GitHub アカウントへの AWS CodeDeploy アプリケーションの接続を作成するには、別のウェブブラウザタブで GitHub からサインアウトします。[GitHub token name (GitHub トークン名)] に、この接続を識別する名前を入力し、[GitHub に接続] を選択します。アプリケーションの GitHub を操作することを CodeDeploy に許可するよう求めるメッセージがウェブページに表示されます。ステップ 10 に進みます。
   - 作成済みの接続を使用するには、その名前を [GitHub token name (GitHub トークン名)] で選択してから [GitHub に接続] を選択します。ステップ 8 に進みます。
   - 別の GitHub アカウントへの接続を作成するには、ウェブブラウザの別のタブで GitHub からサインアウトします。[GitHub token name (GitHub トークン名)] に、接続を識別する名前を入力し、[GitHub に接続] を選択します。アプリケーションの GitHub を操作することを CodeDeploy に許可するよう求めるメッセージがウェブページに表示されます。ステップ 10 に進みます。
8. GitHub にまだサインインしていない場合は、[Sign in] ページの手順に従い、アプリケーションをリンクする GitHub アカウントにサインインします。
9. [Authorize application] を選択します。GitHub は指定したアプリケーションの署名 GitHub アカウントの代わりに、CodeDeploy に GitHub を操作するアクセス許可を付与します。
10. デプロイを作成しない場合は、[Cancel] を選択します。

CodeDeploy が GitHub を操作するためのアクセス許可を取り消すには

1. AWS CodeDeploy アクセス許可を取り消したい GitHub アカウントの認証情報を使用して、GitHub にサインインします。
2. GitHub の [Applications] ページを開いて、承認されたアプリケーションのリストで [CodeDeploy] を見つけ、アプリケーションの承認取り消しの GitHub の手順に従います。

プライベートおよびパブリックな GitHub リポジトリとの CodeDeploy のやり取り

CodeDeploy は、プライベートおよびパブリックの GitHub リポジトリからのアプリケーションのデプロイをサポートします。ユーザーに代わって GitHub にアクセスするアクセス許可を CodeDeploy に付与した場合、CodeDeploy は、そのユーザーの GitHub アカウントがアクセス権を持つすべてのプライベート GitHub リポジトリに対して読み取り/書き込みアクセス権を持ちます。ただ

し、CodeDeploy は、GitHub リポジトリからのみ読み込みます。プライベート GitHub のリポジトリに書き込むことはありません。

CodeDeploy と組織マネージド型の GitHub リポジトリのやり取り

デフォルトでは、組織によって管理される GitHub リポジトリ (アカウントのプライベートまたはパブリックリポジトリとは対照的に) は、CodeDeploy を含め、サードパーティアプリケーションへのアクセスを許可しません。デプロイは、組織によるサードパーティアプリケーション制限が GitHub で有効化され、GitHub リポジトリからコードをデプロイしようとすると失敗します。この問題を解決する 2 つの方法があります。

- 組織のメンバーとして、組織の所有者に CodeDeploy へのアクセスを承認するように要求できます。このアクセスをリクエストする手順は、個々のアカウントに対して、すでに CodeDeploy を認証済みかどうかによって異なります。
  - ユーザーのアカウントに CodeDeploy への認証済みアクセス権がある場合は、「組織への認証済みアプリケーションの承認リクエスト」を参照してください。
  - ユーザーのアカウントに CodeDeploy への承認済みアクセス権がまだない場合は、「組織へのサードパーティアプリケーションの承認リクエスト」を参照してください。
- 組織の所有者は、組織に対するサードパーティアプリケーションの制限を無効にできます。詳細については、「組織に対するサードパーティアプリケーションの制限の無効化」を参照してください。

詳細については、「サードパーティアプリケーションの制限について」を参照してください。

CodeDeploy を使用して CodePipeline から自動的にデプロイする

ソースコードが変わるたびに CodePipeline からデプロイをトリガーできます。詳細については、「CodePipeline」を参照してください。

コミュニティから統合の例

以下のセクションは、ブログの投稿や記事、およびコミュニティで提供されている例へのリンクです。

> **Note**
>
> これらのリンクは、情報提供のみを目的として提供されており、包括的なリストまたはコンテンツの例の内容を推奨するものではありません。 AWS は、外部コンテンツの内容または正確性について責任を負いません。

ブログ記事

- での CodeDeploy プロビジョニングの自動化 CloudFormation

  CloudFormationを使用して、CodeDeploy でアプリケーションのデプロイをプロビジョニングする方法について説明します。

  2016 年 1 月投稿

- AWS Toolkit for Eclipse CodeDeploy との統合 (パート 1)

  AWS Toolkit for Eclipse CodeDeploy との統合 (パート 2)

  AWS Toolkit for Eclipse CodeDeploy との統合 (パート 3)

  Java 開発者が CodeDeploy plugin for Eclipse を使用して、ウェブアプリケーションを Eclipse 開発環境 AWS から直接 にデプロイする方法について説明します。

  2015 年 2 月投稿

- CodeDeploy を使用した GitHub からの自動的なデプロイ

  GitHub から CodeDeploy への自動デプロイを使用して、ソース管理からテストまたは本番環境に至るまでのエンドツーエンドのパイプラインを作成する方法をご覧ください。

  2014 年 12 月投稿

CodeDeploy: チュートリアル

このセクションには、CodeDeploy の使用方法の学習に役立ついくつかのチュートリアルが含まれています。

以下のチュートリアルの手順では、ファイルの格納場所 (例: c:\temp)、およびバケット、サブフォルダ、またはファイルに付ける名前 (例: それぞれ、amzn-s3-demo-bucket、HelloWorldApp、CodeDeployDemo-EC2-Trust.json) を提案していますが、それらを使用する必要はありません。手順を実行する際に、ファイルの場所と名前は必ず置き換えてください。

トピック

- チュートリアル: Amazon EC2 インスタンスに WordPress をデプロイする (Amazon Linux または Red Hat エンタープライズ Linux および Linux、macOS、または Unix)
- チュートリアル: 「Hello, World!」 CodeDeploy を使用したアプリケーション (Windows Server)
- チュートリアル: CodeDeploy (Windows サーバー、Ubuntu サーバー、または Red Hat エンタープライズ Linux) を使用してオンプレミスインスタンスにアプリケーションをデプロイします。
- チュートリアル: CodeDeploy を使用して、Auto Scaling グループにアプリケーションをデプロイします。
- チュートリアル: CodeDeploy を使用して、GitHub からアプリケーションをデプロイします。
- チュートリアル: Amazon ECS へアプリケーションをデプロイする
- チュートリアル: 検証テストを使用して Amazon ECS サービスをデプロイする
- チュートリアル: CodeDeploy と AWS サーバーレスアプリケーションモデルを使用して更新された Lambda 関数をデプロイする

チュートリアル: Amazon EC2 インスタンスに WordPress をデプロイする (Amazon Linux または Red Hat エンタープライズ Linux および Linux、macOS、または Unix)

このチュートリアルでは、PHP および MySQL に基づくオープンソースのブログツールおよびコンテンツ管理システムである WordPress を、Amazon Linux または Red Hat Enterprise Linux (RHEL) を実行している 1 つの Amazon EC2 インスタンスにデプロイします。

お探しのものではありませんか。

- 代わりに Windows Server を実行している Amazon EC2 インスタンスへのデプロイの演習を行う場合は、「チュートリアル: 「Hello, World!」 CodeDeploy を使用したアプリケーション (Windows Server)」を参照してください。
- Amazon EC2 インスタンスではなくオンプレミスインスタンスへのデプロイの演習を行う場合は、「チュートリアル: CodeDeploy (Windows サーバー、Ubuntu サーバー、または Red Hat エンタープライズ Linux) を使用してオンプレミスインスタンスにアプリケーションをデプロイします。」を参照してください。

このチュートリアルのステップは、Linux や macOS、Unix を実行しているローカルの開発用マシンの観点から説明されています。Windows を実行しているローカルマシンでこれらのステップのほとんどを完了できますが、chmod や wget などのコマンド、sed などのアプリケーション、および `/tmp` などのディレクトリパスを扱うステップは環境に合わせて変更する必要があります。

このチュートリアルを開始する前に、「CodeDeploy の開始方法」の前提条件を完了している必要があります。これには、ユーザーの設定、 のインストールまたはアップグレード AWS CLI、IAM インスタンスプロファイルとサービスロールの作成が含まれます。

トピック

- ステップ 1: Amazon Linux または Red Hat エンタープライズ Linux Amazon EC2 インスタンスを起動して設定する
- ステップ 2: Amazon Linux または Red Hat エンタープライズ Linux Amazon EC2 インスタンスにデプロイされるようにソースコンテンツを設定する
- ステップ 3: Amazon S3 に WordPress アプリケーションをアップロードする
- ステップ 4: WordPress アプリケーションをデプロイする
- ステップ 5: WordPress アプリケーションを更新して再デプロイする
- ステップ 6: WordPress のアプリケーションと関連リソースのクリーンアップ

ステップ 1: Amazon Linux または Red Hat エンタープライズ Linux Amazon EC2 インスタンスを起動して設定する

CodeDeploy を使用して WordPress アプリケーションをデプロイするには、Amazon Linux または Red Hat Enterprise Linux (RHEL) を実行する Amazon EC2 インスタンスが必要です。Amazon EC2 インスタンスでは、HTTP 接続を許可する新しいインバウンドセキュリティルールが必要です。このルールは、正しくデプロイした後で WordPress ページをブラウザで表示するために必要です。

「CodeDeploy のための Amazon EC2 インスタンスを作成します。」の手順に従います インスタンスの Amazon EC2 インスタンスタグの割り当てに関するこれらの指示の一部を実行する際には、必ず **Name** のタグキーと、**CodeDeployDemo** のタグ値を指定します。(別のタグキーまたはタグ値を指定した場合、ステップ 4: WordPress アプリケーションをデプロイする の手順で予期しない結果が生成される場合があります)。

Amazon EC2 インスタンスを起動する手順に従った後、このページに戻り、次のセクションに進みます。次のステップとして、CodeDeploy でアプリケーションを作成する には進まないでください。

Amazon Linux と RHEL Amazon EC2 インスタンスに接続します

Amazon EC2 インスタンスが起動した後、手順に従ってインスタンスに接続する演習をします。

1. ssh コマンド (または SSH 対応の PuTTY などのターミナルエミュレータ) を使用して Amazon Linux または RHEL Amazon EC2 インスタンスに接続します。Amazon EC2 インスタンスを開始した時に使用したインスタンスのパブリック DNS アドレスとキーペアのプライベートキーが必要になります。詳細については、「インスタンスへの接続」を参照してください。

   例えば、パブリック DNS アドレスが **ec2-01-234-567-890.compute-1.amazonaws.com** で、SSH アクセスの Amazon EC2 インスタンスキーペアが **codedeploydemo.pem** という名前の場合、以下のように入力します。

   ```
   ssh -i /path/to/codedeploydemo.pem ec2-
   user@ec2-01-234-567-890.compute-1.amazonaws.com
   ```

   */path/to/*codedeploydemo.pem を .pem ファイルのパスに置き換え、例の DNS アドレスを Amazon Linux または RHEL Amazon EC2 インスタンスのアドレスと置き換えます。

   > **Note**
   >
   > キーファイルのアクセス権限がオープンすぎるというエラーを受信した場合は、現在のユーザー (お客様) だけにアクセス権限を与えるように限定する必要があります。例えば、chmod Linux、macOS、または UNIX の場合は、次のように入力します。

   ```
   chmod 400 /path/to/codedeploydemo.pem
   ```

2. サインインしたら、Amazon EC2 インスタンスについては、AMI のバナーを参照してください。Amazon Linux の場合は、次のようになります。

```
       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|
```

3. 実行中の Amazon EC2 インスタンスからサインアウトできます。

> **Warning**
>
> Amazon EC2 インスタンスを停止または削除しないでください。そうしないと、CodeDeploy がインスタンスにデプロイできません。

HTTP トラフィックを許可するインバウンドルールの Amazon Linux または RHEL Amazon EC2 インスタンスへの追加

次のステップでは、デプロイされた WordPress アプリケーションのホームページがブラウザに表示されるように、Amazon EC2 インスタンスに開いている HTTP ポートがあることを確認します。

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/ec2/ で Amazon EC2 コンソールを開きます。
2. インスタンス を選択後、ご自分のインスタンスを選択します。
3. セキュリティグループ 内の 説明 タブで、インバウンドのルールの表示 を選択します。

   セキュリティグループのルールの一覧は、次のように表示されます。

```
Security Groups associated with i-1234567890abcdef0
 Ports     Protocol     Source     launch-wizard-N
 22        tcp          0.0.0.0/0          #
```

4. セキュリティグループ では、Amazon EC2 インスタンスのセキュリティグループを選択します。**launch-wizard-*N*** と名前が付けられる可能性があります。***N*** は、インスタンスの作成時にセキュリティグループに割り当てられた名前です。

   [Inbound] (インバウンド) タブを選択します。次の値を持つルールが表示された場合、ご利用のインスタンスのセキュリティグループは正しく設定されています。

- [Type]: HTTP
- [Protocol]: TCP
- [Port Range]: 80
- ソース: 0.0.0.0/0

5. これらの値を持つルールが表示されない場合は、セキュリティグループへのルールの追加 の手順を使用して、新しいセキュリティルールに追加します。

ステップ 2: Amazon Linux または Red Hat エンタープライズ Linux Amazon EC2 インスタンスにデプロイされるようにソースコンテンツを設定する

ここでは、アプリケーションのソースコンテンツを設定し、インスタンスにデプロイできるものを用意します。

トピック

- ソースコードの入手
- アプリケーションを実行するスクリプトの作成
- アプリケーション仕様ファイルの追加

ソースコードの入手

このチュートリアルでは、開発マシンからターゲット Amazon EC2 インスタンスに WordPress コンテンツ発行プラットフォームをデプロイします。WordPress のソースコードを入手するには、組み込みのコマンドラインの呼び出しを使用できます。または、開発マシンに Git をインストールしている場合は、代わりにそれを使用します。

これらのステップでは、WordPress ソースコードのコピーを開発マシンの `/tmp` ディレクトリにダウンロードすることを前提としています (任意のディレクトリを選択できますが、ステップで `/tmp` が指定されている場合は、その場所に必ず置き換えてください)。

次の 2 つのオプションのいずれかを選択して、開発マシンに WordPress ソースファイルをコピーします。最初のオプションでは、組み込みのコマンドラインの呼び出しを使用します。2 番目のオプションでは、Git を使用します。

トピック

- WordPress のソースコード (組み込みのコマンドライン呼び出し) のコピーを入手するには
- WordPress ソースコード (Git) のコピーを入手するには

WordPress のソースコード (組み込みのコマンドライン呼び出し) のコピーを入手するには

1. wget コマンドを呼び出して、WordPress のソースコードのコピーを .zip ファイル形式で現在のディレクトリにダウンロードします。

```
wget https://github.com/WordPress/WordPress/archive/master.zip
```

2. unzip、mkdir、cp、rm コマンドを呼び出して、以下を実行します。

   - master.zip ファイルを /tmp/WordPress_Temp ディレクトリ (フォルダ) に解凍します。
   - 解凍された内容を、/tmp/WordPress 宛先フォルダにコピーします。
   - 一時的な /tmp/WordPress_Temp フォルダと master ファイルを削除します。

   コマンドを一度に 1 つずつ実行します。

```
unzip master -d /tmp/WordPress_Temp
```

```
mkdir -p /tmp/WordPress
```

```
cp -paf /tmp/WordPress_Temp/WordPress-master/* /tmp/WordPress
```

```
rm -rf /tmp/WordPress_Temp
```

```
rm -f master
```

   これにより、/tmp/WordPress フォルダに WordPress ソースコードファイルのクリーンなセットが配置されます。

WordPress ソースコード (Git) のコピーを入手するには

1. 開発マシンで Git をダウンロードしてインストールします。
2. /tmp/WordPress フォルダで git init コマンドを呼び出します。

3. git clone コマンドを呼び出してパブリック WordPress レポジトリのクローンを作成し、その独自のコピーを /tmp/WordPress 宛先フォルダで作成します。

```
git clone https://github.com/WordPress/WordPress.git /tmp/WordPress
```

これにより、/tmp/WordPress フォルダに WordPress ソースコードファイルのクリーンなセットが配置されます。

アプリケーションを実行するスクリプトの作成

次に、ディレクトリでフォルダとスクリプトを作成します。CodeDeploy はこれらのスクリプトを使用して、ターゲット Amazon EC2 インスタンスでアプリケーションリビジョンをセットアップし、デプロイします。スクリプトの作成には任意のテキストエディタを使用できます。

1. WordPress ソースコードのコピーでスクリプトディレクトリを作成します。

```
mkdir -p /tmp/WordPress/scripts
```

2. install_dependencies.sh に /tmp/WordPress/scripts ファイルを作成します。ファイルに以下の行を追加します。この install_dependencies.sh スクリプトは Apache、MySQL、および PHP をインストールします。また、PHP に MySQL のサポートを追加します。

```
#!/bin/bash
sudo amazon-linux-extras install php7.4
sudo yum install -y httpd mariadb-server php
```

3. start_server.sh に /tmp/WordPress/scripts ファイルを作成します。ファイルに以下の行を追加します。この start_server.sh スクリプトは Apache および MySQL を起動します。

```
#!/bin/bash
systemctl start mariadb.service
systemctl start httpd.service
systemctl start php-fpm.service
```

4. stop_server.sh に /tmp/WordPress/scripts ファイルを作成します。ファイルに以下の行を追加します。この stop_server.sh スクリプトは Apache および MySQL を停止します。

```
#!/bin/bash
isExistApp="pgrep httpd"
if [[ -n $isExistApp ]]; then
systemctl stop httpd.service
fi
isExistApp=pgrep mysqld
if [[ -n $isExistApp ]]; then
systemctl stop mariadb.service
fi
isExistApp=pgrep php-fpm
if [[ -n $isExistApp ]]; then
systemctl stop php-fpm.service

fi
```

5. create_test_db.sh に /tmp/WordPress/scripts ファイルを作成します。ファイルに以下の行を追加します。この create_test_db.sh スクリプトは、MySQL を使用して、WordPress で使用する **test** データベースを作成します。

```

#!/bin/bash
mysql -uroot <<CREATE_TEST_DB
CREATE DATABASE IF NOT EXISTS test;
CREATE_TEST_DB
```

6. 最後に、/tmp/WordPress/scripts に change_permissions.sh スクリプトファイルを作成します。これは Apache のフォルダのアクセス権限を変更するために使用されます。

> **Important**
>
> このスクリプトにより、誰でも書き込めるように、/tmp/WordPress フォルダのアクセス権限が更新されました。これは、「ステップ 5: WordPress アプリケーションを更新して再デプロイする」で WordPress がそのデータベースに書き込みを行うために必要です。WordPress アプリケーションがセットアップされたら、次のコマンドを実行して、アクセス権限をよりセキュアな設定に更新します。
>
> ```
> chmod -R 755 /var/www/html/WordPress
> ```

```
#!/bin/bash
chmod -R 777 /var/www/html/WordPress
```

7. すべてのスクリプトに実行可能権限を付与します。コマンドラインで、次のように入力します。

```
chmod +x /tmp/WordPress/scripts/*
```

アプリケーション仕様ファイルの追加

次に、アプリケーション仕様ファイル (AppSpec ファイル) YAML CodeDeploy によって使用されるフォーマットされたファイルを追加します。

- アプリケーションリビジョンのソースファイルを、ターゲット Amazon EC2 インスタンスの宛先にマッピングする。
- デプロイされたファイルのカスタムアクセス権限を指定する。
- デプロイ中にターゲット Amazon EC2 インスタンスで実行するスクリプトを指定する。

AppSpec のファイル名は、`appspec.yml` とする必要があります。アプリケーションソースコードのルートディレクトリに配置する必要があります。このチュートリアルでは、ルートディレクトリは `/tmp/WordPress` です。

テキストエディタで `appspec.yml` という名前のファイルを作成します。このファイルに次の行を追加します。

```
version: 0.0
os: linux
files:
  - source: /
    destination: /var/www/html/WordPress
hooks:
  BeforeInstall:
    - location: scripts/install_dependencies.sh
      timeout: 300
      runas: root
```

```
  AfterInstall:
    - location: scripts/change_permissions.sh
      timeout: 300
      runas: root
  ApplicationStart:
    - location: scripts/start_server.sh
    - location: scripts/create_test_db.sh
      timeout: 300
      runas: root
  ApplicationStop:
    - location: scripts/stop_server.sh
      timeout: 300
      runas: root
```

CodeDeploy はこの AppSpec ファイルを使用して、開発マシンの `/tmp/WordPress` フォルダのすべてのファイルを、ターゲットの Amazon EC2 インスタンスの `/var/www/html/WordPress` フォルダにコピーします。デプロイ中に CodeDeploy は、デプロイライフサイクルで `root` や `/var/www/html/WordPress/scripts` などの中に指定されたイベントで、指定するスクリプトをターゲットの Amazon EC2 インスタンスの **`BeforeInstall`**、**`AfterInstall`** のフォルダとして実行します。これらのスクリプトのいずれかで実行に 300 秒 (5 分) 以上かかる場合、CodeDeploy はデプロイを停止し、デプロイを失敗としてマークします。

これらの設定の詳細については、「CodeDeploy AppSpec ファイルのリファレンス」を参照してください。

> **Important**
>
> このファイルの項目間のスペースの場所と数は重要です。間隔が正しくない場合、CodeDeploy はデバッグが困難な可能性のあるエラーを発生させます。詳細については、「AppSpec ファイルの間隔」を参照してください。

ステップ 3: Amazon S3 に WordPress アプリケーションをアップロードする

ここでソースコンテンツを CodeDeploy がデプロイできる場所に準備してアップロードします。次の手順では、Amazon S3 バケットをプロビジョニングしてバケット用のアプリケーションリビジョンのファイルを準備し、リビジョンのファイルをバンドルしてから、そのリビジョンをバケットにプッシュする方法を示します。

> **Note**
>
> このチュートリアルでは説明されていませんが、CodeDeploy を使用して GitHub リポジトリからインスタンスにアプリケーションをデプロイできます。詳細については、「GitHub と CodeDeploy との統合」を参照してください。

トピック

- Amazon S3 バケットをプロビジョニングします
- バケットのアプリケーションファイルを準備する
- アプリケーションのファイルを 1 つのアーカイブファイルにバンドルし、アーカイブファイルをプッシュする

Amazon S3 バケットをプロビジョニングします

ストレージコンテナあるいは Amazon S3 バケット を作成、または既存のバケットを使用します。バケットにリビジョンをアップロードできること、およびデプロイで使用する Amazon EC2 インスタンスがバケットからリビジョンをダウンロードできることを確認します。

AWS CLI、Amazon S3 コンソール、または Amazon S3 APIs を使用してAmazon S3バケットを作成できます。バケットを作成したら、バケットとその AWS アクセス許可を付与します。

> **Note**
>
> バケット名は、すべての AWS アカウントで Amazon S3 全体で一意である必要があります。**amzn-s3-demo-bucket** を使用できない場合、**amzn-s3-demo-bucket** の後にダッシュと自分の名前のイニシャル、または他の一意な識別子など別のバケット名を試してください。このチュートリアル全体で、バケット名を **amzn-s3-demo-bucket** に置き換えます。
>
> Amazon S3 バケットは、ターゲット Amazon EC2 インスタンスが起動されるのと同じ AWS リージョンに作成する必要があります。例えば、バケットを米国東部 (バージニア北部) リージョンで作成する場合、対象の Amazon EC2 インスタンスを米国東部 (バージニア北部) リージョンで起動する必要があります。

トピック

- Amazon S3 バケット (CLI) の作成

- Amazon S3 バケット (コンソール) の作成
- Amazon S3 バケットと AWS アカウントにアクセス許可を付与する

Amazon S3 バケット (CLI) の作成

mb コマンドを呼び出して、**amzn-s3-demo-bucket** という名前の Amazon S3 バケットを作成します。

```
aws s3 mb s3://amzn-s3-demo-bucket --region region
```

Amazon S3 バケット (コンソール) の作成

1. Amazon S3 コンソール (https://console.aws.amazon.com/s3/) を開きます。
2. Amazon S3 コンソールで [バケットの作成] を選択します。
3. [Bucket name] ボックスで、バケットの名前を入力します。
4. リージョンリストで、ターゲットリージョンを選択し、作成を選択します。

Amazon S3 バケットと AWS アカウントにアクセス許可を付与する

Amazon S3 バケットへのアップロードには、許可が必要です。Amazon S3 バケットポリシーで、これらのアクセス許可を指定できます。たとえば、次の Amazon S3 バケットポリシーでは、ワイルドカード文字 (*) を使用すると、 AWS アカウント111122223333は という名前の Amazon S3 バケット内の任意のディレクトリにファイルをアップロードできますamzn-s3-demo-bucket。

```
{
    "Statement": [
        {
            "Action": [
                "s3:PutObject"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
            "Principal": {
                "AWS": [
                    "111122223333"
                ]
            }
        }
```

```
    ]
}
```

AWS アカウント ID を表示するには、AWS 「アカウント ID の検索」を参照してください。

今は、Amazon S3 バケットが参加している各 Amazon EC2 インスタンスからのダウンロードリクエストを許可していることを確認するのに適した時期です。Amazon S3 バケットポリシーで、これを指定できます。例えば、次の Amazon S3 バケットポリシーでは、ワイルドカード文字 (*) を使用すると、ARN `arn:aws:iam::444455556666:role/CodeDeployDemo` を含む IAM インスタンスプロファイルがアタッチされた Amazon EC2 インスタンスが、`amzn-s3-demo-bucket` という名前の Amazon S3 バケットの任意のディレクトリからファイルをダウンロードすることを許可します。

```
{
    "Statement": [
        {
            "Action": [
                "s3:Get*",
                "s3:List*"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::444455556666:role/CodeDeployDemo"
                ]
            }
        }
    ]
}
```

Amazon S3 バケットポリシーを生成しアタッチする方法の詳細については、「バケットポリシーの例」を参照してください。

IAM ポリシーを作成しアタッチする方法については、「ポリシーの使用」を参照してください。

バケットのアプリケーションファイルを準備する

WordPress アプリケーションファイル、 およびスクリプトが、次のような開発用マシンで構成されることを確認します。

```
/tmp/
```

```
  |--WordPress/
      |-- appspec.yml
      |-- scripts/
      |    |-- change_permissions.sh
      |    |-- create_test_db.sh
      |    |-- install_dependencies.sh
      |    |-- start_server.sh
      |    |-- stop_server.sh
      |-- wp-admin/
      |    |-- (various files...)
      |-- wp-content/
      |    |-- (various files...)
      |-- wp-includes/
      |    |-- (various files...)
      |-- index.php
      |-- license.txt
      |-- readme.html
      |-- (various files ending with .php...)
```

アプリケーションのファイルを 1 つのアーカイブファイルにバンドルし、アーカイブファイルをプッシュする

WordPress アプリケーションファイルと をアーカイブファイル (アプリケーション リビジョン と呼ばれる) にバンドルします。

> **Note**
>
> バケットにオブジェクトを保存したり、バケットの内外にアプリケーションのリビジョンを転送したりする場合に課金されることがあります。詳細については、「Amazon S3 の料金」を参照してください。

1. 開発マシンで、ファイルが保存されたフォルダに切り替えます。

```
cd /tmp/WordPress
```

> **Note**
>
> このフォルダに切替わらなければ、ファイルのバンドルは現在のフォルダで起動されます。例えば、現在のフォルダが /tmp ではなく /tmp/WordPress である場合、バンド

ルは、tmp サブフォルダ以上を含む可能性のある WordPress フォルダ内のファイルとサブフォルダから開始します。

2. create-application コマンドを呼び出して、**WordPress_App** という名前の新しいアプリケーションを登録します。

```
aws deploy create-application --application-name WordPress_App
```

3. CodeDeploy で push コマンドを呼び出してファイルをまとめてバンドルし、Amazon S3 にリビジョンをアップロードし、アップロードされたリビジョンに関する情報を 1 つの操作で CodeDeploy に登録します。

```
aws deploy push \
  --application-name WordPress_App \
  --s3-location s3://amzn-s3-demo-bucket/WordPressApp.zip \
  --ignore-hidden-files
```

このコマンドは、現在のディレクトリ (隠しファイルを除く) から、**WordPressApp.zip** という名前の 1 つのアーカイブファイルにファイルをバンドルし、リビジョンを **amzn-s3-demo-bucket** バケットにアップロードし、CodeDeploy でアップロードしたリビジョンについての情報を登録します。

ステップ 4: WordPress アプリケーションをデプロイする

ここで、Amazon S3 にアップロードしたサンプルの WordPress アプリケーションリビジョンをデプロイします。 AWS CLI または CodeDeploy コンソールを使用してリビジョンをデプロイし、デプロイの進行状況をモニタリングできます。アプリケーションリビジョンが正常にデプロイされた後に、その結果を確認します。

トピック

- CodeDeploy を使用して、アプリケーションリビジョンをデプロイします
- デプロイをモニタリングおよびトラブルシューティングします。
- デプロイの確認

CodeDeploy を使用して、アプリケーションリビジョンをデプロイします

AWS CLI または コンソールを使用して、アプリケーションリビジョンをデプロイします。

トピック

- アプリケーションリビジョン (CLI) をデプロイするには
- アプリケーションリビジョン (コンソール) のデプロイ

アプリケーションリビジョン (CLI) をデプロイするには

1. デプロイにはデプロイグループが必要です。ただし、デプロイグループを作成する前に、サービスロール ARN が必要です。サービスロールは、ユーザーに代わってサービスアクセス権限を付与する IAM ロールです。この場合、サービスロールは、Amazon EC2 インスタンスにアクセスして Amazon EC2 インスタンスタグを拡張 (読み込み) するためのアクセス権限を CodeDeploy に付与します。

   すでに サービスロールの作成 (CLI) の手順に従ってサービスロールを作成している必要があります。サービスロールの ARN を取得するには、「サービスロール ARN の取得 (CLI)」を参照してください。

2. サービスロールの ARN を取得したら、create-deployment-group コマンドを呼び出し、**WordPress_DepGroup** という名前の Amazon EC2 タグと **WordPress_App** という名前のデプロイ設定を使用して、**CodeDeployDemo** という名前のアプリケーションと関連付けられる **CodeDeployDefault.OneAtATime** という名前のデプロイグループを作成します。

```
aws deploy create-deployment-group \
  --application-name WordPress_App \
  --deployment-group-name WordPress_DepGroup \
  --deployment-config-name CodeDeployDefault.OneAtATime \
  --ec2-tag-filters Key=Name,Value=CodeDeployDemo,Type=KEY_AND_VALUE \
  --service-role-arn serviceRoleARN
```

> **Note**
>
> デプロイメントグループの作成 コマンドは、デプロイおよびインスタンス内の指定されたイベントについて、トピックサブスクライバーに Amazon SNS 通知を送信するトリガーの作成に対応しています。このコマンドは、Amazon CloudWatch アラームのモニタリングしきい値が満たされたときにデプロイを自動的にロールバックし、デプロイを停止するアラームを設定するオプションもサポートします。このチュートリアルでは、これらのアクションに関するコマンドは含まれていません。

3. デプロイを作成する前に、デプロイグループのインスタンスに CodeDeploy エージェントがインストールされている必要があります。 AWS Systems Manager で次のコマンドを使用して、コマンドラインからエージェントをインストールできます。

```
aws ssm create-association \
  --name AWS-ConfigureAWSPackage \
  --targets Key=tag:Name,Values=CodeDeployDemo \
  --parameters action=Install,name=AWSCodeDeployAgent \
  --schedule-expression "cron(0 2 ? * SUN *)"
```

このコマンドは、CodeDeploy エージェントをインストールし、毎週日曜日の午前 2:00 に更新を試行する関連付けを Systems Manager ステートマネージャーに作成します。CodeDeploy エージェントの詳細については、「CodeDeploy エージェントの使用」を参照してください。Systems Manager のさらなる詳細については、「AWS Systems Managerとは」 を参照してください。

4. 次に、create-deployment コマンドを呼び出して、**amzn-s3-demo-bucket** という名前のバケットで **WordPressApp.zip** という名前のアプリケーションバージョンを使用して、**WordPress_App** という名前のアプリケーション、**CodeDeployDefault.OneAtATime** という名前のデプロイ設定と **WordPress_DepGroup** という名前のデプロイグループに関連付けられるデプロイを作成します。

```
aws deploy create-deployment \
  --application-name WordPress_App \
  --deployment-config-name CodeDeployDefault.OneAtATime \
  --deployment-group-name WordPress_DepGroup \
  --s3-location bucket=amzn-s3-demo-bucket,bundleType=zip,key=WordPressApp.zip
```

アプリケーションリビジョン (コンソール) のデプロイ

1. CodeDeploy コンソールを使用してアプリケーションリビジョンをデプロイする前に、サービスロール ARN が必要になります。サービスロールは、ユーザーに代わってサービスアクセス権限を付与する IAM ロールです。この場合、サービスロールは、Amazon EC2 インスタンスにアクセスして Amazon EC2 インスタンスタグを拡張 (読み込み) するためのアクセス権限を CodeDeploy に付与します。

すでに サービスロールの作成 (コンソール) の手順に従ってサービスロールを作成している必要があります。サービスロールの ARN を取得するには、「サービスロール ARN の取得 (コンソール)」を参照してください。

2. ARN があるので、CodeDeploy コンソールを使用して、アプリケーションリビジョンをデプロイします。

   にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

3. ナビゲーションペインで [デプロイ] を展開し、[アプリケーション] を選択します。
4. アプリケーションのリストで、WordPress_App を選択します。
5. [デプロイグループ] タブで、[デプロイグループの作成] を選択します。
6. [Deployment group name] (デプロイグループ名) に「**WordPress_DepGroup**」と入力します。
7. [Deployment type] で、[In-place deployment] を選択します。
8. [環境設定] で、[Amazon EC2 インスタンス] を選択します。
9. を使用した エージェント設定 AWS Systems Managerでは、デフォルトのままにします。
10. [Key] (キー) に、「**Name**」と入力します。
11. [値] には「**CodeDeployDemo**」と入力します。

    > **Note**
    >
    > **CodeDeployDemo** と入力すると、[1] が [Matching instances] の下に表示され、CodeDeploy が一致する Amazon EC2 インスタンスを 1 つ見つけたことを確認します。

12. [デプロイ設定] で [CodeDeployDefault.OneAtATime] を選択します。
13. [サービスロール ARN] でサービスロール ARN を選択し、[デプロイグループの作成] を選択します。
14. [デプロイの作成] を選択します。
15. [デプロイグループ] で [**WordPress_DepGroup**] を選択します。
16. [Repository type] の横の [My application is stored in Amazon S3] を選択します。[リビジョンの場所] で、前に Amazon S3 にアップロードしたサンプルアプリケーション WordPress のリビジョンの場所を入力します。場所の取得

    a. Amazon S3 コンソール (https://console.aws.amazon.com/s3/) を開きます。

b. バケットのリストで、[amzn-s3-demo-bucket] (または、アプリケーションリビジョンをアップロードしたバケットの名前) を選択します。

c. オブジェクトのリストで、WordPressApp.zip を選択します。

d. [概要] タブで、[リンク] フィールドの値をクリップボードにコピーします。

次のように表示されます。

https://s3.amazonaws.com/amzn-s3-demo-bucket/WordPressApp.zip

e. CodeDeploy コンソールに戻り、[リビジョンの場所] に [リンク] フィールドの値を貼り付けます。

17. [ファイルタイプ] リストで、ファイルの種類を検出できないというメッセージが表示される場合は、[.zip] を選択します。
18. (オプション) [Deployment description] ボックスにコメントを入力します。
19. [Deployment group overrides (デプロイグループの上書き)] を展開し、[デプロイ設定] から [CodeDeployDefault.OneAtATime] を選択します。
20. デプロイの開始 を選択します。新しく作成されたデプロイに関する情報は [Deployments] ページに表示されます。

デプロイをモニタリングおよびトラブルシューティングします。

AWS CLI または コンソールを使用して、デプロイをモニタリングおよびトラブルシューティングします。

トピック

- デプロイ (CLI) をモニタリングおよびトラブルシューティングするには
- デプロイ (コンソール) をモニタリングおよびトラブルシューティングするには

デプロイ (CLI) をモニタリングおよびトラブルシューティングするには

1. **WordPress_App** という名前のアプリケーションと **WordPress_DepGroup** という名前のデプロイグループに対して list-deployments コマンドを呼び出して、デプロイの ID を取得します。

```
aws deploy list-deployments --application-name WordPress_App --deployment-group-
name WordPress_DepGroup --query 'deployments' --output text
```

2. デプロイ ID を使用して get-deployment コマンドを呼び出します。

```
aws deploy get-deployment --deployment-id deploymentID --query
 'deploymentInfo.status' --output text
```

3. コマンドはデプロイの全体ステータスを返します。成功すると、値は Succeeded になります。

   全体的なステータスが Failed の場合、list-deployment-instances や デプロイメントインスタンスの取得 などのコマンドを呼び出してトラブルシューティングを行います。トラブルシューティングの他のオプションについては、「ログファイルの分析によるインスタンスでのデプロイの失敗の調査」を参照してください。

デプロイ (コンソール) をモニタリングおよびトラブルシューティングするには

CodeDeploy コンソール の [Deployments] ページで、[Status] 列でデプロイのステータスをモニタリングできます。

特に [Status] 列の値が [Succeeded] 以外の値である場合にデプロイに関する詳細情報を取得するには。

1. [デプロイ] テーブルで、デプロイの名前を選択します。デプロイが失敗すると、失敗の原因を説明するメッセージが表示されます。
2. [インスタンスアクティビティ] に、デプロイに関する詳細情報が表示されます。デプロイ失敗後、デプロイが失敗した Amazon EC2 インスタンスおよびステップを特定できる場合があります。
3. より多くのトラブルシューティングを行う場合、「View Instance Details」で説明されているような手法を使用できます。また、Amazon EC2 インスタンスでデプロイログファイルを分析できます。詳細については、「ログファイルの分析によるインスタンスでのデプロイの失敗の調査」を参照してください。

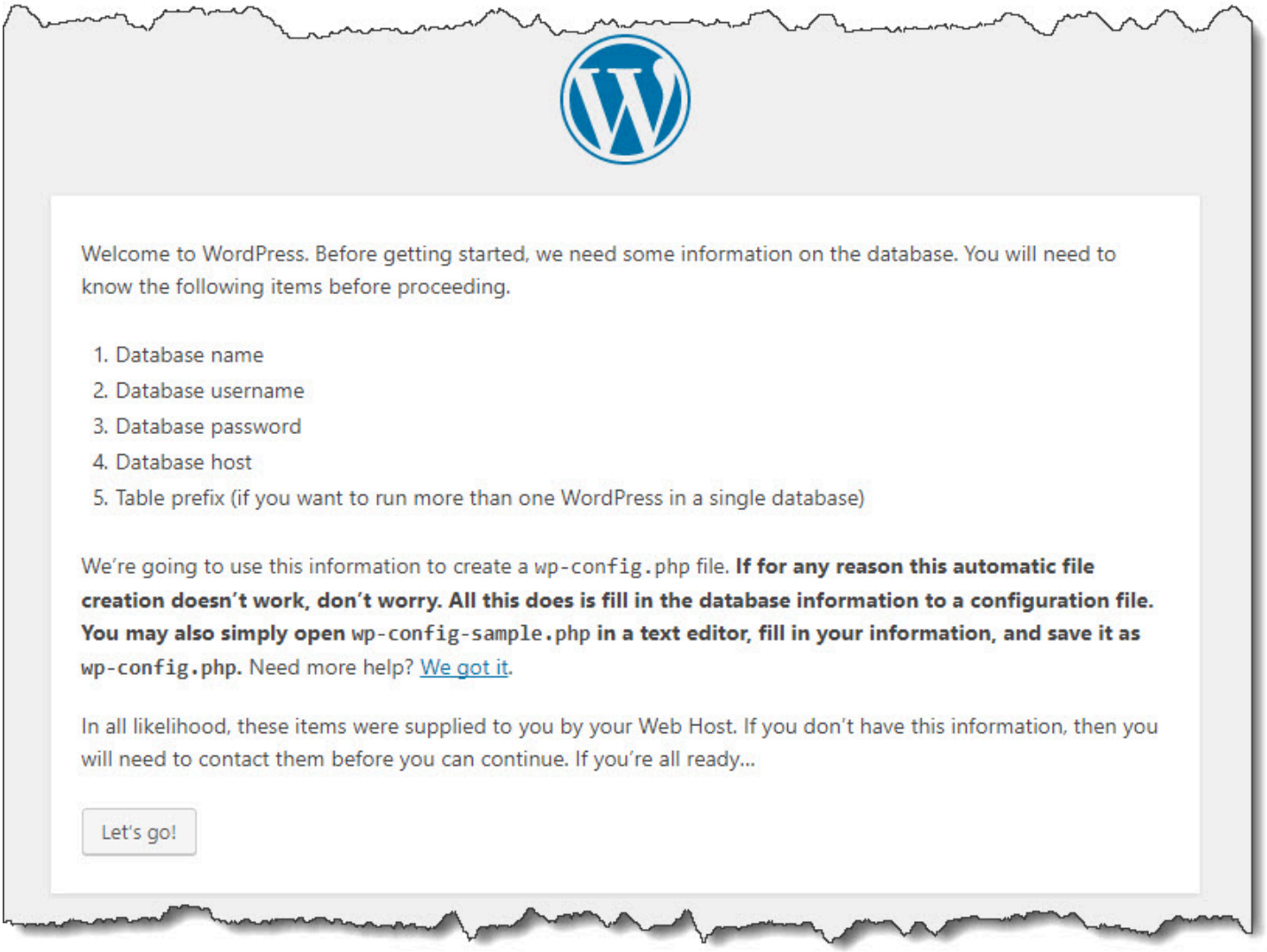
デプロイの確認

デプロイが成功したら、WordPress インストールが動作していることを確認します。Amazon EC2 インスタンスのパブリック DNS アドレスの後に /WordPress を使用して、ウェブブラウザのサイトを表示します。(Amazon EC2 コンソールでパブリック DNS 値を取得するには、Amazon EC2 インスタンスを選択して [説明] タブで [パブリック DNS] で値を探します。)

例えば、Amazon EC2 インスタンスのパブリック DNS アドレスが **ec2-01-234-567-890.compute-1.amazonaws.com** である場合、次の URL を使用します。

```
http://ec2-01-234-567-890.compute-1.amazonaws.com/WordPress
```

ブラウザでサイトを表示すると、次のような WordPress のウェルカムページが表示されます。



Amazon EC2 インスタンスで、HTTP インバウンドルールがそのセキュリティグループに追加されていない場合、WordPress ウェルカムページは表示されません。リモートサーバーが応答していないというメッセージが表示された場合は、Amazon EC2 インスタンスのセキュリティグループにインバウンドルールがあることを確認します。詳細については、「HTTP トラフィックを許可するインバウンドルールの Amazon Linux または RHEL Amazon EC2 インスタンスへの追加」を参照してください。

ステップ 5: WordPress アプリケーションを更新して再デプロイする

正常にアプリケーションリビジョンをデプロイしたので、開発用マシンで WordPress コードを更新し、 CodeDeploy を使用して、サイトを再デプロイします。後で、Amazon EC2 インスタンスのコード変更を確認する必要があります。

トピック

- WordPress サイトの設定
- サイトの変更
- サイトの再デプロイ

WordPress サイトの設定

コード変更の効果を表示するには、完全に機能するインストールができるように、WordPress サイトの設定を終了します。

1. ウェブブラウザにサイトの URL を入力します。URL は Amazon EC2 インスタンスと `/WordPress` 拡張子のパブリック DNS アドレスです。この WordPress サイト例 (Amazon EC2 インスタンスのパブリック DNS アドレスの例) では、URL は **`http://ec2-01-234-567-890.compute-1.amazonaws.com/WordPress`** です。
2. サイトをまだ設定していない場合は、WordPress のデフォルトのウェルカムページが表示されます。[始めましょう] を選択します。
3. デフォルトの MySQL データベースを使用するため、データベース設定ページで、以下の値を入力します。

   - データベース名: **`test`**
   - ユーザー名: **`root`**
   - パスワード: 空白のままにします。
   - データーベースホスト: **`localhost`**
   - テーブルプレフィックス: **`wp_`**

   [Submit] を選択して、データベースをセットアップします。
4. サイト設定を続行します。[Welcome] ページで任意の値を入力して [Install WordPress] を選択します。インストールが完了したら、ダッシュボードにサインインできます。

> **Important**
>
> WordPress アプリケーションのデプロイ中に、**change_permissions.sh** スクリプトにより /tmp/WordPress フォルダのアクセス権限が更新されたため、誰でもこのフォルダに書き込むことができます。ここで、次のコマンドを実行して、所有者のみが書き込めるようにアクセス権限を制限します。
>
> ```
> chmod -R 755 /var/www/html/WordPress
> ```

サイトの変更

WordPress サイトを変更するには、開発用マシンのアプリケーションのフォルダを参照してください。

```
cd /tmp/WordPress
```

サイトの色の一部を変更するには、wp-content/themes/twentyfifteen/style.css ファイルで、テキストエディターまたは sed を使用して、#fff を #768331 に変更します。

Linux または、GNU sed を持つほかのシステムでは、以下を使用します。

```
sed -i 's/#fff/#768331/g' wp-content/themes/twentyfifteen/style.css
```

macOS、Unix、または BSD sed を持つ他のシステムでは、以下を使用します。

```
sed -i '' 's/#fff/#768331/g' wp-content/themes/twentyfifteen/style.css
```

サイトの再デプロイ

サイトのコードを変更したので、Amazon S3 および CodeDeploy を使用してサイトを再デプロイします。

「アプリケーションのファイルを 1 つのアーカイブファイルにバンドルし、アーカイブファイルをプッシュする」の説明に従って、変更内容をバンドルして Amazon S3 にアップロードします。(これらの手順に従うときに、アプリケーションを作成する必要がないことに注意してください。) 新

しいリビジョンに以前と同じキーを指定します (**WordPressApp.zip**)。それを先に作成した同じ Amazon S3 バケットにアップロードします (例: **amzn-s3-demo-bucket**)。

サイトを再デプロイするには AWS CLI、、CodeDeploy コンソール、または CodeDeploy APIs を使用します。

トピック

- サイト (CLI) に再デプロイするには
- サイト (コンソール) の再デプロイ

サイト (CLI) に再デプロイするには

create-deployment コマンドを呼び出して、新しくアップロードされたリビジョンに基づいたデプロイを作成します。**amzn-s3-demo-bucket** という名前のバケットにある **WordPress_App** という名前のアプリケーション、**CodeDeployDefault.OneAtATime** という名前のデプロイ設定、**WordPress_DepGroup** という名前のデプロイグループ、**WordPressApp.zip** という名前のリビジョンを使用します。

```
aws deploy create-deployment \
 --application-name WordPress_App \
 --deployment-config-name CodeDeployDefault.OneAtATime \
 --deployment-group-name WordPress_DepGroup \
 --s3-location bucket=amzn-s3-demo-bucket,bundleType=zip,key=WordPressApp.zip
```

「デプロイをモニタリングおよびトラブルシューティングします。」に説明されているように、デプロイのステータスを確認できます。

CodeDeploy がサイトを再デプロイしたら、ウェブブラウザのサイトに再度アクセスして、色が変更されたことを確認します。(ブラウザを更新することが必要な場合があります。) 色が変更されていた場合は、サイトは正常に変更され、再デプロイされています。

サイト (コンソール) の再デプロイ

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで [デプロイ] を展開し、[アプリケーション] を選択します。
3. アプリケーションのリストで、WordPress_App を選択します。
4. [デプロイグループ] タブで、[**WordPress_DepGroup**] を選択します。
5. [デプロイの作成] を選択します。
6. [Create deployment] ページの

   a. [デプロイグループ] で [**WordPress_DepGroup**] を選択します。

   b. [リポジトリタイプ] エリアで、[My application is stored in (アプリケーションは Amazon S3 に格納されています)] を選択し、リビジョンの Amazon S3 リンクを [リビジョンの場所] ボックスにコピーします。リンク値の確認

      i. 別のブラウザタブで

         にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/s3/ で Amazon S3 コンソールを開きます。

         [amzn-s3-demo-bucket] を参照して開き、リビジョンとして **WordPressApp.zip** を選択します。

      ii. [Properties] ペインが Amazon S3 コンソールに表示されない場合、[Properties] ボタンを選択します。

      iii. [Properties] ペインで、[Link] フィールドの値をCodeDeploy コンソールの [Revision location] ボックスにコピーします。

   c. ファイルの種類を検出できないというメッセージが表示される場合は、[.zip] を選択します。

   d. [Deployment description] ボックスを空白のままにしておきます。

   e. [Deployment group overrides (デプロイグループの上書き)] を展開し、[デプロイ設定] から [CodeDeployDefault.OneAtATime] を選択します。

   f. デプロイの開始 を選択します。新しく作成されたデプロイに関する情報は [Deployments] ページに表示されます。

   g. 「デプロイをモニタリングおよびトラブルシューティングします。」に説明されているように、デプロイのステータスを確認できます。

      CodeDeploy がサイトを再デプロイしたら、ウェブブラウザのサイトに再度アクセスして、色が変更されたことを確認します。(ブラウザを更新することが必要な場合があります。) 色が変更されていた場合は、サイトは正常に変更され、再デプロイされています。

ステップ 6: WordPress のアプリケーションと関連リソースのクリーンアップ

これで、WordPress コードを正常に更新し、サイトを再デプロイしました。このチュートリアル用に作成したリソースの継続的な料金の発生を回避するため、以下を削除する必要があります。

- CloudFormation スタック (または、 の外部で作成した Amazon EC2 インスタンス) CloudFormation。
- Amazon S3 バケットの場合。
- CodeDeploy 内の `WordPress_App` アプリケーションの名前です。
- CodeDeploy エージェントの AWS Systems Manager ステートマネージャーの関連付け。

クリーンアップを実行するには AWS CLI、、 CloudFormation、Amazon S3、Amazon EC2、CodeDeploy コンソール、または AWS APIsを使用できます。

トピック

- リソース (CLI) をクリーンアップするには
- リソース (コンソール) をクリーンアップするには
- 次のステップ

リソース (CLI) をクリーンアップするには

1. このチュートリアルで CloudFormation テンプレートを使用した場合は、 という名前のスタックに対して delete-stack コマンドを呼び出します**`CodeDeployDemoStack`**。これにより、付随するすべての Amazon EC2 インスタンスが終了し、スタックによって作成された付随するすべての IAM ロールが削除されます。

   ```
   aws cloudformation delete-stack --stack-name CodeDeployDemoStack
   ```

2. Amazon S3 バケットを削除するには、rm スイッチを使用して --recursive という名前のバケットに対して **`amzn-s3-demo-bucket`** コマンドを呼び出します。これにより、バケットとバケット内のすべてのオブジェクトが削除されます。

   ```
   aws s3 rm s3://amzn-s3-demo-bucket --recursive --region region
   ```

3. WordPress_App アプリケーションを削除するには、delete-application コマンドを呼び出します。これにより、関連するすべてのデプロイグループレコードと、アプリケーションのデプロイレコードも削除されます。

```
aws deploy delete-application --application-name WordPress_App
```

4. Systems Manager ステートマネージャーの関連付けを削除する場合、delete-association コマンドを呼び出します。

```
aws ssm delete-association --assocation-id association-id
```

describe-association コマンドを呼び出して、*######## ID* を取得することができます。

```
aws ssm describe-association --name AWS-ConfigureAWSPackage --targets
 Key=tag:Name,Values=CodeDeployDemo
```

このチュートリアルで CloudFormation スタックを使用しなかった場合は、 terminate-instances コマンドを呼び出して、手動で作成した Amazon EC2 インスタンスをすべて終了します。終了させる Amazon EC2 インスタンスの ID を指定します。

```
aws ec2 terminate-instances --instance-ids instanceId
```

リソース (コンソール) をクリーンアップするには

このチュートリアルで CloudFormation テンプレートを使用した場合は、関連する CloudFormation スタックを削除します。

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/cloudformation で CloudFormation コンソールを開きます。
2. フィルターボックスに、前に作成した CloudFormation スタック名を入力します (例: **CodeDeployDemoStack**)。
3. スタック名の横のボックスをオンにします。[Actions] メニューで、[Delete Stack] を選択します。

   CloudFormation はスタックを削除し、付随するすべての Amazon EC2 インスタンスを終了し、付随するすべての IAM ロールを削除します。

CloudFormation スタックの外部で作成した Amazon EC2 インスタンスを終了するには:

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/ec2/ で Amazon EC2 コンソールを開きます。
2. [INSTANCES] リストで、[Instances] を選択します。
3. 検索ボックスで、終了する Amazon EC2 インスタンス名 (例: **CodeDeployDemo**) を入力して Enter キーを押します。
4. Amazon EC2 インスタンスを選択します。
5. [Actions] メニューで [Instance State] をポイントし、[Terminate] を選択します。プロンプトが表示されたら、[Yes, Terminate] を選択します。

インスタンスごとにこれらの手順を繰り返します。

Amazon S3 バケットの削除

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/s3/ で Amazon S3 コンソールを開きます。
2. バケットのリストで、前に作成した Amazon S3 バケットの名前を参照して選択します (例: **amzn-s3-demo-bucket**)。
3. バケットを削除する前に、まず、そのコンテンツを削除する必要があります。**WordPressApp.zip** のようなバケット内のすべてのファイルを選択します。[Actions] メニューで、[Delete] を選択します。削除を確認するプロンプトが表示されたら、[OK] を選択します。
4. バケットが空になると、バケットを削除できます。バケットのリストで、バケットの行 (バケット名ではなく) を選択します。[Delete bucket] を選択し、確認が求められたら [OK] を選択します。

CodeDeploy から `WordPress_App` アプリケーションの削除

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

> **Note**
>
> 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで [デプロイ] を展開し、[アプリケーション] を選択します。
3. アプリケーションのリストで、WordPress_App を選択します。
4. [Application details] ページで、[Delete application] を選択します。
5. プロンプトが表示されたら、アプリケーションの名前を入力して削除することを確定し、[削除] を選択します。

Systems Manager ステートマネージャーの関連付けの削除

1. https://console.aws.amazon.com/systems-manager で AWS Systems Manager コンソールを開きます。
2. ナビゲーションペインで、[ステートマネージャー] を選択してください。
3. 作成した関連付けを選択し、[削除] を選択します。

次のステップ

ここまでの作業で、CodeDeploy デプロイが正常に完了し、サイトのコードが更新され、再デプロイされました。

チュートリアル: 「Hello, World!」 CodeDeploy を使用したアプリケーション (Windows Server)

このチュートリアルでは、ウェブサーバーとしてインターネットインフォメーションサービス (IIS) を実行している単一の Windows Server の Amazon EC2 インスタンスに 1 つのウェブページをデプロイします。このウェブページには、「Hello, World!」というシンプルなメッセージが表示されます。メッセージ

お探しのものではありませんか。

- 代わりに Amazon Linux または Red Hat Enterprise Linux (RHEL) Amazon EC2 インスタンスへのデプロイを練習する場合、チュートリアル: Amazon EC2 インスタンスに WordPress をデプロイする (Amazon Linux または Red Hat エンタープライズ Linux および Linux、macOS、または Unix) を参照してください。
- 代わりにオンプレミスインスタンスへのデプロイの演習を行う場合は、「チュートリアル: CodeDeploy (Windows サーバー、Ubuntu サーバー、または Red Hat エンタープライズ Linux) を使用してオンプレミスインスタンスにアプリケーションをデプロイします。」を参照してください。

このチュートリアルのステップは、Windows の使用を前提としています。これらのステップのほとんどは Linux、macOS、Unix を実行しているローカルマシンでも完了できますが、`c:\temp` などの Windows ベースのディレクトリパスを扱うステップでは、対応するパスを使用する必要があります。また、Amazon EC2 インスタンスに接続する場合は、Remote Desktop Protocol (RDP) 経由で、Windows Server を実行する Amazon EC2 インスタンスに接続できるクライアントアプリケーションが必要です。(Windows にはデフォルトで、RDP 接続クライアントアプリケーションが搭載されています)。

このチュートリアルを開始する前に、ユーザーの設定CodeDeploy の開始方法、 のインストールまたはアップグレード、IAM インスタンスプロファイルとサービスロールの作成など AWS CLI、 の前提条件を満たす必要があります。

トピック

- ステップ 1: Windows Server の Amazon EC2 インスタンスを起動する
- ステップ 2: Windows Server の Amazon EC2 instance インスタンスにデプロイするソースコンテンツを設定する
- ステップ 3: 「Hello, World!」をアップロードする Amazon S3 へのアプリケーション
- ステップ 4: Hello World アプリケーションをデプロイする
- ステップ 5:「Hello World!」を更新およびデプロイする アプリケーション
- ステップ 6: 「Hello, World!」をクリーンアップする アプリケーションと関連リソース

ステップ 1: Windows Server の Amazon EC2 インスタンスを起動する

CodeDeploy で Hello World アプリケーションをデプロイする場合、Windows Server を実行している Amazon EC2 インスタンスが必要です。

「CodeDeploy のための Amazon EC2 インスタンスを作成します。」の手順に従います。Amazon EC2 インスタンスタグをインスタンスに割り当てる準備ができたら、必ず **Name** のタグキーと、**CodeDeployDemo** のタグ値を指定します。(別のタグキーまたはタグ値を指定した場合、「ステップ 4: Hello World アプリケーションをデプロイする」の手順で予期しない結果が生成される場合があります)。

Amazon EC2 インスタンスを起動後、このページに戻り、次のセクションに進みます。次のステップとして、CodeDeploy でアプリケーションを作成する には進まないでください。

Amazon EC2 インスタンスに接続します。

Amazon EC2 インスタンスが起動した後、手順に従ってインスタンスに接続する演習をします。

> **Note**
>
> これらの手順では、Windows および Windows Desktop Connection クライアントアプリケーションを実行していることを前提としています。詳細については、「RDP を使用して Windows インスタンスに接続する」を参照してください。他のオペレーティングシステムまたは他の RDP 接続クライアントアプリケーションに、これらの手順を適用する必要が生じる場合もあります。

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/ec2/ で Amazon EC2 コンソールを開きます。
2. ナビゲーションペインの [Instances] (インスタンス) で、[Instances] (インスタンス) を選択します。
3. 一覧で Windows Server インスタンスを参照して選択します。
4. 接続 を選択します。
5. [パスワードの取得]、[ファイルの選択] の順に選択します。
6. Windows Server の Amazon EC2 インスタンスと関連付けられた Amazon EC2 インスタンスキーペアファイルを参照して選択し、[オープン] を選択します。
7. [Decrypt Password] (パスワードを復号化) を選択します。表示されるパスワードをメモしておきます。これはステップ 10 で必要になります。
8. [Download Remote Desktop File] を選択し、ファイルを開きます。
9. リモート接続の発行元を特定できなくても接続を求められる場合は、続行します。
10. ステップ 7 でメモしておいたパスワードを入力し、次に進みます (RDP 接続クライアントアプリケーションでユーザー名が求められた場合は、**`Administrator`** と入力します)。
11. リモートコンピュータの ID を確認できなくても接続を求められた場合は、続行します。
12. 接続後、Windows Server を実行している Amazon EC2 インスタンスのデスクトップが表示されます。
13. これで、Amazon EC2 インスタンスから切断することができます。

> **Warning**
>
> インスタンスを停止または削除しないでください。それ以外の場合は、CodeDeploy をデプロイすることはできません。

Windows Server の Amazon EC2 インスタンスへの HTTP トラフィックを許可するインバウンドルールを追加する

次のステップでは、Windows Server の Amazon EC2 インスタンスにデプロイされたホームページがブラウザに表示されるように、Amazon EC2 インスタンスに開いている HTTP ポートがあることを確認します。

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/ec2/ で Amazon EC2 コンソールを開きます。
2. インスタンス を選択後、ご自分のインスタンスを選択します。
3. セキュリティグループ 内の 説明 タブで、インバウンドのルールの表示 を選択します。

   セキュリティグループのルールの一覧は、次のように表示されます。

   ```
   Security Groups associated with i-1234567890abcdef0
    Ports     Protocol     Source     launch-wizard-N
    22        tcp          0.0.0.0/0          #
   ```

4. セキュリティグループ では、Amazon EC2 インスタンスのセキュリティグループを選択します。**launch-wizard-*N*** と名前が付けられる可能性があります。***N*** は、インスタンスの作成時にセキュリティグループに割り当てられた名前です。

   [Inbound] (インバウンド) タブを選択します。次の値を持つルールが表示された場合、ご利用のインスタンスのセキュリティグループは正しく設定されています。

   - [Type]: HTTP
   - [Protocol]: TCP
   - [Port Range]: 80
   - ソース: 0.0.0.0/0
5. これらの値を持つルールが表示されない場合は、セキュリティグループへのルールの追加 の手順を使用して、新しいセキュリティルールに追加します。

ステップ 2: Windows Server の Amazon EC2 instance インスタンスにデプロイするソースコンテンツを設定する

ここでは、アプリケーションのソースコンテンツを設定して、Amazon EC2 インスタンスにデプロイできるものを準備します。このチュートリアルでは、 Windows Server を実行する Amazon

EC2 インスタンスに 1 つのウェブページをデプロイします。これはウェブサーバーとして Internet Information Services (IIS) を実行します。このウェブページには、「Hello, World!」というシンプルなメッセージが表示されます。メッセージ

トピック

- ウェブページの作成
- アプリケーションを実行するスクリプトの作成
- アプリケーション仕様ファイルの追加

ウェブページの作成

1. HelloWorldApp フォルダで c:\temp というサブディレクトリ (サブフォルダ) を作成し、そのフォルダに切り替えます。

```
mkdir c:\temp\HelloWorldApp
cd c:\temp\HelloWorldApp
```

> **Note**
>
> c:\temp という場所、または HelloWorldApp というサブフォルダ名を必ず使用する必要はありません。別の場所またはサブフォルダ名を使用する場合は、必ずこのチュートリアル全体で使用してください。

2. テキストエディタを使用して、フォルダ内にファイルを作成します。ファイルを index.html と名付けます。

```
notepad index.html
```

3. 次の HTML コードをファイルに追加し、ファイルを保存します。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/
TR/html4/loose.dtd">
<html>
<head>
  <title>Hello, World!</title>
  <style>
    body {
      color: #ffffff;
```

```
      background-color: #0188cc;
      font-family: Arial, sans-serif;
      font-size:14px;
    }
  </style>
</head>
<body>
  <div align="center"><h1>Hello, World!</h1></div>
  <div align="center"><h2>You have successfully deployed an application using
 CodeDeploy</h2></div>
  <div align="center">
    <p>What to do next? Take a look through the <a href="https://aws.amazon.com/
codedeploy">CodeDeploy Documentation</a>.</p>
  </div>
</body>
</html>
```

アプリケーションを実行するスクリプトの作成

次に、ターゲットの Amazon EC2 インスタンスでウェブサーバーをセットアップするために CodeDeploy が使用するスクリプトを作成します。

1. `index.html` ファイルが保存されているのと同じサブフォルダで、テキストエディタを使用して別のファイルを作成します。ファイルを `before-install.bat` と名付けます。

   ```
   notepad before-install.bat
   ```

2. 次のバッチスクリプトコードをファイルに追加し、ファイルを保存します。

   ```
   REM Install Internet Information Server (IIS).
   c:\Windows\Sysnative\WindowsPowerShell\v1.0\powershell.exe -Command Import-Module -
   Name ServerManager
   c:\Windows\Sysnative\WindowsPowerShell\v1.0\powershell.exe -Command Install-
   WindowsFeature Web-Server
   ```

アプリケーション仕様ファイルの追加

次に、ウェブページとバッチスクリプトファイルに加えて、 アプリケーション指定のファイル (AppSpec ファイル) を追加します。AppSpec ファイルは、CodeDeploy によって次の方式で使用される YAML ファイルです。

- アプリケーションリビジョンのソースファイルを、インスタンスの宛先にマッピングします。
- デプロイ中にインスタンスで実行するスクリプトを指定します。

AppSpec のファイル名は、appspec.yml とする必要があります。アプリケーションソースコードのルートフォルダに配置する必要があります。

1. index.html および before-install.bat ファイルが保存されているのと同じサブフォルダで、テキストエディタを使用して別のファイルを作成します。ファイルを appspec.yml と名付けます。

```
notepad appspec.yml
```

2. 次の YAML コードをファイルに追加し、ファイルを保存します。

```
version: 0.0
os: windows
files:
  - source: \index.html
    destination: c:\inetpub\wwwroot
hooks:
  BeforeInstall:
    - location: \before-install.bat
      timeout: 900
```

CodeDeploy は、この AppSpec ファイルを使用してアプリケーションソースコードのルートフォルダにある index.html ファイルを、ターゲット Amazon EC2 インスタンスの c:\inetpub\wwwroot フォルダにコピーします。デプロイ中に、CodeDeploy は before-install.bat デプロイライフサイクルイベントの間にターゲット Amazon EC2 インスタンスで **BeforeInstall** バッチスクリプトを実行します。このスクリプトの実行に 900 秒 (15 分) 以上かかる場合、CodeDeploy はデプロイを停止し、Amazon EC2 インスタンスへのデプロイを失敗とマークします。

これらの設定の詳細については、「CodeDeploy AppSpec ファイルのリファレンス」を参照してください。

> **Important**
>
> このファイルの項目間のスペースの場所と数は重要です。間隔が正しくない場合、CodeDeploy はデバッグが困難な可能性のあるエラーを発生させます。詳細については、「AppSpec ファイルの間隔」を参照してください。

ステップ 3: 「Hello, World!」をアップロードする Amazon S3 へのアプリケーション

ここでソースコンテンツを CodeDeploy がデプロイできる場所に準備してアップロードします。次の手順では、Amazon S3 バケットをプロビジョニングしてバケット用のアプリケーションリビジョンのファイルを準備し、リビジョンのファイルをバンドルしてから、そのリビジョンをバケットにプッシュする方法を示します。

> **Note**
>
> このチュートリアルでは説明されていませんが、CodeDeploy を使用して GitHub リポジトリからインスタンスにアプリケーションをデプロイできます。詳細については、「GitHub と CodeDeploy との統合」を参照してください。

トピック

- Amazon S3 バケットをプロビジョニングします
- バケットのアプリケーションファイルを準備する
- アプリケーションのファイルを 1 つのアーカイブファイルにバンドルし、アーカイブファイルをプッシュする

Amazon S3 バケットをプロビジョニングします

ストレージコンテナあるいは Amazon S3 バケット を作成、または既存のバケットを使用します。バケットにリビジョンをアップロードできること、およびデプロイで使用する Amazon EC2 インスタンスがバケットからリビジョンをダウンロードできることを確認します。

AWS CLI、Amazon S3 コンソール、または Amazon S3 APIs を使用してAmazon S3バケットを作成できます。バケットを作成したら、バケットとその CodeDeploy ユーザーにアクセス許可を付与します。

> **Note**
>
> バケット名は、すべての AWS アカウントで Amazon S3 全体で一意である必要があります。**amzn-s3-demo-bucket** を使用できない場合、**amzn-s3-demo-bucket** の後にダッシュと自分の名前のイニシャル、または他の一意な識別子など別のバケット名を試してください。このチュートリアル全体で、バケット名を **amzn-s3-demo-bucket** に置き換えます。
>
> Amazon S3 バケットは、ターゲット Amazon EC2 インスタンスが起動されるのと同じ AWS リージョンに作成する必要があります。例えば、バケットを米国東部 (バージニア北部) リージョンで作成する場合、対象の Amazon EC2 インスタンスを米国東部 (バージニア北部) リージョンで起動する必要があります。

トピック

- Amazon S3 バケット (CLI) の作成
- Amazon S3 バケット (コンソール) の作成
- Amazon S3 バケットと AWS アカウントにアクセス許可を付与する

Amazon S3 バケット (CLI) の作成

mb コマンドを呼び出して、**amzn-s3-demo-bucket** という名前の Amazon S3 バケットを作成します。

```
aws s3 mb s3://amzn-s3-demo-bucket --region region
```

Amazon S3 バケット (コンソール) の作成

1. Amazon S3 コンソール (https://console.aws.amazon.com/s3/) を開きます。
2. Amazon S3 コンソールで [バケットの作成] を選択します。
3. [Bucket name] ボックスで、バケットの名前を入力します。
4. リージョンリストで、ターゲットリージョンを選択し、作成を選択します。

Amazon S3 バケットと AWS アカウントにアクセス許可を付与する

Amazon S3 バケットへのアップロードには、許可が必要です。Amazon S3 バケットポリシーで、これらのアクセス許可を指定できます。たとえば、次の Amazon S3 バケットポリシーでは、ワイルド

カード文字 (*) を使用すると、 AWS アカウント111122223333は という名前の Amazon S3 バケット内の任意のディレクトリにファイルをアップロードできますamzn-s3-demo-bucket。

```
{
    "Statement": [
        {
            "Action": [
                "s3:PutObject"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
            "Principal": {
                "AWS": [
                    "111122223333"
                ]
            }
        }
    ]
}
```

AWS アカウント ID を表示するには、AWS 「アカウント ID の検索」を参照してください。

今は、Amazon S3 バケットが参加している各 Amazon EC2 インスタンスからのダウンロードリクエストを許可していることを確認するのに適した時期です。Amazon S3 バケットポリシーで、これを指定できます。例えば、次の Amazon S3 バケットポリシーでは、ワイルドカード文字 (*) を使用すると、ARN `arn:aws:iam::444455556666:role/CodeDeployDemo` を含む IAM インスタンスプロファイルがアタッチされた Amazon EC2 インスタンスが、`amzn-s3-demo-bucket` という名前の Amazon S3 バケットの任意のディレクトリからファイルをダウンロードすることを許可します。

```
{
    "Statement": [
        {
            "Action": [
                "s3:Get*",
                "s3:List*"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::444455556666:role/CodeDeployDemo"
                ]
```

```
            }
        }
    ]
}
```

Amazon S3 バケットポリシーを生成しアタッチする方法の詳細については、「バケットポリシーの例」を参照してください。

ステップ 1: セットアップ で作成した CodeDeploy 管理ユーザーに Amazon S3 バケットにリビジョンをアップロードするアクセス許可が必要です。これを指定する 1 つの方法は、IAM ポリシーを使用してユーザーのアクセス権限セットに追加するか、IAM ロール (ユーザーに引き受けを許可する) に追加することです。次の IAM ポリシーでは、ユーザーが `amzn-s3-demo-bucket` という名前の Amazon S3 バケット内の任意の場所でリビジョンをアップロードできるようにします。

JSON

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect":"Allow",
      "Action":["s3:PutObject"],
      "Resource":"arn:aws:s3:::amzn-s3-demo-bucket/*"
    }
  ]
}
```

IAM ポリシーの作成方法については、「IAM ユーザーガイド」の「IAM ポリシーの作成」を参照してください。アクセス権限セットにポリシーを追加する方法については、「AWS IAM アイデンティティセンター ユーザーガイド」の「アクセス権限セットを作成します。」を参照してください。

バケットのアプリケーションファイルを準備する

ウェブページ、AppSpec ファイル、およびスクリプトが開発マシン上で次のように整理されていることを確認します。

```
c:\
  |-- temp\
        |--HelloWorldApp\
```

```
            |-- appspec.yml
            |-- before-install.bat
            |-- index.html
```

アプリケーションのファイルを 1 つのアーカイブファイルにバンドルし、アーカイブファイルをプッシュする

ファイルをアーカイブファイル (アプリケーションリビジョンとも呼ばれる) にバンドルします。

> **Note**
>
> バケットにオブジェクトを保存したり、バケットの内外にアプリケーションのリビジョンを転送したりする場合に課金されることがあります。詳細については、「Amazon S3 の料金」を参照してください。

1. 開発マシンで、ファイルが保存されたフォルダに切り替えます。

   ```
   cd c:\temp\HelloWorldApp
   ```

   > **Note**
   >
   > このフォルダに切替わらなければ、ファイルのバンドルは現在のフォルダで起動されます。例えば、現在のフォルダが `c:\temp` ではなく `c:\temp\HelloWorldApp` である場合、バンドルは、`c:\temp` サブフォルダ以上を含む可能性のある `HelloWorldApp` フォルダ内のファイルとサブフォルダから開始します。

2. create-application コマンドを呼び出して、**`HelloWorld_App`** という名前の新しいアプリケーションを CodeDeploy に登録します。

   ```
   aws deploy create-application --application-name HelloWorld_App
   ```

3. CodeDeploy び push コマンドを呼び出してファイルをまとめてバンドルし、Amazon S3 にリビジョンをアップロードし、アップロードされたリビジョンに関する情報を 1 つの操作で CodeDeploy に登録します。

   ```
   aws deploy push --application-name HelloWorld_App --s3-location s3://amzn-s3-demo-bucket/HelloWorld_App.zip --ignore-hidden-files
   ```

このコマンドは、現在のディレクトリ (隠しファイルを除く) から、`HelloWorld_App.zip` という名前の 1 つのアーカイブファイルにファイルをバンドルし、リビジョンを **amzn-s3-demo-bucket** バケットにアップロードし、CodeDeploy でアップロードしたリビジョンについての情報を登録します。

ステップ 4: Hello World アプリケーションをデプロイする

ここで、Amazon S3 にアップロードした Hello World サンプルアプリケーションのリビジョンをデプロイします。 AWS CLI または CodeDeploy ンソールを使用して、リビジョンをデプロイし、デプロイのの進行状況をモニタリングします。アプリケーションリビジョンが正常にデプロイされた後に、その結果を確認します。

トピック

- CodeDeploy を使用して、アプリケーションリビジョンをデプロイします。
- デプロイをモニタリングおよびトラブルシューティングします。
- デプロイの確認

CodeDeploy を使用して、アプリケーションリビジョンをデプロイします。

アプリケーションをデプロイするには、CLI またはコンソールを使用できます。

トピック

- アプリケーションリビジョン (CLI) をデプロイするには
- アプリケーションリビジョン (コンソール) のデプロイ

アプリケーションリビジョン (CLI) をデプロイするには

1. まず、デプロイにはデプロイグループが必要です。ただし、デプロイグループを作成する前に、サービスロール ARN が必要です。サービスロールは、ユーザーに代わってサービスアクセス権限を付与する IAM ロールです。この場合、サービスロールは、Amazon EC2 インスタンスにアクセスして Amazon EC2 インスタンスタグを拡張 (読み込み) するためのアクセス権限を CodeDeploy に付与します。

   すでに サービスロールの作成 (CLI) の手順に従ってサービスロールを作成している必要があります。サービスロールの ARN を取得するには、「サービスロール ARN の取得 (CLI)」を参照してください。

2. ARN を取得したら、create-deployment-group コマンドを呼び出して、**HelloWorld_DepGroup** という名前のアプリケーションと関連付けられる **HelloWorld_App** という名前のデプロイグループを作成し、**CodeDeployDemo** という名前の Amazon EC2 インスタンスタグと、サービスロールARNと関連付けられる **CodeDeployDefault.OneAtATime** という名前のデプロイ設定を使用します。

```
aws deploy create-deployment-group --application-name HelloWorld_App
 --deployment-group-name HelloWorld_DepGroup --deployment-
config-name CodeDeployDefault.OneAtATime --ec2-tag-filters
 Key=Name,Value=CodeDeployDemo,Type=KEY_AND_VALUE --service-role-arn serviceRoleARN
```

> **Note**
>
> -デプロイメントグループの作成 コマンドは、デプロイおよびインスタンス内の指定されたイベントについて、トピックサブスクライバーに Amazon SNS 通知を送信するトリガーの作成に対応しています。このコマンドは、Amazon CloudWatch アラームのモニタリングしきい値が満たされたときにデプロイを自動的にロールバックし、デプロイを停止するアラームを設定するオプションもサポートします。このチュートリアルでは、これらのアクションに関するコマンドは含まれていません。

3. デプロイを作成する前に、デプロイグループのインスタンスに CodeDeploy エージェントがインストールされている必要があります。 AWS Systems Manager で次のコマンドを使用して、コマンドラインからエージェントをインストールできます。

```
aws ssm create-association --name AWS-ConfigureAWSPackage
 --targets Key=tag:Name,Values=CodeDeployDemo --parameters
 action=Install,name=AWSCodeDeployAgent --schedule-expression "cron(0 2 ? * SUN
 *)"
```

このコマンドは、CodeDeploy エージェントをインストールし、毎週日曜日の午前 2:00 に更新を試行する関連付けを Systems Manager ステートマネージャーに作成します。CodeDeploy エージェントの詳細については、「CodeDeploy エージェントの使用」を参照してください。Systems Manager のさらなる詳細については、「AWS Systems Managerとは」 を参照してください。

4. 次に、create-deployment コマンドを呼び出して、**amzn-s3-demo-bucket** という名前のバケットで **HelloWorld_App.zip** という名前のアプリケーションバージョンを使用して、**HelloWorld_App** という名前のアプリケーション、**CodeDeployDefault.OneAtATime**

という名前のデプロイ設定と **HelloWorld_DepGroup** という名前のデプロイグループに関連付けられるデプロイを作成します。

```
aws deploy create-deployment --application-name HelloWorld_App --deployment-config-name CodeDeployDefault.OneAtATime --deployment-group-name HelloWorld_DepGroup --s3-location bucket=amzn-s3-demo-bucket,bundleType=zip,key=HelloWorld_App.zip
```

アプリケーションリビジョン (コンソール) のデプロイ

1. CodeDeploy コンソールを使用してアプリケーションリビジョンをデプロイする前に、サービスロール ARN が必要になります。サービスロールは、ユーザーに代わってサービスアクセス権限を付与する IAM ロールです。この場合、サービスロールは、Amazon EC2 インスタンスにアクセスして Amazon EC2 インスタンスタグを拡張 (読み込み) するためのアクセス権限を CodeDeploy に付与します。

   すでに サービスロールの作成 (コンソール) の手順に従ってサービスロールを作成している必要があります。サービスロールの ARN を取得するには、「サービスロール ARN の取得 (コンソール)」を参照してください。
2. ARN があるため、CodeDeploy コンソールを使用して、アプリケーションリビジョンをデプロイできます。

   にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。
3. ナビゲーションペインで [デプロイ] を展開し、[アプリケーション] を選択します。
4. [HelloWorld_App] を選択します。
5. [デプロイグループ] タブで、[デプロイグループの作成] を選択します。
6. [Deployment group name] (デプロイグループ名) に「**HelloWorld_DepGroup**」と入力します。
7. [サービスロール] で、サービスロールの名前を選択します。
8. [デプロイタイプ] で、[インプレース] を選択します。
9. [環境設定] で、[Amazon EC2 インスタンス] を選択します。

10. を使用した エージェント設定 AWS Systems Managerでは、デフォルトのままにします。
11. [Key] (キー) に、「**Name**」と入力します。
12. [値] には「**CodeDeployDemo**」と入力します。
13. [デプロイ設定] で [CodeDeployDefault.OneAtATime] を選択します。
14. [ロードバランサー] で、[Enable load balancing (ロードバランシングの有効化)] をオフにします。
15. デプロイグループの作成 を選択します。
16. [デプロイの作成] を選択します。
17. [Deployment group] で、[HelloWorld_DepGroup] を選択します。
18. [リビジョンタイプ] では [アプリケーションは Amazon S3 に格納されています] を選択し、[リビジョンの場所] では以前に Amazon S3 にアップロードしたサンプルの Hello World アプリケーションリビジョンの場所を入力します。場所の取得

    a. Amazon S3 コンソール (https://console.aws.amazon.com/s3/) を開きます。
    b. バケットのリストで、[amzn-s3-demo-bucket] (または、アプリケーションリビジョンをアップロードしたバケットの名前) を選択します。
    c. オブジェクトのリストで、HelloWorld_App.zip を選択します。
    d. [概要] タブで、[パスのコピー] を選択します。
    e. CodeDeploy コンソールに戻り、[リビジョンの場所] に [リンク] フィールドの値を貼り付けます。
19. [リビジョンファイルの種類] で、[.zip] を選択します。
20. (オプション) [デプロイの説明] にコメントを入力します。
21. [デプロイの作成] を選択します。新しく作成されたデプロイに関する情報は [Deployments] ページに表示されます。

デプロイをモニタリングおよびトラブルシューティングします。

AWS CLI または コンソールを使用して、デプロイをモニタリングおよびトラブルシューティングします。

トピック

- デプロイ (CLI) をモニタリングおよびトラブルシューティングするには
- デプロイ (コンソール) をモニタリングおよびトラブルシューティングするには

デプロイ (CLI) をモニタリングおよびトラブルシューティングするには

1. **HelloWorld_App** という名前のアプリケーションと **HelloWorld_DepGroup** という名前のデプロイグループに対して list-deployments コマンドを呼び出して、デプロイの ID を取得します。

   ```
   aws deploy list-deployments --application-name HelloWorld_App --deployment-group-
   name HelloWorld_DepGroup --query "deployments" --output text
   ```

2. デプロイ ID を使用して get-deployment コマンドを呼び出します。

   ```
   aws deploy get-deployment --deployment-id deploymentID --query
    "deploymentInfo.status" --output text
   ```

3. コマンドはデプロイの全体ステータスを返します。成功すると、値は Succeeded になります。

   全体的なステータスが Failed の場合、list-deployment-instances や デプロイメントインスタンスの取得 などのコマンドを呼び出してトラブルシューティングを行います。トラブルシューティングの他のオプションについては、「ログファイルの分析によるインスタンスでのデプロイの失敗の調査」を参照してください。

デプロイ (コンソール) をモニタリングおよびトラブルシューティングするには

CodeDeploy コンソール の [Deployments] ページで、[Status] 列でデプロイのステータスをモニタリングできます。

特に [Status] 列の値が [Succeeded] 以外の値である場合にデプロイに関する詳細情報を取得するには。

1. [デプロイ] テーブルで、デプロイ ID を選択します。デプロイが失敗したら、失敗の原因を説明するメッセージがデプロイの詳細ページに表示されます。
2. インスタンスのデプロイに関する詳細情報が表示されます。デプロイ失敗後、デプロイが失敗した Amazon EC2 インスタンスおよびステップを特定できる場合があります。
3. より多くのトラブルシューティングを行う場合、View Instance Details のような手法を使用できます。また、Amazon EC2 インスタンスでデプロイログファイルを分析できます。詳細については、「ログファイルの分析によるインスタンスでのデプロイの失敗の調査」を参照してください。

デプロイの確認

デプロイが成功したら、インストールが動作していることを確認します。Amazon EC2 インスタンスのパブリック DNS アドレスを使用して、ウェブブラウザのウェブページを表示します。(Amazon EC2 コンソールでパブリック DNS 値を取得するには、 Amazon EC2 インスタンスを選択して [説明] タブで [パブリック DNS] で値を探します。)

例えば、Amazon EC2 インスタンスのパブリック DNS アドレスが **ec2-01-234-567-890.compute-1.amazonaws.com** である場合、次の URL を使用します。

```
http://ec2-01-234-567-890.compute-1.amazonaws.com
```

成功すると、Hello World ウェブページが表示されます。

ステップ 5:「Hello World!」を更新およびデプロイする アプリケーション

アプリケーションリビジョンを正常にデプロイしたので、開発マシンでウェブページのコードを更新し、CodeDeploy を使用してサイトを再デプロイします。デプロイ後、Amazon EC2 インスタンスで変更を確認できます。

トピック

- ウェブページの変更
- サイトの再デプロイ

ウェブページの変更

1. `c:\temp\HelloWorldApp` サブフォルダに移動して、テキストエディタを使用して `index.html` ファイルを変更します。

   ```
   cd c:\temp\HelloWorldApp
   notepad index.html
   ```

2. `index.html` ファイルのコンテンツを改訂するために、ウェブページの背景色と一部のテキストを変更し、ファイルを保存します。

   ```
   <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/
   TR/html4/loose.dtd">
   <html>
   ```

```
<head>
  <title>Hello Again, World!</title>
  <style>
    body {
      color: #ffffff;
      background-color: #66cc00;
      font-family: Arial, sans-serif;
      font-size:14px;
    }
  </style>
</head>
<body>
  <div align="center"><h1>Hello Again, World!</h1></div>
  <div align="center"><h2>You have successfully deployed a revision of an
 application using CodeDeploy</h2></div>
  <div align="center">
    <p>What to do next? Take a look through the <a href="https://aws.amazon.com/
codedeploy">CodeDeploy Documentation</a>.</p>
  </div>
</body>
</html>
```

サイトの再デプロイ

コードを変更したので、Amazon S3 と CodeDeploy を使用して、ウェブページを再デプロイします。

「アプリケーションのファイルを 1 つのアーカイブファイルにバンドルし、アーカイブファイルをプッシュする」の説明に従って、変更内容をバンドルして Amazon S3 にアップロードします。(これらの手順に従うときに、新しいアプリケーションを作成する必要はありません。) このリビジョンに以前と同じキーを指定します (**HelloWorld_App.zip**)。それを先に作成した同じ Amazon S3 バケットにアップロードします (例: **amzn-s3-demo-bucket**)。

AWS CLI または CodeDeploy コンソールを使用してサイトを再デプロイします。

トピック

- サイト (CLI) に再デプロイするには
- サイト (コンソール) の再デプロイ

サイト (CLI) に再デプロイするには

create-deployment コマンドを呼び出して、**amzn-s3-demo-bucket** という名前のバケットにある、**HelloWorld_App** という名前のアプリケーション、**CodeDeployDefault.OneAtATime** という名前のデプロイ設定、**HelloWorld_DepGroup** という名前のデプロイグループ、および **HelloWorld_App.zip** という名前のリビジョンをそれぞれ再度使用して、アップロードしたリビジョンに基づくデプロイを作成します。

```
 aws deploy create-deployment --application-name HelloWorld_App --deployment-config-
name CodeDeployDefault.OneAtATime --deployment-group-name HelloWorld_DepGroup --s3-
location bucket=amzn-s3-demo-bucket,bundleType=zip,key=HelloWorld_App.zip
```

「デプロイをモニタリングおよびトラブルシューティングします。」に説明されているように、新しいデプロイのステータスを確認できます。

CodeDeploy でサイトを再デプロイしたら、ウェブブラウザでサイトに再度アクセスし、ウェブページの背景色とテキストが変更されていることを確認します。(ブラウザを更新することが必要な場合があります。) 背景色とテキストが変更されていれば、これで完了です。サイトは変更され、再デプロイされています。

サイト (コンソール) の再デプロイ

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで、[アプリケーション] を選択します。
3. [アプリケーション] リストで、[HelloWorld_App] を選択します。
4. [デプロイ] タブで、[デプロイの作成] を選択します。

   a. [Deployment group] リストで、[HelloWorld_DepGroup] を選択します。
   b. [リビジョンの場所] に、リビジョンの Amazon S3 リンクを入力します。

      リンク値の確認

      i. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/s3/ で Amazon S3 コンソールを開きます。

amzn-s3-demo-bucket を参照して開いた後、Amazon S3 コンソールでリビジョン、**HelloWorld_App.zip** を選択します。

ii. [Properties] ペインが Amazon S3 コンソールに表示されない場合、[Properties] ボタンを選択します。

iii. [プロパティ] ペインで、[リンク] フィールドの値をコピーします

iv. CodeDeploy コンソールに戻り、[リビジョンの場所] にリンクを貼り付けます。

v.

c. [リビジョンファイルの種類] で、ファイルの種類を検出できないというメッセージが表示された場合は、[.zip] を選択します。

d. [デプロイメントの説明] は空白のままにしておきます。

e. [Deployment group overrides (デプロイグループのオーバーライド)] を展開し、[デプロイ設定] リストで、[CodeDeployDefault.OneAtATime]、[デプロイの作成] の順に選択します。

「デプロイをモニタリングおよびトラブルシューティングします。」に説明されているように、デプロイのステータスを確認できます。

CodeDeploy でサイトを再デプロイしたら、ウェブブラウザでサイトに再度アクセスし、ウェブページの背景色とテキストが変更されていることを確認します。(ブラウザを更新することが必要な場合があります。) 背景色とテキストが変更されていれば、これで完了です。サイトは変更され、再デプロイされています。

ステップ 6: 「Hello, World!」をクリーンアップする アプリケーションと関連リソース

これで「Hello, World!」コードを正常に更新しました。コードを記述し、サイトを再デプロイします。このチュートリアルを完了するために作成したリソースの継続的な料金の発生を回避するため、以下を削除する必要があります。

- CloudFormation スタック (または、 の外部で作成した Amazon EC2 インスタンス) CloudFormation。
- Amazon S3 バケットの場合。
- CodeDeploy 内の `HelloWorld_App` アプリケーションの名前です。
- CodeDeploy エージェントの AWS Systems Manager ステートマネージャーの関連付け。

クリーンアップを実行するには AWS CLI、、 CloudFormation、Amazon S3、Amazon EC2、CodeDeploy コンソール、または AWS APIsを使用できます。

トピック

- クリーンアップリソース (CLI) の使用
- リソース (コンソール) をクリーンアップするには
- 次のステップ

クリーンアップリソース (CLI) の使用

1. このチュートリアルで CloudFormation スタックを使用した場合は、 という名前のスタックに対して delete-stack コマンドを呼び出してスタックを削除します**CodeDeployDemoStack**。これにより、すべての付随する Amazon EC2 インスタンスが削除され、スタックによって最初に作成されたすべての付随する IAM ロールが削除されます。

   ```
   aws cloudformation delete-stack --stack-name CodeDeployDemoStack
   ```

2. Amazon S3 バケットを削除するには、rm スイッチを使用して --recursive という名前のバケットに対して **amzn-s3-demo-bucket** コマンドを呼び出します。これにより、バケットとバケット内のすべてのオブジェクトが削除されます。

   ```
   aws s3 rm s3://amzn-s3-demo-bucket --recursive --region region
   ```

3. CodeDeploy から HelloWorld_App アプリケーションを削除する場合、delete-application コマンドを呼び出します。これにより、すべての関連するデプロイグループレコードと、アプリケーションのデプロイレコードが削除されます。

   ```
   aws deploy delete-application --application-name HelloWorld_App
   ```

4. Systems Manager ステートマネージャーの関連付けを削除する場合、delete-association コマンドを呼び出します。

   ```
   aws ssm delete-association --assocation-id association-id
   ```

   describe-association コマンドを呼び出して、*######## ID* を取得することができます。

```
aws ssm describe-association --name AWS-ConfigureAWSPackage --targets
 Key=tag:Name,Values=CodeDeployDemo
```

5. このチュートリアルで CloudFormation スタックを使用しなかった場合は、 terminate-instances コマンドを呼び出して、手動で作成した Amazon EC2 インスタンスを終了します。終了する Amazon EC2 インスタンスの ID を指定します。

```
aws ec2 terminate-instances --instance-ids instanceId
```

リソース (コンソール) をクリーンアップするには

このチュートリアルで CloudFormation テンプレートを使用した場合は、関連する CloudFormation スタックを削除します。

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/cloudformation で CloudFormation コンソールを開きます。
2. 検索ボックスに、 CloudFormation スタック名 (など**CodeDeployDemoStack**) を入力します。
3. スタック名の横のチェックボックスをオンにします。
4. [Actions] メニューで、[Delete Stack] を選択します。これにより、スタックが削除され、すべての付随する Amazon EC2 インスタンスとすべての付随する IAM ロールも削除されます。

CloudFormation スタックの外部で作成した Amazon EC2 インスタンスを終了するには:

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/ec2/ で Amazon EC2 コンソールを開きます。
2. [Instances] エリアで、[Instances] を選択します。
3. 検索ボックスで、削除する Amazon EC2 インスタンスの名前を入力し、[Enter] キーを押します。
4. Amazon EC2 インスタンスを選択します。
5. [Actions] を選択して [Instance State] をポイントし、[Terminate] を選択します。プロンプトが表示されたら、[Yes, Terminate] を選択します。追加の Amazon EC2 インスタンスに対して、これらのステップを繰り返します。

Amazon S3 バケットの削除

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/s3/ で Amazon S3 コンソールを開きます。
2. バケットのリストで、Amazon S3 バケットの名前 (**amzn-s3-demo-bucket** など) を参照して選択します。
3. バケットを削除する前に、まず、そのコンテンツを削除する必要があります。**HelloWorld_App.zip** のようなバケット内のすべてのファイルを選択します。[Actions] メニューで、[Delete] を選択します。削除を確認するプロンプトが表示されたら、[OK] を選択します。
4. バケットが空になると、バケットを削除できます。バケットのリストで、バケットの行 (バケット名ではなく) を選択します。[Delete bucket] を選択し、確認が求められたら [OK] を選択します。

CodeDeploy から `HelloWorld_App` アプリケーションの削除

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで [デプロイ] を展開し、[アプリケーション] を選択します。
3. **HelloWorld_App** を選択します。
4. [アプリケーションを削除] を選択します。
5. 確認を求めるメッセージが表示されたら、**Delete**と入力し、[削除] を選択してください。

Systems Manager ステートマネージャーの関連付けの削除。

1. https://console.aws.amazon.com/systems-manager で AWS Systems Manager コンソールを開きます。
2. ナビゲーションペインで、[ステートマネージャー] を選択してください。
3. 作成した関連付けを選択し、[削除] を選択します。

次のステップ

ここまでの作業で、CodeDeploy を使って正常にデプロイを完了しました。お疲れ様でした。

チュートリアル: CodeDeploy (Windows サーバー、Ubuntu サーバー、または Red Hat エンタープライズ Linux) を使用してオンプレミスインスタンスにアプリケーションをデプロイします。

このチュートリアルでは、Windows サーバー、Ubuntu サーバー、または Red Hat エンタープライズ Linux (RHEL) を実行する単一のオンプレミスインスタンス (つまりは、Amazon EC2 インスタンスではない物理デバイス) へのサンプルアプリケーションリビジョンのデプロイをガイドすることで、CodeDeploy の経験を得ることができます。オンプレミスインスタンスについての情報、および CodeDeploy の使用方法については、Working with On-Premises Instances を参照してください。

お探しのものではありませんか。

- Amazon Linux または RHEL を実行する Amazon EC2 インスタンスへのデプロイの演習を行うには、チュートリアル: Amazon EC2 インスタンスに WordPress をデプロイする (Amazon Linux または Red Hat エンタープライズ Linux および Linux、macOS、または Unix) を参照してください。
- Windows サーバーを実行する Amazon EC2 インスタンスへのデプロイの演習を行うには、チュートリアル: 「Hello, World!」 CodeDeploy を使用したアプリケーション (Windows Server) を参照してください。

トピック

- 前提条件
- ステップ 1: オンプレミスインスタンスを設定する
- ステップ 2: サンプルのアプリケーションリビジョンを作成する
- ステップ 3: アプリケーションリビジョンをバンドルし、Amazon S3 にアップロードする
- ステップ 4: アプリケーションリビジョンをデプロイする
- ステップ 5: デプロイを確認する
- ステップ 6: リソースをクリーンアップする

前提条件

このチュートリアルを開始する前に、ユーザーの設定CodeDeploy の開始方法、 のインストールまたはアップグレード、サービスロールの作成など AWS CLI、 の前提条件を満たす必要があります。前提条件で説明したように、IAM インスタンスプロファイルを作成する必要はありません。オンプレミスインスタンスは、IAM インスタンスプロファイルを使用しません。

オンプレミスインスタンスとして設定する物理デバイスでは、「CodeDeploy エージェントで対応するオペレーティングシステム」に示したいずれかのオペレーティングシステムを実行している必要があります。

ステップ 1: オンプレミスインスタンスを設定する

オンプレミスインスタンスにデプロイする前に、設定を行う必要があります。「Working with On-Premises Instances」の指示に従ってから、このページに戻ります。

CodeDeploy エージェントをインストールします。

オンプレミスインスタンスを設定したら、CodeDeploy エージェントのインストール のオンプレミスインスタンスのステップに従い、このページに戻ります。

ステップ 2: サンプルのアプリケーションリビジョンを作成する

このステップでは、オンプレミスインスタンスにデプロイするサンプルのアプリケーションリビジョンを作成します。

オンプレミスインスタンス上に既にインストールされている、または組織のポリシーによってインストールが許可されているソフトウェアと機能を知るのは難しいため、オンプレミスインスタンスの場所にテキストファイルを書き込むために、ここで提供するサンプルアプリケーションリビジョンでは、バッチスクリプト (Windows Server の場合) またはシェルスクリプト (Ubuntu Server および RHEL の場合) を使用します。Install、 AfterInstall、 ApplicationStart および ValidateService を含む、複数の CodeDeploy デプロイライフサイクルイベントごとに 1 つのファイルが書き込まれます。BeforeInstall デプロイライフサイクルイベント中にスクリプトが実行され、このサンプルの前のデプロイ中に書き込まれた古いファイルを削除し、新しいファイルを書き込むオンプレミスインスタンス上に場所を作成します。

> **Note**
>
> 以下のいずれも該当しない場合、このサンプルのアプリケーションリビジョンはデプロイに失敗することがあります。

- オンプレミスインスタンスで CodeDeploy エージェントを起動するユーザーには、スクリプトを実行する権限がありません。
- ユーザーに、スクリプトにリストされている場所でフォルダを作成または削除する権限がない。
- ユーザーに、スクリプトにリストされている場所でテキストファイルを作成する権限がない。

> **Note**
>
> Windows Server インスタンスを設定し、別のサンプルをデプロイする場合は、ステップ 2: Windows Server の Amazon EC2 instance インスタンスにデプロイするソースコンテンツを設定する チュートリアルの「チュートリアル: 「Hello, World!」 CodeDeploy を使用したアプリケーション (Windows Server)」のサンプルを使用することをお勧めします。
> RHEL インスタンスを設定し、別のサンプルをデプロイする場合は、ステップ 2: Amazon Linux または Red Hat エンタープライズ Linux Amazon EC2 インスタンスにデプロイされるようにソースコンテンツを設定する のチュートリアルの チュートリアル: Amazon EC2 インスタンスに WordPress をデプロイする (Amazon Linux または Red Hat エンタープライズ Linux および Linux、macOS、または Unix) のサンプルを使用することをお勧めします。
> 現在、Ubuntu サーバー用の代替サンプルはありません。

1. 開発マシンで、サンプルのアプリケーションリビジョンのファイルを保存する、`CodeDeployDemo-OnPrem` という名前のサブディレクトリ (サブフォルダ) を作成し、そのサブフォルダに切り替えます。この例では、`c:\temp` のフォルダを Windows サーバーのルートフォルダとして使用するか、`/tmp` のフォルダを Ubuntu サーバーおよび RHEL のルートフォルダとして使用することを前提としています。別のフォルダを使用する場合は、このチュートリアル全体でそのフォルダに置き換えてください。

   Windows の場合:

   ```
   mkdir c:\temp\CodeDeployDemo-OnPrem
   cd c:\temp\CodeDeployDemo-OnPrem
   ```

   Linux、macOS、Unix の場合:

```
mkdir /tmp/CodeDeployDemo-OnPrem
cd /tmp/CodeDeployDemo-OnPrem
```

2. CodeDeployDemo-OnPrem サブフォルダのルートで、テキストエディタを使用して appspec.yml および install.txt という 2 つのファイルを作成します。

   Windows サーバーのための appspec.yml

```
version: 0.0
os: windows
files:
  - source: .\install.txt
    destination: c:\temp\CodeDeployExample
hooks:
  BeforeInstall:
    - location: .\scripts\before-install.bat
      timeout: 900
  AfterInstall:
    - location: .\scripts\after-install.bat
      timeout: 900
  ApplicationStart:
    - location: .\scripts\application-start.bat
      timeout: 900
  ValidateService:
    - location: .\scripts\validate-service.bat
      timeout: 900
```

   Ubuntu ServerとRHELのための appspec.yml：

```
version: 0.0
os: linux
files:
  - source: ./install.txt
    destination: /tmp/CodeDeployExample
hooks:
  BeforeInstall:
    - location: ./scripts/before-install.sh
      timeout: 900
  AfterInstall:
    - location: ./scripts/after-install.sh
      timeout: 900
```

```
  ApplicationStart:
    - location: ./scripts/application-start.sh
      timeout: 900
  ValidateService:
    - location: ./scripts/validate-service.sh
      timeout: 900
```

AppSpec ファイルの詳細については、「CodeDeploy 用のアプリケーション仕様ファイルをリビジョンに追加」および「CodeDeploy AppSpec ファイルのリファレンス」を参照してください。

`install.txt`:

```
The Install deployment lifecycle event successfully completed.
```

3. `CodeDeployDemo-OnPrem` サブフォルダのルートの下に、`scripts` サブフォルダを作成し、そのサブフォルダに切り替えます。

   Windows の場合:

   ```
   mkdir c:\temp\CodeDeployDemo-OnPrem\scripts
   cd c:\temp\CodeDeployDemo-OnPrem\scripts
   ```

   Linux、macOS、Unix の場合:

   ```
   mkdir -p /tmp/CodeDeployDemo-OnPrem/scripts
   cd /tmp/CodeDeployDemo-OnPrem/scripts
   ```

4. `scripts` のサブフォルダのルートで、テキストエディタを使用して、 Windows サーバーの場合は `before-install.bat`、`after-install.bat`、`application-start.bat`、`validate-service.bat`、あるいは Ubuntu サーバーおよび RHEL の場合は、`before-install.sh`、`after-install.sh`、`application-start.sh`、`validate-service.sh` という名前の 4 つのファイルを作成します。

   Windows サーバーの場合

   `before-install.bat`:

   ```
   set FOLDER=%HOMEDRIVE%\temp\CodeDeployExample
   ```

```
if exist %FOLDER% (
  rd /s /q "%FOLDER%"
)

mkdir %FOLDER%
```

`after-install.bat`:

```
cd %HOMEDRIVE%\temp\CodeDeployExample

echo The AfterInstall deployment lifecycle event successfully completed. > after-
install.txt
```

`application-start.bat`:

```
cd %HOMEDRIVE%\temp\CodeDeployExample

echo The ApplicationStart deployment lifecycle event successfully completed. >
 application-start.txt
```

`validate-service.bat`:

```
cd %HOMEDRIVE%\temp\CodeDeployExample

echo The ValidateService deployment lifecycle event successfully completed. >
 validate-service.txt
```

Ubuntu Server と RHEL の場合:

`before-install.sh`:

```
#!/bin/bash
export FOLDER=/tmp/CodeDeployExample

if [ -d $FOLDER ]
then
 rm -rf $FOLDER
fi

mkdir -p $FOLDER
```

`after-install.sh`:

```
#!/bin/bash
cd /tmp/CodeDeployExample

echo "The AfterInstall deployment lifecycle event successfully completed." > after-
install.txt
```

`application-start.sh`:

```
#!/bin/bash
cd /tmp/CodeDeployExample

echo "The ApplicationStart deployment lifecycle event successfully completed." >
 application-start.txt
```

`validate-service.sh`:

```
#!/bin/bash
cd /tmp/CodeDeployExample

echo "The ValidateService deployment lifecycle event successfully completed." >
 validate-service.txt

unset FOLDER
```

5. Ubuntu サーバーおよび RHEL の場合のみ、4 つのシェルスクリプトに実行権限があることを確認します。

```
chmod +x ./scripts/*
```

ステップ 3: アプリケーションリビジョンをバンドルし、Amazon S3 にアップロードする

アプリケーションリビジョンをデプロイする前に、ファイルをバンドルし、ファイルバンドルを Amazon S3 バケットにアップロードしておく必要があります。「CodeDeploy でアプリケーションを作成する」および「Amazon S3 に CodeDeploy のリビジョンをプッシュする (EC2/オンプレミスのデプロイのみ)」の手順に従います。(アプリケーションとデプロイグループには任意の名前を付け

ることができますが、アプリケーション名に「CodeDeploy-OnPrem-App」、デプロイグループ名に「CodeDeploy-OnPrem-DG」を使用することをお勧めします)。これらの手順を実行したら、このページに戻ります。

> **Note**
>
> または、GitHub リポジトリにファイルバンドルをアップロードし、そこからデプロイできます。詳細については、「GitHub と CodeDeploy との統合」を参照してください。

ステップ 4: アプリケーションリビジョンをデプロイする

アプリケーションリビジョンを Amazon S3 バケットにアップロードしたら、オンプレミスインスタンスへのデプロイを試します。「CodeDeploy でデプロイを作成する」の指示に従ってから、このページに戻ります。

ステップ 5: デプロイを確認する

デプロイが成功したことを確認するには、「CodeDeploy デプロイの詳細を表示する」の手順に従い、このページに戻ります。

デプロイが成功している場合は、c:\temp\CodeDeployExample のフォルダ (Windows の場合) または /tmp/CodeDeployExample (Ubuntu サーバーおよび RHEL の場合) に、 4 つのテキストフォルダが見つかります。

デプロイが失敗した場合は、「View Instance Details」および「インスタンスの問題のトラブルシューティング」のトラブルシューティングステップに従ってください。必要な修正を行い、アプリケーションリビジョンを再バンドルしてアップロードしてから、デプロイを再試行します。

ステップ 6: リソースをクリーンアップする

このチュートリアル用に作成したリソースの料金が継続的に発生しないようにするため、それ以上使用しない場合は Amazon S3 バケットを削除します。CodeDeploy とオンプレミスインスタンス中のアプリケーションおよびデプロイグループレコードのような関連リソースを、クリーンアップすることもできます。

AWS CLI または CodeDeploy コンソールと Amazon S3 コンソールの組み合わせと を使用して、リソース AWS CLI をクリーンアップできます。

リソースのクリーンアップ (CLI)

Amazon S3 バケットを削除するには

- バケット (例: `--recursive`) に対して、`amzn-s3-demo-bucket` のスイッチを用いて rm コマンドを呼び出します。バケットとバケット内のすべてのオブジェクトが削除されます。

```
aws s3 rm s3://your-bucket-name --recursive --region region
```

CodeDeploy 中のアプリケーションとデプロイグループレコードを削除するには

- アプリケーションに対して delete-application コマンドを呼び出します (例: `CodeDeploy-OnPrem-App`)。デプロイおよびデプロイグループのレコードが削除されます。

```
aws deploy delete-application --application-name your-application-name
```

オンプレミスインスタンスを登録解除し、IAM ユーザーを削除するには

- オンプレミスインスタンスとリージョンに対して deregister コマンドを呼び出します。

```
aws deploy deregister --instance-name your-instance-name --delete-iam-user --
region your-region
```

> **Note**
>
> このオンプレミスインスタンスに関連付けられた IAM ユーザーを削除しない場合は、代わりに `--no-delete-iam-user` のオプションを使用します。

CodeDeploy エージェントをアンインストールし、オンプレミスインスタンスから設定ファイルを削除するには

- オンプレミスインスタンスから uninstall コマンドを呼び出します。

```
aws deploy uninstall
```

これで、このチュートリアルで使用したリソースをクリーンアップするすべてのステップが完了しました。

リソースのクリーンアップ (コンソール)

Amazon S3 バケットを削除するには

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/s3/ で Amazon S3 コンソールを開きます。
2. 削除するバケットの横にあるアイコン (例: `amzn-s3-demo-bucket`) を選択します。ただし、バケット自体を選択しないでください。
3. [アクション]、[削除] の順に選択します。
4. バケットを削除するように求められたら、[OK] を選択します。

CodeDeploy 中のアプリケーションとデプロイグループレコードを削除するには

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで、[アプリケーション] を選択します。
3. 削除するアプリケーションの名前 (`CodeDeploy-OnPrem-App` など) を選択し、[アプリケーションの削除] を選択します。
4. プロンプトが表示されたら、アプリケーションの名前を入力して削除することを確定し、[削除] を選択します。

AWS CodeDeploy コンソールを使用してオンプレミスインスタンスの登録を解除したり、CodeDeploy エージェントをアンインストールしたりすることはできません。「オンプレミスインスタンスを登録解除し、IAM ユーザーを削除するには」の手順に従います。

チュートリアル: CodeDeploy を使用して、Auto Scaling グループにアプリケーションをデプロイします。

このチュートリアルでは、CodeDeploy を使用して Auto Scaling グループにアプリケーションリビジョンをデプロイします。Amazon EC2 Auto Scaling は、事前定義された条件を使用して Amazon EC2 インスタンスを起動した後に、不要になった Amazon EC2 インスタンスを終了します。Amazon EC2 Auto Scaling は、デプロイメントの負荷を処理するために常に適切な数の Amazon EC2 インスタンスを利用できるようにすることで、CodeDeploy のスケーリングに役立ちます。CodeDeploy による Amazon EC2 Auto Scaling 統合の詳細については、CodeDeploy と Amazon EC2 Auto Scaling の統合 を参照してください。

トピック

- 前提条件
- ステップ 1: Auto Scaling グループを作成して設定します。
- ステップ 2: Auto Scaling グループにアプリケーションをデプロイする
- ステップ 3: 結果の確認
- ステップ 4: Auto Scaling グループの Amazon EC2 インスタンスの数を増やす
- ステップ 5: 結果を再度確認します
- ステップ 6: クリーンアップする

前提条件

このチュートリアルを実行するには:

- のセットアップと設定CodeDeploy の開始方法、IAM インスタンスプロファイル (**CodeDeployDemo-EC2-Instance-Profile**) AWS CLI とサービスロール () の作成など、 のすべてのステップを完了します**CodeDeployDemo**。サービスロール は、ユーザーに代わってサービスアクセス権限を付与する、特別なタイプの IAM ロールです。
- 起動テンプレートを使用して Auto Scaling グループを作成する場合は、次の権限を追加する必要があります。
  - `ec2:RunInstances`
  - `ec2:CreateTags`
  - `iam:PassRole`

さらなる詳細については、ステップ 2: サービスロールを作成する、Creating a launch template for an Auto Scaling group、および Amazon EC2 Auto Scaling User Guide 中の Launch template support を参照してください。

- Ubuntu サーバーインスタンスおよび CodeDeploy と互換性のあるリビジョンを作成して使用します。リビジョンでは、次のいずれかを実行できます。
  - 「チュートリアル: CodeDeploy (Windows サーバー、Ubuntu サーバー、または Red Hat エンタープライズ Linux) を使用してオンプレミスインスタンスにアプリケーションをデプロイします。」チュートリアルの ステップ 2: サンプルのアプリケーションリビジョンを作成する のサンプルリビジョンを作成して使用します。
  - リビジョンを独自に作成するには、「CodeDeploy のアプリケーションリビジョンの操作」を参照してください。
- Inbound rule を用いて、**CodeDeployDemo-AS-SG** という名前のセキュリティグループを作成します。
  - Type: HTTP
  - ソース：どこでも

  これは、アプリケーションを表示し、デプロイメントの成功を確認するために必要です。セキュリティグループの作成方法については、Amazon EC2 user guide 中の Creating a security group を参照してください。

ステップ 1: Auto Scaling グループを作成して設定します。

このステップでは、単一の Amazon Linux、RHEL、または Windows サーバーの Amazon EC2 インスタンスを含む Auto Scaling グループを作成します。後のステップで、もう 1 つの Amazon EC2 インスタンスを追加するように Amazon EC2 Auto Scaling に指示し、CodeDeploy はそれにリビジョンをデプロイします。

トピック

- Auto Scaling グループ (CLI) を作成して設定するには
- Auto Scaling グループ (コンソール) を作成して設定するには

Auto Scaling グループ (CLI) を作成して設定するには

1. create-launch-template コマンドを呼び出して、Amazon EC2 起動テンプレートを作成します。

このコマンドを呼び出す前に、プレースホルダー *image-id* で表される、このチュートリアルで使用する AMI の ID が必要です。プレースホルダー *key-name* で表される、 Amazon EC2 インスタンスへのアクセスを有効にする Amazon EC2 インスタンスのキーペアの名前も必要です。

このチュートリアルで使用する AMI の ID を取得するには。

a. Amazon EC2 コンソールの https://console.aws.amazon.com/ec2/ を開いてください。

b. ナビゲーションペインで、[Instances] の下にある、[Instances] を選択して、[Launch Instance] を選択します。

c. Choose an Amazon Machine Image ページの Quick Start タブ上で、Amazon Linux 2 AMI、Red Hat Enterprise Linux 7.1、Ubuntu Server 14.04 LTS、あるいは Microsoft Windows Server 2012 R2 の横にある AMI の ID をメモします。

> **Note**
>
> CodeDeploy と互換性がある AMI のカスタムバージョンの場合、クイックスタート タブを参照する代わりに、ここでそれを選択します。CodeDeploy および Amazon EC2 Auto Scaling でカスタム AMI を使用することについての詳細については、CodeDeploy と Amazon EC2 Auto Scaling でのカスタム AMI の使用 を参照してください。

Amazon EC2 インスタンスのキーペアについては、Amazon EC2 インスタンスのキーペアの名前を使用します。

create-launch-template コマンドを呼び出します。

ローカル Linux、macOS、あるいは Unix マシンについて

```
aws ec2 create-launch-template \
  --launch-template-name CodeDeployDemo-AS-Launch-Template \
  --launch-template-data file://config.json
```

`config.json` ファイルのコンテンツ:

```
{
  "InstanceType":"t1.micro",
```

```
  "ImageId":"image-id",
  "IamInstanceProfile":{
    "Name":"CodeDeployDemo-EC2-Instance-Profile"
  },
  "KeyName":"key-name"
}
```

ローカル Windows マシンの場合

```
aws ec2 create-launch-template --launch-template-name CodeDeployDemo-AS-Launch-
Template --launch-template-data file://config.json
```

`config.json` ファイルのコンテンツ:

```
{
  "InstanceType":"t1.micro",
  "ImageId":"image-id",
  "IamInstanceProfile":{
    "Name":"CodeDeployDemo-EC2-Instance-Profile"
  },
  "KeyName":"key-name"
}
```

これらのコマンドは、`config.json` ファイルとともに、Auto Scaling グループ用の CodeDeployDemo-as-Launch-Template という名前の Amazon EC2 起動テンプレートを作成します。このテンプレートは、t1.micro Amazon EC2 インスタンスタイプに基づいて次のステップで作成されます。`ImageId`、`IamInstanceProfile`、`KeyName` への入力に基づいて、起動テンプレートでは AMI ID、起動時にインスタンスに渡す IAM ロールに関連付けられたインスタンスプロファイルの名前、およびインスタンスへの接続時に使用する Amazon EC2 キーペアも指定します。

2. create-auto-scaling-group のコマンドをを呼び出して、Auto Scaling グループを作成します。のリージョンとエンドポイントにリストされているリージョンの 1 つにあるいずれかのアベイラビリティーゾーンの名前が必要です。これはAWS 全般のリファレンスプレースホルダー *`Availability-zone`* で表されます。

> **Note**
>
> リージョン内のアベイラビリティーゾーンのリストを表示するには、以下を呼び出します。

```
aws ec2 describe-availability-zones --region region-name
```

例えば、米国西部 (オレゴン) リージョンのアベイラビリティーゾーンのリストを表示するには、次のように呼び出します。

```
aws ec2 describe-availability-zones --region us-west-2
```

リージョン名識別子のリストについては、「」を参照してくださいリージョン別リソースキットバケット名。

ローカル Linux、macOS、Unix マシンについて

```
aws autoscaling create-auto-scaling-group \
  --auto-scaling-group-name CodeDeployDemo-AS-Group \
  --launch-template CodeDeployDemo-AS-Launch-Template,Version='$Latest' \
  --min-size 1 \
  --max-size 1 \
  --desired-capacity 1 \
  --availability-zones availability-zone \
  --tags Key=Name,Value=CodeDeployDemo,PropagateAtLaunch=true
```

ローカル Windows マシンの場合

```
aws autoscaling create-auto-scaling-group --auto-scaling-group-name
 CodeDeployDemo-AS-Group --launch-template LaunchTemplateName=CodeDeployDemo-
AS-Launch-Template,Version="$Latest" --min-size 1 --max-size 1 --
desired-capacity 1 --availability-zones availability-zone --tags
 Key=Name,Value=CodeDeployDemo,PropagateAtLaunch=true
```

これらのコマンドは、**CodeDeployDemo-AS-Group** という名前の Amazon EC2 起動テンプレートに基づいた **CodeDeployDemo-AS-Launch-Template** という名前の Auto Scaling グループを作成します。この Auto Scaling グループには Amazon EC2 インスタンスが 1 つだけあり、指定したアベイラビリティーゾーンに作成されます。この Auto Scaling グループ内の各インスタンスには、タグ Name=CodeDeployDemo があります。このタグは、後で CodeDeploy エージェントをインストールするときに使用されます。

3. **CodeDeployDemo-AS-Group** に対して describe-auto-scaling-groups コマンドを呼び出します。

```
aws autoscaling describe-auto-scaling-groups --auto-scaling-group-names
 CodeDeployDemo-AS-Group --query "AutoScalingGroups[0].Instances[*].[HealthStatus,
 LifecycleState]" --output text
```

戻り値に、Healthy および InService と表示されるまで続行しないでください。

4. Auto Scaling グループ中のインスタンスには、CodeDeploy デプロイで使用されるためにインストールされている CodeDeploy エージェントがある必要があります。Auto Scaling グループの作成時に追加されたタグ AWS Systems Manager を使用して、 から create-association コマンドを呼び出して CodeDeploy エージェントをインストールします。

```
aws ssm create-association \
  --name AWS-ConfigureAWSPackage \
  --targets Key=tag:Name,Values=CodeDeployDemo \

 --parameters action=Install, name=AWSCodeDeployAgent \
  --schedule-expression "cron(0 2 ? * SUN *)"
```

このコマンドは、Systems Manager ステートマネージャーで関連付けを作成します。これは、Auto Scaling グループのすべてのインスタンスに CodeDeploy エージェントをインストールし、毎週日曜日の午前 2 時に更新を試みます。CodeDeploy エージェントの詳細については、「CodeDeploy エージェントの使用」を参照してください。Systems Manager のさらなる詳細については、「AWS Systems Managerとは」 を参照してください。

Auto Scaling グループ (コンソール) を作成して設定するには

1. Amazon EC2 コンソールの https://console.aws.amazon.com/ec2/ を開いてください。
2. グローバルナビゲーションバーで、 のリージョンとエンドポイントにリストされているリージョンのいずれかAWS 全般のリファレンスが選択されていることを確認します。Amazon EC2 Auto Scaling リソースは指定したリージョンに関連付けられ、CodeDeploy は一部のリージョンでのみサポートされています。
3. ナビゲーションペインで、[インスタンス] の [テンプレートの起動] を選択します。
4. [起動テンプレートの作成] を選択してください。

5. Launch template name and description ダイアログで、Launch template name に **CodeDeployDemo-AS-Launch-Template** を入力します。他のフィールドはデフォルトのためのデフォルトをそのままにします。
6. Amazon machine image (AMI) ダイアログで、AMI の下にあるドロップダウンをクリックし、このチュートリアルで使用する AMI を選択します。

   - AMI ドロップダウンの Quick Start 上で、次のどれかを選択します：Amazon Linux 2 AMI、Red Hat Enterprise Linux 7.1、Ubuntu Server 14.04 LTS、またはMicrosoft Windows Server 2012 R2

   > **Note**
   >
   > CodeDeploy と互換性がある AMI のカスタムバージョンの場合、クイックスタートタブを参照する代わりに、ここでそれを選択します。CodeDeploy および Amazon EC2 Auto Scaling でカスタム AMI を使用する方法についての詳細は、CodeDeploy と Amazon EC2 Auto Scaling でのカスタム AMI の使用 を参照してください。

7. Instance type で、ドロップダウンを選択し、t1.micro を選択します。検索バーを使用すると、よりすばやく検索できます。
8. Key pair (login) ダイアログボックスで、[Choose an existing key pair を選択します。[キーペアの選択] のドロップダウンリストで、前のステップで作成した Amazon EC2 インスタンスのキーペアを選択します。
9. Network settings ダイアログボックスで、[Virtual Public Cloud (VPC) を選択します。

   Security groupsドロップダウンで、tutorial's prerequisites section (**CodeDeployDemo-AS-SG**) で作成したセキュリティグループを選択します。
10. Advanced details ダイアログボックスを展開します。IAM instance profile ドロップダウンで、IAM instance profile の下で以前の (**CodeDeployDemo-EC2-Instance-Profile**) を作成した IAM ロールを選択します。

    デフォルトの残りはそのままにしておきます。
11. [起動テンプレートの作成] を選択してください。
12. Next steps ダイアログボックスで、[Create Auto Scaling group を選択します。
13. Choose launch template or configuration (起動テンプレートまたは設定の選択) ページで、Auto Scaling group name を **CodeDeployDemo-AS-Group** に入力します。

14. 起動テンプレートダイアログボックスで、起動テンプレート (**CodeDeployDemo-AS-Launch-Template**) を入力する必要があります。そうでない場合は、ドロップダウンメニューから、それを選択します。デフォルトのままにして、Next を選択します。
15. [ネットワーク] セクションの下にある [インスタンス起動オプションを選択] ページで、[VPC] はデフォルト VPC を選択します。次に、アベイラビリティーゾーンとサブネットにはデフォルトサブネットを選択します。デフォルトを選択できない場合は、VPC を作成する必要があります。Amazon S3 の詳細については、Amazon VPC の開始方法 を参照してください。
16. [Instance type requirements] (インスタンスタイプの要件) セクションでは、このステップを簡略化するためにデフォルト設定を使用します。(起動テンプレートを上書きしないでください。) このチュートリアルでは、起動テンプレートで指定されたインスタンスタイプを使用して、オンデマンドインスタンスのみを起動します。
17. Next を選択して Configure advanced options ページに行きます。
18. デフォルト値をそのまま保ち、Next を選択します。
19. Configure group size and scaling policies ページで、1 のデフォルトの Group size の値をそのままにします。[次へ] を選択します。
20. 通知の設定の手順をスキップし、Next を選択します。
21. Add tags ページ上で、後で CodeDeploy エージェントをインストールするときに使用するタグを追加します。[タグを追加] を選択します。

    a. [Key] (キー) に、「**Name**」と入力します。

    b. [値] には「**CodeDeployDemo**」と入力します。

    [次へ] を選択します。
22. Review ページで上の Auto Scaling グループの情報を確認し、Create Auto Scaling group を選択します。
23. ナビゲーションバーで、選択された Auto Scaling Groups を用いて **CodeDeployDemo-AS-Group** を選択し、Instance Management タブを選びます。[Lifecycle] 列に [InService] の値が表示されて、[Health Status] 列に、[Healthy] の値が表示されるまで進まないでください。
24. CodeDeploy エージェントをインストールするには、「CodeDeploy エージェントのインストール」のステップに従い、`Name=CodeDeployDemo` インスタンスタグを使用します。

ステップ 2: Auto Scaling グループにアプリケーションをデプロイする

このステップでは、Auto Scaling グループの単一 Amazon EC2 インスタンスにリビジョンをデプロイします。

トピック

- デプロイを作成するには (CLI)
- デプロイを作成するには (コンソール)

デプロイを作成するには (CLI)

1. create-application コマンドを呼び出して、**SimpleDemoApp** と言う名前のアプリケーションを作成します。

   ```
   aws deploy create-application --application-name SimpleDemoApp
   ```

2. ステップ 2: CodeDeployのサービスのロールを作成する の手順に従ってサービスロールを作成している必要があります。サービスロールは、Amazon EC2 インスタンスにアクセスしてタグを拡張 (読み込み) する許可を CodeDeploy に付与します。サービスロール ARN が必要になります。サービスロール ARN を取得するには、サービスロール ARN の取得 (CLI) の手順に従います。

3. これで、指定したサービスロール ARN で、create-deployment-group コマンドを呼び出して **SimpleDemoDG** という名前のデプロイグループを作成し、**SimpleDemoApp** という名前のアプリケーションと関連付け、**CodeDeployDemo-AS-Group** と言う名前の Auto Scaling グループと **CodeDeployDefault.OneAtATime** と言う名前のデプロイ設定を使用するサービスロール ARN が作成されました。

   > **Note**
   >
   > create-deployment-group コマンドは、デプロイおよびインスタンス内の指定されたイベントについて、トピックサブスクライバーに Amazon SNS 通知を送信するトリガーの作成をサポートします。このコマンドは、Amazon CloudWatch アラームのモニタリングしきい値が満たされたときにデプロイを自動的にロールバックし、デプロイを停止するアラームを設定するオプションもサポートします。このチュートリアルでは、これらのアクションのためのコマンドは含まれていません。

ローカル Linux、macOS、Unix マシンについて

```
aws deploy create-deployment-group \
  --application-name SimpleDemoApp \
  --auto-scaling-groups CodeDeployDemo-AS-Group \
  --deployment-group-name SimpleDemoDG \
  --deployment-config-name CodeDeployDefault.OneAtATime \
  --service-role-arn service-role-arn
```

ローカル Windows マシンの場合

```
aws deploy create-deployment-group --application-name SimpleDemoApp --auto-scaling-
groups CodeDeployDemo-AS-Group --deployment-group-name SimpleDemoDG --deployment-
config-name CodeDeployDefault.OneAtATime --service-role-arn service-role-arn
```

4. 指定された場所のリビジョンを使用して、**SimpleDemoApp** という名前のアプリケーションと関連付けられたデプロイ、**CodeDeployDefault.OneAtATime** という名前のデプロイ設定、**SimpleDemoDG** という名前のデプロイグループを作成する create-deployment コマンドを呼び出します。

   Amazon Linux および RHEL の Amazon EC2インスタンスの場合、ローカルの Linux、macOS、または Unixマシンから呼び出します

   ```
   aws deploy create-deployment \
     --application-name SimpleDemoApp \
     --deployment-config-name CodeDeployDefault.OneAtATime \
     --deployment-group-name SimpleDemoDG \
     --s3-location bucket=bucket-name,bundleType=zip,key=samples/latest/
   SampleApp_Linux.zip
   ```

   bucket-name は、リージョンの CodeDeploy リソースキットファイルが含まれている Amazon S3 バケットの名前です。例えば、米国東部 (オハイオ) リージョンの場合、***#####*** を aws-codedeploy-us-east-2 に置き換えます。バケット名のリストについては、リージョン別リソースキットバケット名 を参照してください。

   ローカル Windows マシンから呼び出した Amazon Linux および RHEL Amazon EC2 instances の場合

```
aws deploy create-deployment --application-name SimpleDemoApp --deployment-config-
name CodeDeployDefault.OneAtATime --deployment-group-name SimpleDemoDG --s3-
location bucket=bucket-name,bundleType=zip,key=samples/latest/SampleApp_Linux.zip
```

bucket-name は、リージョンの CodeDeploy リソースキットファイルが含まれている Amazon S3 バケットの名前です。例えば、米国東部 (オハイオ) リージョンの場合、##### を aws-codedeploy-us-east-2 に置き換えます。バケット名のリストについては、リージョン別リソースキットバケット名 を参照してください。

ローカルの Linux、macOS、または Unix マシンから呼び出した Windows サーバー Amazon Linux および RHEL の Amazon EC2インスタンスの場合

```
aws deploy create-deployment \
  --application-name SimpleDemoApp \
  --deployment-config-name CodeDeployDefault.OneAtATime \
  --deployment-group-name SimpleDemoDG \
  --s3-location bucket=bucket-name,bundleType=zip,key=samples/latest/
SampleApp_Windows.zip
```

bucket-name は、リージョンの CodeDeploy リソースキットファイルが含まれている Amazon S3 バケットの名前です。例えば、米国東部 (オハイオ) リージョンの場合、##### を aws-codedeploy-us-east-2 に置き換えます。バケット名のリストについては、リージョン別リソースキットバケット名 を参照してください。

ローカル Windows マシンから呼び出した Windows サーバー Amazon EC2 インスタンスの場合

```
aws deploy create-deployment --application-name SimpleDemoApp --deployment-config-
name CodeDeployDefault.OneAtATime --deployment-group-name SimpleDemoDG --s3-
location bucket=bucket-name,bundleType=zip,key=samples/latest/SampleApp_Windows.zip
```

bucket-name は、リージョンの CodeDeploy リソースキットファイルが含まれている Amazon S3 バケットの名前です。例えば、米国東部 (オハイオ) リージョンの場合、##### を aws-codedeploy-us-east-2 に置き換えます。バケット名のリストについては、リージョン別リソースキットバケット名 を参照してください。

> **Note**
>
> 現在のところ、CodeDeploy は Ubuntu サーバーの Amazon EC2 インスタンスにデプロイするサンプルリビジョンを提供していません。リビジョンを独自に作成するには、CodeDeploy のアプリケーションリビジョンの操作 を参照してください。

5. get-deployment コマンドを呼び出して、デプロイが成功したことを確認します。

   このコマンドを呼び出す前に、create-deployment コマンドの呼び出しで返された、デプロイの ID が必要になります。デプロイ ID を再度取得することが必要な場合には、**SimpleDemoApp** という名前のアプリケーションと **SimpleDemoDG** と言う名前のデプロイグループに対して、list-deployments コマンドを呼び出します。

   ```
   aws deploy list-deployments --application-name SimpleDemoApp --deployment-group-
   name SimpleDemoDG --query "deployments" --output text
   ```

   次に、デプロイ ID を使用して get-deployment コマンドを呼び出します。

   ```
   aws deploy get-deployment --deployment-id deployment-id --query
    "deploymentInfo.status" --output text
   ```

   Succeeded の値が返されるまで続けないでください。

デプロイを作成するには (コンソール)

1. ステップ 2: CodeDeployのサービスのロールを作成する の手順に従ってサービスロールを作成している必要があります。サービスロールは、インスタンスにアクセスしてタグを拡張 (読み込み) する許可を CodeDeploy に付与します。CodeDeploy コンソールを使用してアプリケーションリビジョンをデプロイする前に、サービスロール ARN が必要になります。サービスロール ARN を取得するには、サービスロール ARN の取得 (コンソール) の手順に従います。
2. サービスロール ARN があるので、CodeDeploy コンソールを使用して、アプリケーションリビジョンをデプロイできます。

   にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

> **Note**
>
> 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

3. ナビゲーションペインで [デプロイ] を展開し、[アプリケーション] を選択します。
4. [アプリケーションを作成] を選択します。
5. [カスタムアプリケーション] を選択します。
6. [アプリケーション名] に、「**SimpleDemoApp**」と入力します。
7. [コンピューティングプラットフォーム] で [EC2/オンプレミス] を選択します。
8. [アプリケーションを作成] を選択します。
9. [デプロイグループ] タブで、[デプロイグループの作成] を選択します。
10. [Deployment group name] (デプロイグループ名) に「**SimpleDemoDG**」と入力します。
11. [サービスロール] で、 サービスロールの名前を選択します。
12. [デプロイタイプ] で、[インプレース] を選択します。
13. [環境設定] で、[Auto Scaling グループ]、[**CodeDeployDemo-AS-Group**] の順に選択します。
14. [デプロイ設定] で [CodeDeployDefault.OneAtATime] を選択します。
15. [Enable load balancing (ロードバランシングの有効化)] のチェックを外します。
16. デプロイグループの作成 を選択します。
17. デプロイグループページで、[デプロイの作成] を選択します。
18. [Revision type (リビジョンのタイプ)] の横の [My application is stored in Amazon S3 (Amazon S3 に保存されているアプリケーション)] を選択します。
19. リビジョンの場所で、オペレーティングシステムとリージョンのサンプルアプリケーションの場所を入力します。

    Amazon Linux、RHEL Amazon EC2 インスタンスの場合

リージョン	サンプルアプリケーションの場所
米国東部 (オハイオ) リージョン	`http://s3-us-east-2.amazonaws.com/aws-codedeploy-us-east-2/samples/latest/SampleApp_Linux.zip`

リージョン	サンプルアプリケーションの場所
米国東部(バージニア州北部) リージョン	http://s3.amazonaws.com/aws-codedeploy-us-east-1/samples/latest/SampleApp_Linux.zip
US West (N. California) Region	http://s3-us-west-1.amazonaws.com/aws-codedeploy-us-west-1/samples/latest/SampleApp_Linux.zip
米国西部 (オレゴン) リージョン	http://s3-us-west-2.amazonaws.com/aws-codedeploy-us-west-2/samples/latest/SampleApp_Linux.zip
カナダ (中部) リージョン	http://s3-ca-central-1.amazonaws.com/aws-codedeploy-ca-central-1/samples/latest/SampleApp_Linux.zip
欧州 (アイルランド) リージョン	http://s3-eu-west-1.amazonaws.com/aws-codedeploy-eu-west-1/samples/latest/SampleApp_Linux.zip
欧州 (ロンドン) リージョン	http://s3-eu-west-2.amazonaws.com/aws-codedeploy-eu-west-2/samples/latest/SampleApp_Linux.zip
欧州(パリ)リージョン	http://s3-eu-west-3.amazonaws.com/aws-codedeploy-eu-west-3/samples/latest/SampleApp_Linux.zip

リージョン	サンプルアプリケーションの場所
欧州(フランクフルト)リージョン	http://s3-eu-central-1.amazonaws.com/aws-codedeploy-eu-central-1/samples/latest/SampleApp_Linux.zip
イスラエル (テルアビブ) リージョン	https://aws-codedeploy-il-central-1.s3.il-central-1.amazonaws.com/samples/latest/SampleApp_Linux.zip
アジアパシフィック (香港) リージョン	https://aws-codedeploy-ap-east-1.s3.ap-east-1.amazonaws.com/samples/latest/SampleApp_Linux.zip
Asia Pacific (Tokyo) Region	http://s3-ap-northeast-1.amazonaws.com/aws-codedeploy-ap-northeast-1/samples/latest/SampleApp_Linux.zip
Asia Pacific (Seoul) Region	http://s3-ap-northeast-2.amazonaws.com/aws-codedeploy-ap-northeast-2/samples/latest/SampleApp_Linux.zip
アジアパシフィック (シンガポール) リージョン	http://s3-ap-southeast-1.amazonaws.com/aws-codedeploy-ap-southeast-1/samples/latest/SampleApp_Linux.zip
アジアパシフィック (シドニー) リージョン	http://s3-ap-southeast-2.amazonaws.com/aws-codedeploy-ap-southeast-2/samples/latest/SampleApp_Linux.zip

リージョン	サンプルアプリケーションの場所
アジアパシフィック (メルボルン) リージョン	https://aws-codedeploy-ap-southeast-4.s3.ap-southeast-4.amazonaws.com/samples/latest/SampleApp_Linux.zip
アジアパシフィック (ムンバイ) リージョン	http://s3-ap-south-1.amazonaws.com/aws-codedeploy-ap-south-1/samples/latest/SampleApp_Linux.zip
南米 (サンパウロ) リージョン	http://s3-sa-east-1.amazonaws.com/aws-codedeploy-sa-east-1/samples/latest/SampleApp_Linux.zip

Windows Server Amazon EC2 インスタンスの場合

リージョン	サンプルアプリケーションの場所
米国東部 (オハイオ) リージョン	http://s3-us-east-2.amazonaws.com/aws-codedeploy-us-east-2/samples/latest/SampleApp_Windows.zip
米国東部(バージニア州北部) リージョン	http://s3.amazonaws.com/aws-codedeploy-us-east-1/samples/latest/SampleApp_Windows.zip
US West (N. California) Region	http://s3-us-west-1.amazonaws.com/aws-codedeploy-us-west-1/samples/latest/SampleApp_Windows.zip

リージョン	サンプルアプリケーションの場所
米国西部 (オレゴン) リージョン	http://s3-us-west-2.amazonaws.com/aws-codedeploy-us-west-2/samples/latest/SampleApp_Windows.zip
カナダ (中部) リージョン	http://s3-ca-central-1.amazonaws.com/aws-codedeploy-ca-central-1/samples/latest/SampleApp_Windows.zip
欧州 (アイルランド) リージョン	http://s3-eu-west-1.amazonaws.com/aws-codedeploy-eu-west-1/samples/latest/SampleApp_Windows.zip
欧州 (ロンドン) リージョン	http://s3-eu-west-2.amazonaws.com/aws-codedeploy-eu-west-2/samples/latest/SampleApp_Windows.zip
欧州(パリ)リージョン	http://s3-eu-west-3.amazonaws.com/aws-codedeploy-eu-west-3/samples/latest/SampleApp_Windows.zip
欧州(フランクフルト)リージョン	http://s3-eu-central-1.amazonaws.com/aws-codedeploy-eu-central-1/samples/latest/SampleApp_Windows.zip
イスラエル (テルアビブ) リージョン	https://aws-codedeploy-il-central-1.s3.il-central-1.amazonaws.com/samples/latest/SampleApp_Windows.zip

リージョン	サンプルアプリケーションの場所
アジアパシフィック (香港) リージョン	https://aws-codedeploy-ap-east-1.s3.ap-east-1.amazonaws.com/samples/latest/SampleApp_Windows.zip
アジアパシフィック (ソウル) リージョン	http://s3-ap-northeast-2.amazonaws.com/aws-codedeploy-ap-northeast-2/samples/latest/SampleApp_Windows.zip
アジアパシフィック (シンガポール) リージョン	http://s3-ap-southeast-1.amazonaws.com/aws-codedeploy-ap-southeast-1/samples/latest/SampleApp_Windows.zip
アジアパシフィック (シドニー) リージョン	http://s3-ap-southeast-2.amazonaws.com/aws-codedeploy-ap-southeast-2/samples/latest/SampleApp_Windows.zip
アジアパシフィック (メルボルン) リージョン	https://aws-codedeploy-ap-southeast-4.s3.ap-southeast-4.amazonaws.com/samples/latest/SampleApp_Windows.zip
アジアパシフィック (ムンバイ) リージョン	http://s3-ap-south-1.amazonaws.com/aws-codedeploy-ap-south-1/samples/latest/SampleApp_Windows.zip
南米 (サンパウロ) リージョン	http://s3-sa-east-1.amazonaws.com/aws-codedeploy-sa-east-1/samples/latest/SampleApp_Windows.zip

Ubuntu Server Amazon EC2 インスタンスの場合

Amazon S3 に格納されるカスタムアプリケーションリビジョンの場所を入力します。

20. [デプロイメントの説明] は空白のままにしておきます。
21. [Advanced] を展開します。
22. [デプロイの作成] を選択します。

> **Note**
>
> Succeeded の代わりに Failed が表示された場合、デプロイをモニタリングおよびトラブルシューティングします。 にある手法の一部を試してみることもできます (**SimpleDemoApp** のアプリケーション名、および **SimpleDemoDG** のデプロイグループ名を使用して)。

ステップ 3: 結果の確認

このステップでは、CodeDeploy が Auto Scaling グループの単一 Amazon EC2 インスタンスについての **SimpleDemoApp** のリビジョンをインストールしたかどうかを確認します。

トピック

- 結果を確認するには (CLI)
- 結果を確認するには (コンソール)

結果を確認するには (CLI)

まず、Amazon EC2 インスタンスのパブリック DNS が必要です。

を使用して AWS CLI 、 describe-instances コマンドを呼び出して Auto Scaling グループ内の Amazon EC2 インスタンスのパブリック DNS を取得します。

このコマンドを呼び出す前に、Amazon EC2 インスタンスの ID が必要です。この ID を取得するには、以前に行ったように、**CodeDeployDemo-AS-Group** に対して describe-auto-scaling-groups を呼び出します。

```
aws autoscaling describe-auto-scaling-groups --auto-scaling-group-names CodeDeployDemo-
AS-Group --query "AutoScalingGroups[0].Instances[*].InstanceId" --output text
```

次に describe-instances コマンドを呼び出します。

```
aws ec2 describe-instances --instance-id instance-id --query
 "Reservations[0].Instances[0].PublicDnsName" --output text
```

返される値は Amazon EC2 インスタンスのパブリック DNS です。

ウェブブラウザを使用して、次のような URL を使用して Amazon EC2 インスタンスにデプロイした [SimpleDemoApp] リビジョンを表示します。

```
http://ec2-01-234-567-890.compute-1.amazonaws.com
```

成功のページが表示されると、リビジョンは Auto Scaling グループの単一の Amazon EC2 インスタンスに CodeDeploy を使用して正しくデプロイされました。

次に、Amazon EC2 インスタンスを Auto Scaling グループに追加します。Amazon EC2 Auto Scaling が Amazon EC2 インスタンスを追加すると、CodeDeploy は新しいインスタンスにリビジョンをデプロイします。

結果を確認するには (コンソール)

まず、Amazon EC2 インスタンスのパブリック DNS が必要です。

Amazon EC2 コンソールの https://console.aws.amazon.com/ec2/ を開いてください。

Amazon EC2 ナビゲーションペインの Auto Scaling の下で、Auto Scaling グループ を選択し、**CodeDeployDemo-AS-Group** のエントリを選択します。

インスタンス タブで、リスト内の Amazon EC2 インスタンス ID を選択します。

[Instances] ページの、[Description] タブで、[Public DNS] 値をメモします。次のように表示されます。**ec2-01-234-567-890.compute-1.amazonaws.com**

ウェブブラウザを使用して、次のような URL を使用して Amazon EC2 インスタンスにデプロイした [SimpleDemoApp] リビジョンを表示します。

```
http://ec2-01-234-567-890.compute-1.amazonaws.com
```

成功のページが表示されると、リビジョンは Auto Scaling グループの単一の Amazon EC2 インスタンスに CodeDeploy を使用して正しくデプロイされました。

次に、Auto Scaling グループに Amazon EC2 インスタンスを追加します。Amazon EC2 Auto Scaling が Amazon EC2 インスタンスを追加した後、CodeDeploy は新しい Amazon EC2 インスタンスにリビジョンをデプロイします。

ステップ 4: Auto Scaling グループの Amazon EC2 インスタンスの数を増やす

このステップでは、追加の Amazon EC2 インスタンスを作成するように Auto Scaling グループに指示します。Amazon EC2 Auto Scaling がインスタンスを作成すると、CodeDeploy によってリビジョンがデプロイされます。

トピック

- Auto Scaling グループ (CLI) の Amazon EC2 インスタンスの数をスケールアウトする
- デプロイグループ (コンソール) で Amazon EC2 インスタンスの数をスケールアップするには

Auto Scaling グループ (CLI) の Amazon EC2 インスタンスの数をスケールアウトする

1. update-auto-scaling-group のコマンドを呼び出し、**CodeDeployDemo-AS-Group** という名前の Auto Scaling グループの Amazon EC2 インスタンスを、1 から 2 に増やします。

   ローカル Linux、macOS、あるいは Unix マシンについて

   ```
   aws autoscaling update-auto-scaling-group \
     --auto-scaling-group-name CodeDeployDemo-AS-Group \
     --min-size 2 \
     --max-size 2 \
     --desired-capacity 2
   ```

   ローカル Windows マシンの場合

   ```
   aws autoscaling update-auto-scaling-group --auto-scaling-group-name CodeDeployDemo-AS-Group --min-size 2 --max-size 2 --desired-capacity 2
   ```

2. Auto Scaling グループに 2 つの Amazon EC2 インスタンスが存在することを確認します。**CodeDeployDemo-AS-Group** に対して describe-auto-scaling-groups コマンドを呼び出します。

```
aws autoscaling describe-auto-scaling-groups --auto-scaling-group-names
 CodeDeployDemo-AS-Group --query "AutoScalingGroups[0].Instances[*].[HealthStatus,
 LifecycleState]" --output text
```

戻り値に、Healthy と InService の両方が表示されるまで続行しないでください。

デプロイグループ (コンソール) で Amazon EC2 インスタンスの数をスケールアップするには

1. Amazon EC2 ナビゲーションバーで、Auto Scaling の下で、Auto Scaling Groups を選択し、次に **CodeDeployDemo-AS-Group** を選択します。
2. [アクション] を選択し、[編集] を選択します。
3. [詳細] タブで、[必要]、[最小] および [最大] ボックスに **2** と入力し、[保存] を選択します。
4. [Instances] タブを選択します。新しい Amazon EC2 インスタンスが一覧に表示されます。(インスタンスが表示されない場合、[Refresh] ボタンを数回選択する必要が生じる場合があります)。[Lifecycle] 列に [InService] の値が表示されて、[Health Status] 列に、[Healthy] の値が表示されるまで進まないでください。

ステップ 5: 結果を再度確認します

このステップでは、CodeDeploy が Auto Scaling グループの新しいインスタンスの [SimpleDemoApp] リビジョンをインストールしたかどうかを確認します。

トピック

- 自動デプロイの結果を確認するには (CLI)
- 自動デプロイの結果を確認するには (コンソール)

自動デプロイの結果を確認するには (CLI)

1. get-deployment コマンドを確認する前に、自動デプロイの ID が必要です。ID を取得するには、**SimpleDemoApp** という名前のアプリケーションと **SimpleDemoDG** と言う名前のデプロイグループに対して list-deployments コマンドを呼び出します。

```
aws deploy list-deployments --application-name SimpleDemoApp --deployment-group-
name SimpleDemoDG --query "deployments" --output text
```

2 つのデプロイ ID があるはずです。get-deployment コマンドの呼び出しでは、まだ使用していない ID を使用してください。

```
aws deploy get-deployment --deployment-id deployment-id --query "deploymentInfo.
[status, creator]" --output text
```

デプロイのステータスに加えて、autoScaling がコマンド出力に表示されます (autoScaling は Amazon EC2 Auto Scaling が作成したデプロイ)。

デプロイのステータスに Succeeded が表示されるまで進まないでください。

2. describe-instances のコマンドを呼び出す前に、新しい Amazon EC2 インスタンスの ID が必要です。この ID を取得するには、**CodeDeployDemo-AS-Group** に対して describe-auto-scaling-groups コマンドをもう一度呼び出します。

```
aws autoscaling describe-auto-scaling-groups --auto-scaling-group-names
 CodeDeployDemo-AS-Group --query "AutoScalingGroups[0].Instances[*].InstanceId" --
output text
```

次に describe-instances コマンドを呼び出します。

```
aws ec2 describe-instances --instance-id instance-id --query
 "Reservations[0].Instances[0].PublicDnsName" --output text
```

describe-instances のコマンドの出力で、新しい Amazon EC2 インスタンスの パブリック DNS をメモします。

3. ウェブブラウザを使用して、次のような URL を使用して Amazon EC2 インスタンスにデプロイした SimpleDemoApp のリビジョンを表示します。

```
http://ec2-01-234-567-890.compute-1.amazonaws.com
```

成功のページが表示されると、CodeDeploy によってリビジョンが Auto Scaling グループのスケールアップされた Amazon EC2 インスタンスに正常にデプロイされています。

自動デプロイの結果を確認するには (コンソール)

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで、Deploy を展開し、Deployments を選択します。

3. Amazon EC2 Auto Scaling が作成したデプロイのデプロイ ID を選択します。

   .

4. [デプロイ] ページに、デプロイに関する情報が表示されます。通常の場合は、ユーザーがデプロイを作成しますが、Amazon EC2 Auto Scaling はユーザーに代わってリビジョンを新しい Amazon EC2 インスタンスにデプロイします。
5. ページ上部に [成功] と表示されたら、インスタンスで結果を確認します。最初に、インスタンスのパブリック DNS を取得する必要があります。
6. Amazon EC2 ナビゲーションペインの Auto Scaling の下で、Auto Scaling グループ を選択し、**CodeDeployDemo-AS-Group** のエントリを選択します。
7. Instances タブで、新しい Amazon EC2 インスタンスの ID を選択します。
8. [Instances] ページの、[Description] タブで、[Public DNS] 値をメモします。次のように表示されます。**ec2-01-234-567-890.compute-1.amazonaws.com**

インスタンスにデプロイされた SimpleDemoApp リビジョンを、次のような URL を使用して表示します。

```
http://ec2-01-234-567-890.compute-1.amazonaws.com
```

成功のページが表示されると、CodeDeploy によってリビジョンが Auto Scaling グループのスケールアップされた Amazon EC2 インスタンスに正常にデプロイされています。

ステップ 6: クリーンアップする

このステップでは、このチュートリアルで使用したリソースの料金が継続的に発生するのを避けるために Auto Scaling グループを削除します。必要に応じて、Auto Scaling 設定、および CodeDeploy デプロイコンポーネントレコードを削除できます。

トピック

- リソース (CLI) をクリーンアップするには
- リソース (コンソール) をクリーンアップするには

リソース (CLI) をクリーンアップするには

1. **CodeDeployDemo-AS-Group** に対して delete-auto-scaling-group コマンドを呼び出すことによって、Auto Scaling グループを削除し ます。これにより、Amazon EC2 インスタンスも終了します。

```
aws autoscaling delete-auto-scaling-group --auto-scaling-group-name CodeDeployDemo-AS-Group --force-delete
```

2. 必要に応じて、**CodeDeployDemo-AS-Launch-Template** という名前の起動設定に対して、delete-launch-template のコマンドを呼び出し、Auto Scaling 起動テンプレートを削除します。

```
aws ec2 delete-launch-template --launch-template-name CodeDeployDemo-AS-Launch-Template
```

3. 必要に応じて、delete-application という名前のアプリケーションに対して、**SimpleDemoApp** のコマンドを呼び出し、CodeDeploy からのアプリケーションを削除します。これにより、関連するすべてのデプロイ、デプロイグループ、およびリビジョンレコードも削除されます。

```
aws deploy delete-application --application-name SimpleDemoApp
```

4. Systems Manager ステートマネージャーの関連付けを削除するには、delete-association のコマンドを呼び出します。

```
aws ssm delete-association --assocation-id association-id
```

describe-association のコマンドを呼び出すことで、*association-id* を取得できます。

```
aws ssm describe-association --name AWS-ConfigureAWSPackage --targets
 Key=tag:Name,Values=CodeDeployDemo
```

リソース (コンソール) をクリーンアップするには

Amazon EC2 インスタンスを終了する Auto Scaling グループを削除するには

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/ec2/ で Amazon EC2 コンソールを開きます。
2. Amazon EC2 ナビゲーションペインで、Auto Scaling の下で、Auto Scaling Groups を選択してから、**CodeDeployDemo-AS-Group** のエントリを選択します。
3. [Actions] を選択して、[Delete] を選択し、次に [Yes, Delete] を選択します。

(オプション) 起動テンプレートを削除するには

1. ナビゲーションバーで、Auto Scaling の下で、Launch Configurations を選択し、**CodeDeployDemo-AS-Launch-Template** を選択します。
2. [Actions] を選択して、[Delete launch configuration] を選択し、[Yes, Delete] を選択します。

1. 必要に応じて、CodeDeploy からアプリケーションを削除します。これにより、関連するすべてのデプロイ、デプロイグループ、およびリビジョンレコードも削除されます。https://console.aws.amazon.com/codedeploy で、CodeDeploy コンソールを開きます。
2. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

> **Note**
>
> 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

ナビゲーションペインで [デプロイ] を展開し、[アプリケーション] を選択します。

3. アプリケーションのリストで、SimpleDemoApp を選択します。
4. [Application details] ページで、[Delete application] を選択します。

5. 確認を求めるメッセージが表示されたら、**Delete**と入力し、[削除] を選択してください。

Systems Manager ステートマネージャーの関連付けの削除。

1. https://console.aws.amazon.com/systems-manager で AWS Systems Manager コンソールを開きます。
2. ナビゲーションペインで、[ステートマネージャー] を選択してください。
3. 作成した関連付けを選択し、[削除] を選択します。

チュートリアル: CodeDeploy を使用して、GitHub からアプリケーションをデプロイします。

このチュートリアルでは、CodeDeploy を使用して GitHub からサンプルアプリケーションリビジョンを、Amazon Linux を実行している単一の Amazon EC2 インスタンス、単一の Red Hat Enterprise Linux (RHEL) インスタンス、または単一の Windows サーバーインスタンスにデプロイします。GitHub と CodeDeploy の統合については、GitHub と CodeDeploy との統合 を参照してください。

> **Note**
>
> CodeDeploy を使用して、アプリケーションリビジョンを GitHub から Ubuntu サーバーインスタンスにデプロイすることもできます。ステップ 2: サンプルのアプリケーションリビジョンを作成する の チュートリアル: CodeDeploy (Windows サーバー、Ubuntu サーバー、または Red Hat エンタープライズ Linux) を使用してオンプレミスインスタンスにアプリケーションをデプロイします。 の中に記述されているサンプルリビジョンを使用するか、Ubuntu サーバーインスタンスおよび CodeDeploy と互換性があるリビジョンを作成できます。独自のリビジョンを作成するには、CodeDeploy のリビジョンを計画する と CodeDeploy 用のアプリケーション仕様ファイルをリビジョンに追加 を参照してください。

トピック

- 前提条件
- ステップ 1: GitHub アカウントを設定します。
- ステップ 2: GitHub リポジトリを作成します。
- ステップ 3: GitHub リポジトリにサンプルアプリケーションをアップロードします。

- ステップ 4: インスタンスをプロビジョニングします。
- ステップ 5: アプリケーションおよびデプロイグループを作成します。
- ステップ 6: アプリケーションをインスタンスにデプロイします。
- ステップ 7: デプロイをモニタリングおよび確認します。
- ステップ 8: クリーンアップする

前提条件

このチュートリアルを開始する前に、以下を実行します。

- ローカルマシンで Git をインストールします。Git をインストールするには、Git downloads を参照してください。
- AWS CLIのインストールと設定を含む、「CodeDeploy の開始方法」の手順を完了します。これは、 を使用して GitHub からインスタンス AWS CLI にリビジョンをデプロイする場合に特に重要です。

ステップ 1: GitHub アカウントを設定します。

リビジョンが保存される GitHub リポジトリを作成するには、GitHub アカウントが必要です。すでに GitHub アカウントをお持ちの場合は、ステップ 2: GitHub リポジトリを作成します。 に進んでください。

1. https://github.com/join にアクセスします。
2. ユーザー名、E メールアドレス、パスワードを入力します。
3. [Sign up for GitHub] を選択し、指示に従います。

ステップ 2: GitHub リポジトリを作成します。

リビジョンを保存するには、GitHub リポジトリが必要です。

すでに GitHub リポジトリがある場合は、このチュートリアル全体で **CodeDeployGitHubDemo** の名前を必ず置き換えて、「ステップ 3: GitHub リポジトリにサンプルアプリケーションをアップロードします。」に進んでください。

1. GitHub home page で、次のいずれかの操作を実行します。

- [Your repositories] で、[New repository] を選択します。
- ナビゲーションバーで、[Create new (+)] を選択し、[New repository] を選択します。

2. [Create a new repository] ページで、次の操作を実行します。

- [リポジトリ名] ボックスに「**CodeDeployGitHubDemo**」と入力します。
- [Public] を選択します。

> **Note**
>
> デフォルトの [Public] オプションを選択すると、誰でもこのリポジトリを表示できます。[プライベート] オプションを選択して、リポジトリを表示してコミットできるユーザーを制限できます。

- [Initialize this repository with a README] チェックボックスをオフにします。代わりに、次のステップでは、`README.md` ファイルを手動で作成します。
- [Create repository (リポジトリの作成)] を選択します。

3. ローカルマシンタイプ別の手順に従い、コマンドラインを使用してリポジトリを作成します。

> **Note**
>
> GitHub で 2 要素認証を有効にした場合、パスワードの入力を求められたら、GitHub のログインパスワードの代わりに必ず個人アクセストークンを入力するようにしてください。詳細については、Providing your 2FA authentication code を参照してください。

ローカル Linux、macOS、Unix マシンについて

1. ターミナルから、次のコマンドを一度に 1 つずつ実行します。*user-name* は GitHub のユーザー名です。

```
mkdir /tmp/CodeDeployGitHubDemo
```

```
cd /tmp/CodeDeployGitHubDemo
```

```
touch README.md
```

```
git init
```

```
git add README.md
```

```
git commit -m "My first commit"
```

```
git remote add origin https://github.com/user-name/CodeDeployGitHubDemo.git
```

```
git push -u origin master
```

2. ターミナルを /tmp/CodeDeployGitHubDemo の場所で開いたままにします。

ローカル Windows マシンの場合

1. 管理者として実行するコマンドプロンプトから、次のコマンドを一度に 1 つずつ実行します。

```
mkdir c:\temp\CodeDeployGitHubDemo
```

```
cd c:\temp\CodeDeployGitHubDemo
```

```
notepad README.md
```

2. Notepad に README.md ファイルを保存します。Notepad を閉じます。次のコマンドを一度に 1 つずつ実行します。*user-name* は GitHub のユーザー名です。

```
git init
```

```
git add README.md
```

```
git commit -m "My first commit"
```

```
git remote add origin https://github.com/user-name/CodeDeployGitHubDemo.git
```

```
git push -u origin master
```

3. コマンドプロンプトを c:\temp\CodeDeployGitHubDemo の場所で開いたままにします。

ステップ 3: GitHub リポジトリにサンプルアプリケーションをアップロードします。

このステップでは、パブリックの Amazon S3 バケットから GitHub リポジトリにサンプルリビジョンをコピーします。(分かりやすいように、このチュートリアルに用意してあるサンプルリビジョンは単一のウェブページです。)

> **Note**
>
> サンプルリビジョンの代わりに自身のリビジョンの 1 つを使用する場合は、以下が必要です。
>
> - CodeDeploy のリビジョンを計画する と CodeDeploy 用のアプリケーション仕様ファイルをリビジョンに追加 のガイドラインに従う。
> - 対応するインスタンスタイプを使用する。
> - GitHub のダッシュボードからアクセス可能である。
>
> リビジョンがこれらの要件を満たしている場合は、「ステップ 5: アプリケーションおよびデプロイグループを作成します。」に進んでください。
> Ubuntu インスタンスにデプロイする場合は、Ubuntu サーバーインスタンスおよび CodeDeploy と互換性があるリビジョンを GitHub リポジトリにアップロードする必要があります。詳細については、「CodeDeploy のリビジョンを計画する」および「CodeDeploy 用のアプリケーション仕様ファイルをリビジョンに追加」を参照してください。

トピック

- ローカル Linux、macOS、あるいは Unix マシンからサンプルリビジョンをプッシュします
- ローカル Windows マシンからサンプルリビジョンをプッシュする

ローカル Linux、macOS、あるいは Unix マシンからサンプルリビジョンをプッシュします

ターミナルを /tmp/CodeDeployGitHubDemo などの場所で開いたままにして、以下のコマンドを一度に 1 つずつ実行します。

> **Note**
>
> デプロイ先を Windows サーバーインスタンスにする場合は、コマンドで SampleApp_Windows.zip の代わりに SampleApp_Linux.zip を使用します。

```
(Amazon S3 copy command)
```

```
unzip SampleApp_Linux.zip
```

```
rm SampleApp_Linux.zip
```

```
git add .
```

```
git commit -m "Added sample app"
```

```
git push
```

(Amazon S3 copy command) は次のいずれかです。

- aws s3 cp s3://aws-codedeploy-us-east-2/samples/latest/SampleApp_Linux.zip . --region us-east-2 (米国東部 (オハイオ) リージョンの場合)
- aws s3 cp s3://aws-codedeploy-us-east-1/samples/latest/SampleApp_Linux.zip . --region us-east-1 (米国東部 (バージニア北部) リージョンの場合)
- aws s3 cp s3://aws-codedeploy-us-west-1/samples/latest/SampleApp_Linux.zip . --region us-west-1 (米国西部 (北カリフォルニア) リージョンの場合)

- aws s3 cp s3://aws-codedeploy-us-west-2/samples/latest/SampleApp_Linux.zip . --region us-west-2 (米国西部 (オレゴン) リージョンの場合)
- aws s3 cp s3://aws-codedeploy-ca-central-1/samples/latest/SampleApp_Linux.zip . --region ca-central-1 (カナダ (中部) リージョンの場合)
- aws s3 cp s3://aws-codedeploy-eu-west-1/samples/latest/SampleApp_Linux.zip . --region eu-west-1 欧州 (アイルランド) リージョン用
- aws s3 cp s3://aws-codedeploy-eu-west-2/samples/latest/SampleApp_Linux.zip . --region eu-west-2 欧州 (ロンドン) リージョン用
- aws s3 cp s3://aws-codedeploy-eu-west-3/samples/latest/SampleApp_Linux.zip . --region eu-west-3 欧州 (パリ) リージョン用
- aws s3 cp s3://aws-codedeploy-eu-central-1/samples/latest/SampleApp_Linux.zip . --region eu-central-1 (欧州 (フランクフルト) リージョンの場合)
- aws s3 cp s3://aws-codedeploy-il-central-1/samples/latest/SampleApp_Linux.zip . --region il-central-1 (イスラエル (テルアビブ) リージョンの場合)
- aws s3 cp s3://aws-codedeploy-ap-east-1/samples/latest/SampleApp_Linux.zip . --region ap-east-1 アジアパシフィック (香港) リージョン用
- aws s3 cp s3://aws-codedeploy-ap-northeast-1/samples/latest/SampleApp_Linux.zip . --region ap-northeast-1 アジアパシフィック (東京) リージョン用
- aws s3 cp s3://aws-codedeploy-ap-northeast-2/samples/latest/SampleApp_Linux.zip . --region ap-northeast-2 アジアパシフィック (ソウル) リージョン用
- aws s3 cp s3://aws-codedeploy-ap-southeast-1/samples/latest/SampleApp_Linux.zip . --region ap-southeast-1 (アジアパシフィック (シンガポール) リージョンの場合)
- aws s3 cp s3://aws-codedeploy-ap-southeast-2/samples/latest/SampleApp_Linux.zip . --region ap-southeast-2 アジアパシフィック (シドニー) リージョン用
- aws s3 cp s3://aws-codedeploy-ap-southeast-4/samples/latest/SampleApp_Linux.zip . --region ap-southeast-4 アジアパシフィック (メルボルン) リージョン用

- `aws s3 cp s3://aws-codedeploy-ap-south-1/samples/latest/SampleApp_Linux.zip . --region ap-south-1` アジアパシフィック (ムンバイ) リージョン用
- `aws s3 cp s3://aws-codedeploy-sa-east-1/samples/latest/SampleApp_Linux.zip . --region sa-east-1` (南米 (サンパウロ) リージョンの場合)

ローカル Windows マシンからサンプルリビジョンをプッシュする

コマンドプロンプトを `c:\temp\CodeDeployGitHubDemo`の場所などで開いたままにして、次のコマンドを一度に 1 つずつ実行します。

> **Note**
>
> デプロイ先を Amazon Linux または RHEL インスタンスにデプロイする予定がある場合は、コマンドで `SampleApp_Linux.zip` の代わりに `SampleApp_Windows.zip` を使用します。

```
(Amazon S3 copy command)
```

the ZIP ファイルの内容の解凍先を、新しいサブディレクトリではなく、直接ローカルディレクトリ (`c:\temp\CodeDeployGitHubDemo` など) にします。

```
git add .
```

```
git commit -m "Added sample app"
```

```
git push
```

(Amazon S3 copy command) は次のいずれかです。

- `aws s3 cp s3://aws-codedeploy-us-east-2/samples/latest/SampleApp_Windows.zip . --region us-east-2` (米国東部 (オハイオ) リージョンの場合)
- `aws s3 cp s3://aws-codedeploy-us-east-1/samples/latest/SampleApp_Windows.zip . --region us-east-1` (米国東部 (バージニア北部) リージョンの場合)

- aws s3 cp s3://aws-codedeploy-us-west-1/samples/latest/SampleApp_Windows.zip . --region us-west-1 (米国西部 (北カリフォルニア) リージョンの場合)
- aws s3 cp s3://aws-codedeploy-us-west-2/samples/latest/SampleApp_Windows.zip . --region us-west-2 (米国西部 (オレゴン) リージョンの場合)
- aws s3 cp s3://aws-codedeploy-ca-central-1/samples/latest/SampleApp_Windows.zip . --region ca-central-1 (カナダ (中部) リージョンの場合)
- aws s3 cp s3://aws-codedeploy-eu-west-1/samples/latest/SampleApp_Windows.zip . --region eu-west-1 欧州 (アイルランド) リージョン用
- aws s3 cp s3://aws-codedeploy-eu-west-2/samples/latest/SampleApp_Windows.zip . --region eu-west-2 欧州 (ロンドン) リージョン用
- aws s3 cp s3://aws-codedeploy-eu-west-3/samples/latest/SampleApp_Windows.zip . --region eu-west-3 欧州 (パリ) リージョン用
- aws s3 cp s3://aws-codedeploy-eu-central-1/samples/latest/SampleApp_Windows.zip . --region eu-central-1 (欧州 (フランクフルト) リージョンの場合)
- aws s3 cp s3://aws-codedeploy-il-central-1/samples/latest/SampleApp_Windows.zip . --region il-central-1 (イスラエル (テルアビブ) リージョンの場合)
- aws s3 cp s3://aws-codedeploy-ap-east-1/samples/latest/SampleApp_Windows.zip . --region ap-east-1 アジアパシフィック (香港) リージョン用
- aws s3 cp s3://aws-codedeploy-ap-northeast-1/samples/latest/SampleApp_Windows.zip . --region ap-northeast-1 アジアパシフィック (東京) リージョン用
- aws s3 cp s3://aws-codedeploy-ap-northeast-2/samples/latest/SampleApp_Windows.zip . --region ap-northeast-2 アジアパシフィック (ソウル) リージョンの
- aws s3 cp s3://aws-codedeploy-ap-southeast-1/samples/latest/SampleApp_Windows.zip . --region ap-southeast-1 (アジアパシフィック (シンガポール) リージョンの場合)
- aws s3 cp s3://aws-codedeploy-ap-southeast-2/samples/latest/SampleApp_Windows.zip . --region ap-southeast-2 アジアパシフィック (シドニー) リージョン用

- `aws s3 cp s3://aws-codedeploy-ap-southeast-4/samples/latest/SampleApp_Windows.zip . --region ap-southeast-4` アジアパシフィック (メルボルン) リージョン用
- `aws s3 cp s3://aws-codedeploy-ap-south-1/samples/latest/SampleApp_Windows.zip . --region ap-south-1` アジアパシフィック (ムンバイ) リージョン用
- `aws s3 cp s3://aws-codedeploy-sa-east-1/samples/latest/SampleApp_Windows.zip . --region sa-east-1` (南米 (サンパウロ) リージョンの場合)

独自のリビジョンを Ubuntu サーバーインスタンスにプッシュするには、リビジョンをローカルリポジトリにコピーしてから、次のコマンドを呼び出します。

```
git add .
git commit -m "Added Ubuntu app"
git push
```

ステップ 4: インスタンスをプロビジョニングします。

このステップでは、サンプルアプリケーションのデプロイ先であるインスタンスを作成または設定します。CodeDeploy でサポートされているオペレーティングシステムのいずれかを実行している Amazon EC2 インスタンスまたはオンプレミスインスタンスにデプロイできます。詳細については、CodeDeploy エージェントで対応するオペレーティングシステム を参照してください。(CodeDeploy デプロイで使用するインスタンスが設定済みである場合は、次のステップまでスキップします。)

インスタンスをプロビジョニングするには

1. Amazon EC2 インスタンス (コンソール) を起動します。 の指示に従って、インスタンスをプロビジョニングします。
2. インスタンスを起動するときは、タグの追加 ページで必ずタグを指定してください。タグを指定する方法の詳細については、Amazon EC2 インスタンス (コンソール) を起動します。 を参照してください。

CodeDeploy エージェントがインスタンスで実行されていることを確認するには、

- CodeDeploy エージェントが実行されていることの確認 の指示に従って、エージェントが実行されていることを確認します。

インスタンスを正常にプロビジョンし、CodeDeploy エージェントが実行されていることを確認したら、次のステップに進みます。

ステップ 5: アプリケーションおよびデプロイグループを作成します。

このステップでは、CodeDeploy コンソールまたは AWS CLI を使用して、GitHub リポジトリからサンプルリビジョンをデプロイするために使用するアプリケーションとデプロイグループを作成します。

アプリケーションおよびデプロイグループの作成 (コンソール)

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで Deploy を展開し、Applications を選択します。
3. [アプリケーションの作成]、[カスタムアプリケーション] の順に選択します。
4. [アプリケーション名] に、「**CodeDeployGitHubDemo-App**」と入力します。
5. [コンピューティングプラットフォーム] で [EC2/オンプレミス] を選択します。
6. [アプリケーションを作成] を選択します。
7. [デプロイグループ] タブで、[デプロイグループの作成] を選択します。
8. [Deployment group name] (デプロイグループ名) に「**CodeDeployGitHubDemo-DepGrp**」と入力します。
9. Service role で、CodeDeploy のサービスロールを作成 で作成した CodeDeploy サービスロールの名前を選択します。
10. [デプロイタイプ] で、[インプレース] を選択します。
11. Environment configuration で、使用するインスタンスのタイプに応じて、Amazon EC2 instances あるいは On-premises instances を選択します。[キー] と [値] に、ステップ 4: インスタンスをプロビジョニングします。 の一部としてインスタンスに適用されたインスタンスタグのキーと値を入力します。
12. [デプロイ設定] で [CodeDeployDefault.AllatOnce] を選択します。

13. [ロードバランサー] で、[Enable load balancing (ロードバランシングの有効化)] をオフにします。
14. [Advanced] を展開します。
15. [アラーム] で [アラーム設定を無視する] を選択します。
16. [デプロイグループの作成] を選択し、次のステップに進みます。

アプリケーションおよびデプロイグループの作成 (CLI)

1. create-application のコマンドを呼び出して、CodeDeploy で `CodeDeployGitHubDemo-App` と言う名前のアプリケーションを作成します。

```
aws deploy create-application --application-name CodeDeployGitHubDemo-App
```

2. create-deployment-group コマンドを呼び出して `CodeDeployGitHubDemo-DepGrp` と言う名前のデプロイグループを作成します。

   - Amazon EC2 インスタンスにデプロイする場合、*ec2-tag-key* は、ステップ 4: インスタンスをプロビジョニングします。 の一部として Amazon EC2 インスタンスに適用した Amazon EC2 インスタンスのタグキーです。
   - Amazon EC2 インスタンスにデプロイする場合、*ec2-tag-value* は、ステップ 4: インスタンスをプロビジョニングします。 の一部として Amazon EC2 インスタンスに適用した Amazon EC2 インスタンスのタグ値です。
   - オンプレミスインスタンスにデプロイする場合、*on-premises-tag-key* は、「ステップ 4: インスタンスをプロビジョニングします。」の一部としてオンプレミスインスタンスに適用したオンプレミスインスタンスのタグキーです。
   - オンプレミスインスタンスにデプロイする場合、*on-premises-tag-value* は、「ステップ 4: インスタンスをプロビジョニングします。」の一部としてオンプレミスインスタンスに適用したオンプレミスインスタンスのタグ値です。
   - *service-role-arn* は、CodeDeploy のサービスロールを作成 で作成したサービスロールのサービスロール ARN です。(サービスロール ARN の取得 (CLI) の手順に従って、サービスロール ARN を見つけます)。

```
aws deploy create-deployment-group --application-name CodeDeployGitHubDemo-App
 --ec2-tag-filters Key=ec2-tag-key,Type=KEY_AND_VALUE,Value=ec2-tag-value --on-
premises-tag-filters Key=on-premises-tag-key,Type=KEY_AND_VALUE,Value=on-premises-
```

```
tag-value --deployment-group-name CodeDeployGitHubDemo-DepGrp --service-role-
arn service-role-arn
```

> **Note**
>
> create-deployment-group コマンドは、デプロイおよびインスタンス内の指定されたイベントについて、トピックサブスクライバーに Amazon SNS 通知を送信するトリガーの作成をサポートします。このコマンドは、Amazon CloudWatch アラームのモニタリングしきい値が満たされたときにデプロイを自動的にロールバックし、デプロイを停止するアラームを設定するオプションもサポートします。このチュートリアルでは、これらのアクションコマンドは含まれていません。

ステップ 6: アプリケーションをインスタンスにデプロイします。

このステップでは、CodeDeploy コンソールまたは AWS CLI を使用して、GitHub リポジトリからインスタンスにサンプルリビジョンをデプロイします。

リビジョンをデプロイするには (コンソール)

1. [デプロイグループの詳細] ページで、[デプロイの作成] を選択します。
2. [デプロイグループ] で [**CodeDeployGitHubDemo-DepGrp**] を選択します。
3. [リビジョンタイプ] で [GitHub] を選択します。
4. [Connect to GitHub] で、次のいずれかを実行します。
   - GitHub アカウントに対する CodeDeploy アプリケーションの接続を作成するには、ウェブブラウザの別のタブで GitHub からサインアウトします。[GitHub アカウント] に、この接続を識別する名前を入力し、[GitHub に接続] を選択します。CodeDeployGitHubDemo-App という名前のアプリケーションを GitHub で操作することを CodeDeploy に許可するよう求めるメッセージがウェブページに表示されます。ステップ 5 に進みます。
   - 作成済みの接続を使用するには、その名前を [GitHub account] で選択してから [Connect to GitHub] を選択します。ステップ 7 に進みます。
   - 別の GitHub アカウントへの接続を作成するには、ウェブブラウザの別のタブで GitHub からサインアウトします。[Connect to a different GitHub account] を選択し、[Connect to GitHub] を選択します。ステップ 5 に進みます。
5. [Sign in] ページの手順に従って、GitHub アカウントにサインインします。

6. [Authorize application] ページで、[Authorize application] を選択します。
7. CodeDeploy の Create deployment ページ上で、Repository name に、サインインに使用した GitHub のユーザー名に続いてスラッシュ (/)、アプリケーションリビジョンをプッシュしたリポジトリの名前 (例えば ***my-github-user-name*/CodeDeployGitHubDemo** など) を順に入力します。

   入力する値が不確実な場合、または異なるリポジトリを指定する場合:

   a. ウェブブラウザの別のタブで、GitHub dashboard にアクセスします。
   b. [Your repositories] で、ターゲットリポジトリの名前の上にマウスを置きます。GitHub ユーザーまたは組織の名前、スラッシュ (/)、リポジトリの名前の順序でツールヒントが表示されます。この値を [Repository name (リポジトリ名)] に入力します。

   > **Note**
   >
   > [Your repositories (自分のリポジトリ)] にターゲットリポジトリの名前が表示されない場合は、[Search GitHub (GitHub の検索)] ボックスを使用して、ターゲットリポジトリと GitHub ユーザーまたは組織の名前を検索します。

8. [Commit ID (コミット ID)] ボックスに、アプリケーションリビジョンの GitHub へのプッシュに関連付けられているコミットの ID を入力します。

   入力する値が不確実な場合:

   a. ウェブブラウザの別のタブで、GitHub dashboard にアクセスします。
   b. Your repositories で、CodeDeployGitHubDemo を選択します。
   c. コミットのリストで、アプリケーションリビジョンの GitHub へのプッシュに関連付けられているコミット ID を検索してコピーします。通常、この ID は 40 文字で、文字と数字の両方で構成されます (コミット ID の短いバージョンを使用しないでください。通常は、長いバージョンの最初の 10 文字です)。
   d. [Commit ID] ボックスにコミット ID を貼り付けます。

9. [Deploy] を選択して、次のステップに進みます。

リビジョンをデプロイするには (CLI)

GitHub とやり取りする AWS CLI コマンド (次に呼び出す create-deployment コマンドなど) を呼び出す前に、GitHub ユーザーアカウントを使用して`CodeDeployGitHubDemo-App`アプリケーション

の GitHub とやり取りするアクセス許可を CodeDeploy に付与する必要があります。現在、この操作を行うには、CodeDeploy コンソールを使用する必要があります。

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで Deploy を展開し、Applications を選択します。
3. CodeDeployGitHubDemo-App を選択します。
4. [デプロイ] タブで、[デプロイの作成] を選択します。

   > **Note**
   >
   > 新しいデプロイは作成されません。これは現在、GitHub ユーザーアカウントに代わって GitHub を操作するための権限を CodeDeploy に付与する唯一の方法です。

5. CodeDeployGitHubDemo-App から、CodeDeployGitHubDemo-DepGrp を選択します。
6. [リビジョンタイプ] で [GitHub] を選択します。
7. [Connect to GitHub] で、次のいずれかを実行します。

   - GitHub アカウントに対する CodeDeploy アプリケーションの接続を作成するには、ウェブブラウザの別のタブで GitHub からサインアウトします。[GitHub account] に、この接続を識別する名前を入力し、[Connect to GitHub] を選択します。`CodeDeployGitHubDemo-App` という名前のアプリケーションの GitHub を操作することを CodeDeployに許可するよう求めるメッセージがウェブページに表示されます。ステップ 8 に進みます。
   - 作成済みの接続を使用するには、その名前を [GitHub account] で選択してから [Connect to GitHub] を選択します。ステップ 10 に進みます。
   - 別の GitHub アカウントへの接続を作成するには、ウェブブラウザの別のタブで GitHub からサインアウトします。[Connect to a different GitHub account] を選択し、[Connect to GitHub] を選択します。ステップ 8 に進みます。

8. [Sign in] ページの手順に従って、GitHub のユーザー名、または E メールとパスワードでサインインします。
9. [Authorize application] ページで、[Authorize application] を選択します。

10. CodeDeploy の Create deployment ページで 、Cancel を選択します。
11. create-deployment コマンドを呼び出して、リビジョンを GitHub リポジトリからインスタンスにデプロイします。

    - *repository* は、GitHub アカウント名です。スラッシュ (/)、リポジトリの名前が後に続きます (CodeDeployGitHubDemo)。例: MyGitHubUserName/CodeDeployGitHubDemo。

      使用する値が不確実な場合、または異なるリポジトリを指定する場合:

      1. ウェブブラウザの別のタブで、GitHub dashboard にアクセスします。
      2. [Your repositories] で、ターゲットリポジトリの名前の上にマウスを置きます。GitHub ユーザーまたは組織の名前、スラッシュ (/)、リポジトリの名前の順序でツールヒントが表示されます。これが使用する値です。

      > **Note**
      >
      > [Your repositories] にターゲットリポジトリの名前が表示されない場合、[Search GitHub] ボックスを使用して、ターゲットリポジトリと、対応する GitHub ユーザーまたは組織の名前を検索します。

    - *commit-id* は、リポジトリにプッシュしたアプリケーションリビジョンのバージョンに関連付けられているコミットです (例: f835159a...528eb76f)。

      使用する値が不確実な場合:

      1. ウェブブラウザの別のタブで、GitHub dashboard にアクセスします。
      2. Your repositories で、CodeDeployGitHubDemo を選択します。
      3. コミットのリストで、アプリケーションリビジョンの GitHub へのプッシュに関連付けられているコミット ID を検索します。通常、この ID は 40 文字で、文字と数字の両方で構成されます (コミット ID の短いバージョンを使用しないでください。通常は、長いバージョンの最初の 10 文字です)。この値を使用します。

    ローカル Linux、macOS、あるいは Unix マシンを使用している場合

```
aws deploy create-deployment \
  --application-name CodeDeployGitHubDemo-App \
  --deployment-config-name CodeDeployDefault.OneAtATime \
  --deployment-group-name CodeDeployGitHubDemo-DepGrp \
```

```
  --description "My GitHub deployment demo" \
  --github-location repository=repository,commitId=commit-id
```

ローカル Windows マシンを使用している場合

```
aws deploy create-deployment --application-name CodeDeployGitHubDemo-App --
deployment-config-name CodeDeployDefault.OneAtATime --deployment-group-name
 CodeDeployGitHubDemo-DepGrp --description "My GitHub deployment demo" --github-
location repository=repository,commitId=commit-id
```

ステップ 7: デプロイをモニタリングおよび確認します。

このステップでは、CodeDeploy コンソールまたは AWS CLI を使用してデプロイの成功を確認します。作成または設定したインスタンスにデプロイしたウェブページを表示するには、ウェブブラウザを使用します。

> **Note**
>
> Ubuntu インスタンスにデプロイする場合、独自のテスト戦略を使用して、デプロイされたリビジョンがインスタンスで予期したとおりに機能するかどうかを確認し、次のステップに進みます。

デプロイをモニタリングおよび確認するには (コンソール)

1. ナビゲーションペインで Deploy を展開し、Deployments を選択します。

2. デプロイのリストで、CodeDeployGitHubDemo-App の Application の値、および CodeDeployGitHubDemo-DepGrp の CodeDeployGitHubDemo-DepGrp の値が [-DepGrp] である行を探します。Succeeded または Failed が、[Status] 列に表示されない場合は、定期的に [Refresh] ボタンを選択します。
3. Status の列に Failed が表示される場合は、インスタンスの詳細の表示 (コンソール) の指示に従って、デプロイのトラブルシューティングを行います。
4. Status 列で Succeeded が表示される場合は、ウェブブラウザを通してデプロイを確認できます。このサンプルリビジョンでは、インスタンスに単一のウェブページをデプロイします。Amazon EC2 インスタンスにデプロイする場合は、ウェブブラウ

ザで、インスタンスの http://*public-dns* にアクセスします (例えば、http://ec2-01-234-567-890.compute-1.amazonaws.com など)。

5. ウェブページを表示できれば成功です。 AWS CodeDeploy を使用して GitHub からリビジョンが正しくデプロイされました。これで、「ステップ 8: クリーンアップする」に進むことができます。

デプロイをモニタリングおよび確認するには (CLI)

1. list-deployments コマンドを呼び出して、CodeDeployGitHubDemo-App という名前のアプリケーションと CodeDeployGitHubDemo-DepGrp という名前のデプロイグループのデプロイ ID を取得します。

```
aws deploy list-deployments --application-name CodeDeployGitHubDemo-App --
deployment-group-name CodeDeployGitHubDemo-DepGrp --query "deployments" --output
 text
```

2. get-deployment コマンドを呼び出して、list-deployments コマンドからの出力にデプロイメントの ID を指定します。

```
aws deploy get-deployment --deployment-id deployment-id --query "deploymentInfo.
[status, creator]" --output text
```

3. [Failed] が返されたら、インスタンスの詳細の表示 (コンソール) の指示に従って、デプロイのトラブルシューティングを行います。
4. [Succeeded] が返されたら、ウェブブラウザでデプロイを確認できます。このサンプルリビジョンは、インスタンスにデプロイされた単一のウェブページです。Amazon EC2 インスタンスにデプロイする場合は、Amazon EC2 インスタンスのための http://*public-dns* にアクセスすることで、このページをウェブブラウザで表示できます (たとえば、http://ec2-01-234-567-890.compute-1.amazonaws.com など)。
5. ウェブページを表示できれば成功です。 AWS CodeDeploy を使用して GitHub リポジトリから正常にデプロイされました。

ステップ 8: クリーンアップする

このチュートリアルの間に使用したリソースに対する追加料金を防ぐため、Amazon EC2 インスタンスと関連リソースを終了する必要があります。必要に応じて、このチュートリアルに関連付けられ

ている CodeDeploy デプロイコンポーネントレコードを削除できます。このチュートリアル専用に GitHub リポジトリを使用した場合は、今すぐ削除することもできます。

CloudFormation スタックを削除するには (CloudFormation テンプレートを使用して Amazon EC2 インスタンスを作成した場合)

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/cloudformation で CloudFormation コンソールを開きます。
2. [スタック] 列で、`CodeDeploySampleStack` で始まるスタックを選択します。
3. [削除] を選択します。
4. プロンプトが表示されたら、[スタックの削除] を選択します。Amazon EC2 インスタンスおよび関連する IAM インスタンスプロファイルとサービスロールが削除されます。

手動で登録を解除およびオンプレミスインスタンスをクリーンアップするには (オンプレミスインスタンスをプロビジョニングした場合)

1. を使用して AWS CLI 、your-*your-instance-name*-region で関連付けられたリージョンに対して登録解除コマンドを呼び出します。

```
aws deploy deregister --instance-name your-instance-name --no-delete-iam-user --
region your-region
```

2. オンプレミスインスタンスから、uninstall コマンドを呼び出します:

```
aws deploy uninstall
```

Amazon EC2 インスタンスを手動で終了するには (手動で Amazon EC2 インスタンスを起動した場合)

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/ec2/ で Amazon EC2 コンソールを開きます。
2. ナビゲーションペインの [Instances] (インスタンス) で、[Instances] (インスタンス) を選択します。
3. 終了した Amazon EC2 インスタンスの横にあるボックスを選択します。[Actions] メニューで [Instance State] をポイントし、[Terminate] を選択します。

4. プロンプトが表示されたら、[Yes, Terminate] を選択します。

CodeDeploy デプロイコンポーネントレコードを削除するには

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

> **Note**
>
> 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで Deploy を展開し、Applications を選択します。

3. CodeDeployGitHubDemo-App を選択します。
4. [アプリケーションを削除] を選択します。
5. 確認を求めるメッセージが表示されたら、**Delete**と入力し、[削除] を選択してください。

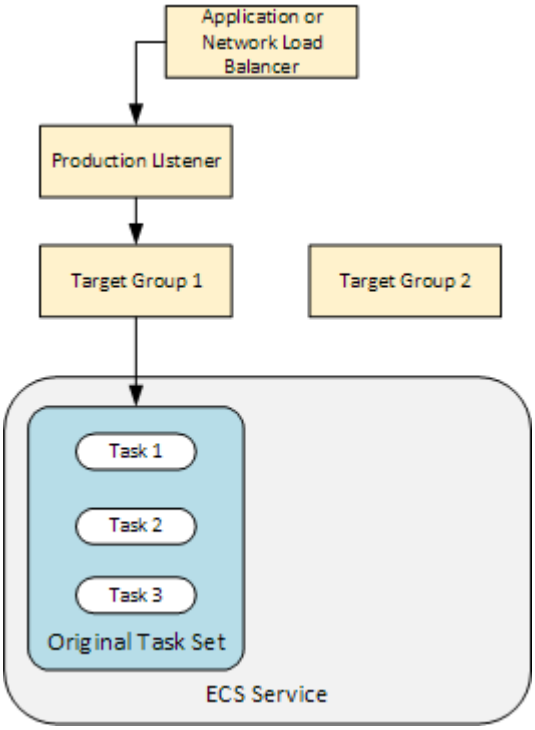
GitHub リポジトリを削除するには

GitHub help の Deleting a repository を参照してください。

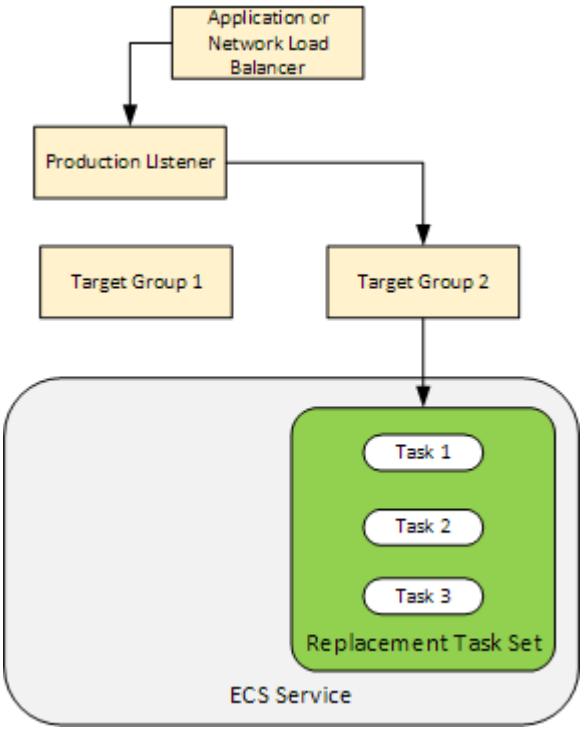
チュートリアル: Amazon ECS へアプリケーションをデプロイする

このチュートリアルでは、CodeDeploy を使用して アプリケーションを Amazon ECS にデプロイする方法について学びます。すでに作成し、Amazon ECS にデプロイ済みのアプリケーションによって開始します。最初のステップは、タスク定義ファイルを新しいタグで変更してアプリケーションを更新することです。次に、CodeDeploy を使用して、更新をデプロイします。デプロイ中に、CodeDeploy は新しい置き換えタスクセットに更新をインストールします。そして、本稼働トラフィックは、元のタスクセットにある Amazon ECS アプリケーションの元のバージョンから、置き換えタスクセットの更新されたバージョンに移行します。

Amazon ECS デプロイ中、CodeDeploy は 2 つのターゲットグループと 1 つの本稼働トラフィックリスナーで構成されたロードバランサーを使用します。次の図表は、デプロイが始まるの前に、ロードバランサー、本稼働、リスナー、ターゲットグループ、および Amazon ECS アプリケーションがどのように関連しているかを示しています。このチュートリアルでは、Application Load Balancer を使用します。Network Load Balancer を使用することもできます。



デプロイが成功すると、本稼働トラフィックリスナーは新しい置き換えタスクセットにトラフィックを提供し、元のタスクセットは終了します。次の図は、デプロイが成功した後にリソースがどのように関連しているかを示しています。詳細については、「Amazon ECS デプロイ中の処理で起こっていること」を参照してください。



を使用してアプリケーションを Amazon ECS に AWS CLI デプロイする方法については、「チュートリアル: Blue/Green デプロイを使用したサービスの作成」を参照してください。CodePipeline を使用して、CodeDeploy を用いての Amazon ECS サービスへの deploy の変更を検出し、自動的にデプロイする方法については、チュートリアル: Create a pipeline with an Amazon ECR source and ECS-to-CodeDeploy deployment を参照してください。

このチュートリアルを完了したら、作成した CodeDeploy アプリケーションとデプロイグループを使用して、チュートリアル: 検証テストを使用して Amazon ECS サービスをデプロイする 中にデプロイ検証テストを追加できます。

トピック

- 前提条件
- ステップ 1: Amazon ECS アプリケーションを更新する
- ステップ 2: AppSpec ファイルを作成します。
- ステップ 3: CodeDeploy コンソールを使用してアプリケーションをデプロイする
- ステップ 4: クリーンアップする

前提条件

このチュートリアルを完了するには、まず以下を行う必要があります。

- 「CodeDeploy の開始方法」のステップ 2 と 3 を完了します。
- 2 つのターゲットグループと 1 つのリスナーを用いて設定した Application Load Balancer を作成します。コンソールを使用してロードバランサーを作成する方法については、「CodeDeploy Amazon ECS デプロイ用のロードバランサー、ターゲットグループ、リスナーをセットアップする」を参照してください。を使用してロードバランサーを作成する方法については AWS CLI、「Amazon Elastic Container Service ユーザーガイド」の「ステップ 1: Application Load Balancer を作成する」を参照してください。ロードバランサーを作成するときは、このチュートリアルで以下の点に注意してください。
  - ロードバランサーの名前。
  - ターゲットグループの名前。
  - ロードバランサーのリスナーが使用するポート。
- Amazon ECS クラスターとサービスを作成します。さらなる詳細については、ステップ 2、3、および Amazon Elastic Container サービスユーザーガイド の チュートリアル: Creating a service

using a blue/green deployment の ステップ 4 を参照してください。このチュートリアルでは、以下の点に注意してください。

- Amazon ECS クラスターの名前。
- Amazon ECS サービスで使用されるタスク定義の ARN。
- Amazon ECS サービスで使用されるコンテナの名前。

- AppSpec ファイル用の Amazon S3 バケットを作成します。

ステップ 1: Amazon ECS アプリケーションを更新する

このセクションでは、タスク定義の新しいリビジョンで Amazon ECS アプリケーションを更新します。更新されたリビジョンは、新しいキーとタグのペアを追加します。ステップ 3: CodeDeploy コンソールを使用してアプリケーションをデプロイする では、Amazon ECS アプリケーションの更新バージョンをデプロイします。

タスク定義を更新するには

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで、[タスク定義] を選択します。
3. Amazon ECS サービスによって使用されるタスク定義を選択します。
4. タスク定義リビジョンを選択し、[新しいリビジョンを作成]、[新しいリビジョンを作成] を選択します。
5. このチュートリアルでは、タグを追加してタスク定義を少し更新します。ページの下部にある [タグ] で、新しいキーと値のペアを入力して新しいタグを作成します。
6. [作成] を選択します。

   タスク定義のリビジョン番号が 1 つずつ増加します。
7. JSON タブを選択します。この情報は次のステップで必要なため、以下の点に注意してください。

   - `taskDefinitionArn` の値。形式は `arn:aws:ecs:`*`aws-region`*`:`*`account-id`*`:task-definition/`*`task-definition-family`*`:`*`task-definition-revision` です。これは、更新されたタスク定義の ARN です。
   - `containerDefinitions` 要素の、`name` の値。これはコンテナの名前です。
   - `portMappings` 要素の、`containerPort` の値。これはコンテナのポートです。

ステップ 2: AppSpec ファイルを作成します。

このセクションでは、AppSpec ファイルを作成し、前提条件 のセクションで作成した Amazon S3 バケットにアップロードします。Amazon ECS デプロイのための AppSpec ファイル は、タスク定義、コンテナ名、およびコンテナポートを指定します。詳細については、「Amazon ECS デプロイの AppSpec ファイルの例」および「Amazon ECS デプロイ用の AppSpec の「resources」セクション」を参照してください。

AppSpec ファイルを作成するには

1. [YAML] を使用して AppSpec ファイルを作成したい場合は、`appspec.yml` という名前のファイルを作成します。JSON を使用して AppSpec ファイルを作成したい場合は、`appspec.json` という名前のファイルを作成します。
2. AppSpec ファイルのために YAML または JSON を使用するかどうかに応じて、適切なタブを選択し、そのコンテンツを、先ほど作成した AppSpec ファイルにコピーします。`TaskDefinition` プロパティには、「ステップ 1: Amazon ECS アプリケーションを更新する」セクションで書き留めたタスク定義 ARN を使用します。

   JSON AppSpec

```
{
  "version": 0.0,
  "Resources": [
    {
      "TargetService": {
        "Type": "AWS::ECS::Service",
        "Properties": {
          "TaskDefinition": "arn:aws:ecs:aws-region-id:aws-account-id:task-
definition/ecs-demo-task-definition:revision-number",
          "LoadBalancerInfo": {
            "ContainerName": "your-container-name",
            "ContainerPort": your-container-port
          }
        }
      }
    }
  ]
}
```

YAML AppSpec

```
version: 0.0
Resources:
  - TargetService:
      Type: AWS::ECS::Service
      Properties:
        TaskDefinition: "arn:aws:ecs:aws-region-id:aws-account-id:task-
definition/ecs-demo-task-definition:revision-number"
        LoadBalancerInfo:
          ContainerName: "your-container-name"
          ContainerPort: your-container-port
```

> **Note**
>
> 置き換えタスクセットは、元のタスクセットからサブネット、セキュリティグループ、プラットフォームバージョン、割り当てられたパブリック IP 値を継承します。AppSpec ファイル中でオプションのプロパティを設定することで、置き換えタスクセットのこれらの値を上書きできます。詳細については、「Amazon ECS デプロイ用の AppSpec の「resources」セクション」および「Amazon ECS デプロイの AppSpec ファイルの例」を参照してください。

3. このチュートリアルの前提条件として、作成済みの S3 バケットに AppSpec ファイルをアップロードします。

ステップ 3: CodeDeploy コンソールを使用してアプリケーションをデプロイする

このセクションでは、CodeDeploy アプリケーションとデプロイグループを作成して、更新されたアプリケーションを Amazon ECS にデプロイします。デプロイ中、CodeDeploy はアプリケーションの本稼働トラフィックを、新しい置き換えタスクセットの新しいバージョンに移行します。このステップを完了するには、以下の項目が必要です。

- Amazon ECS クラスターの名前。
- Amazon ECS サービスの名前。
- Application Load Balancer の名前

- 本稼働リスナーポート。
- ターゲットグループ名。
- 作成した S3 バケットの名前。

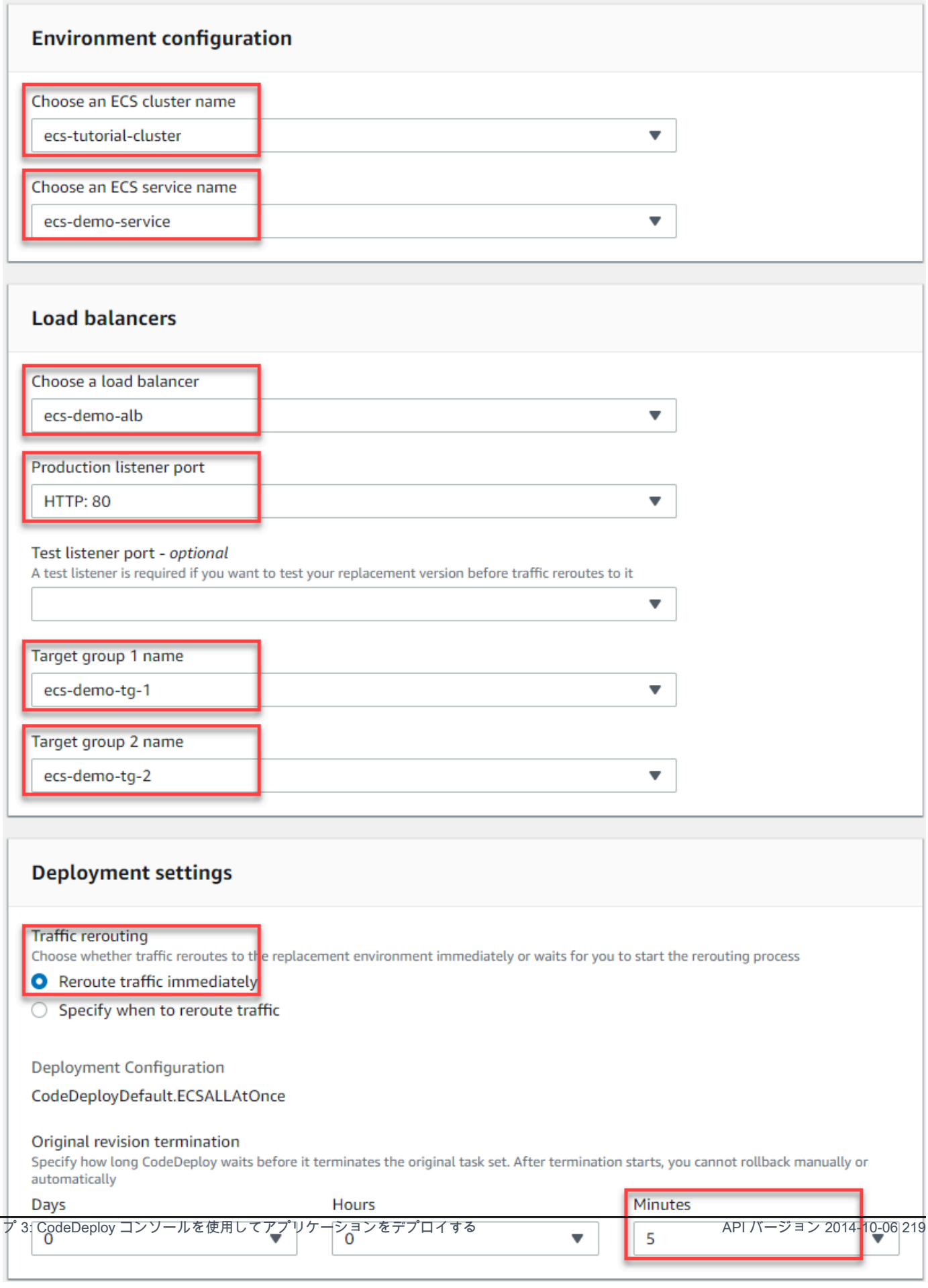
CodeDeploy でアプリケーションを作成するには

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy/ で CodeDeploy コンソールを開きます。
2. [アプリケーションを作成] を選択します。
3. [アプリケーション名] に、「**ecs-demo-codedeploy-app**」と入力します。
4. [コンピューティングプラットフォーム] で [Amazon ECS] を選択します。
5. [アプリケーションを作成] を選択します。

CodeDeploy デプロイ グループを作成するには

1. アプリケーションのページの [デプロイグループ] タブで、[デプロイグループの作成] を選択します。
2. [Deployment group name] (デプロイグループ名) に「**ecs-demo-dg**」と入力します。
3. サービスロール で、CodeDeploy に Amazon ECS へのアクセスを許可するサービスロールを選択します。詳細については、「の ID とアクセスの管理 AWS CodeDeploy」を参照してください。
4. 環境設定 で、Amazon ECS クラスターの名前とサービス名を選択します。
5. Load balancers から、Amazon ECSサービスにトラフィックを提供するロードバランサーの名前を選択します。
6. 本番稼働用リスナーポートから、Amazon ECS サービスへの本番稼働用トラフィックを処理するリスナーのポートとプロトコル (HTTP: 80 など) を選択します。このチュートリアルにはオプションのテストリスナーが含まれていないため、[Test listener port (リスナーポートをテスト)]からポートを選択しないでください。
7. [Target group 1 name (ターゲットグループ 1 の名前)] および [Target group 2 name (ターゲットグループ 2 の名前)] から、デプロイ時にトラフィックをルーティングする 2 つの異なるターゲットグループを選択します。これらが、ロードバランサー用に作成したターゲットグループであることを確認します。どちらがターゲットグループ 1 に使用され、どちらがターゲットグループ 2 に使用されるかは関係ありません。
8. [Reroute traffic immediately (すぐにトラフィックを再ルーティングする)]を選択します。

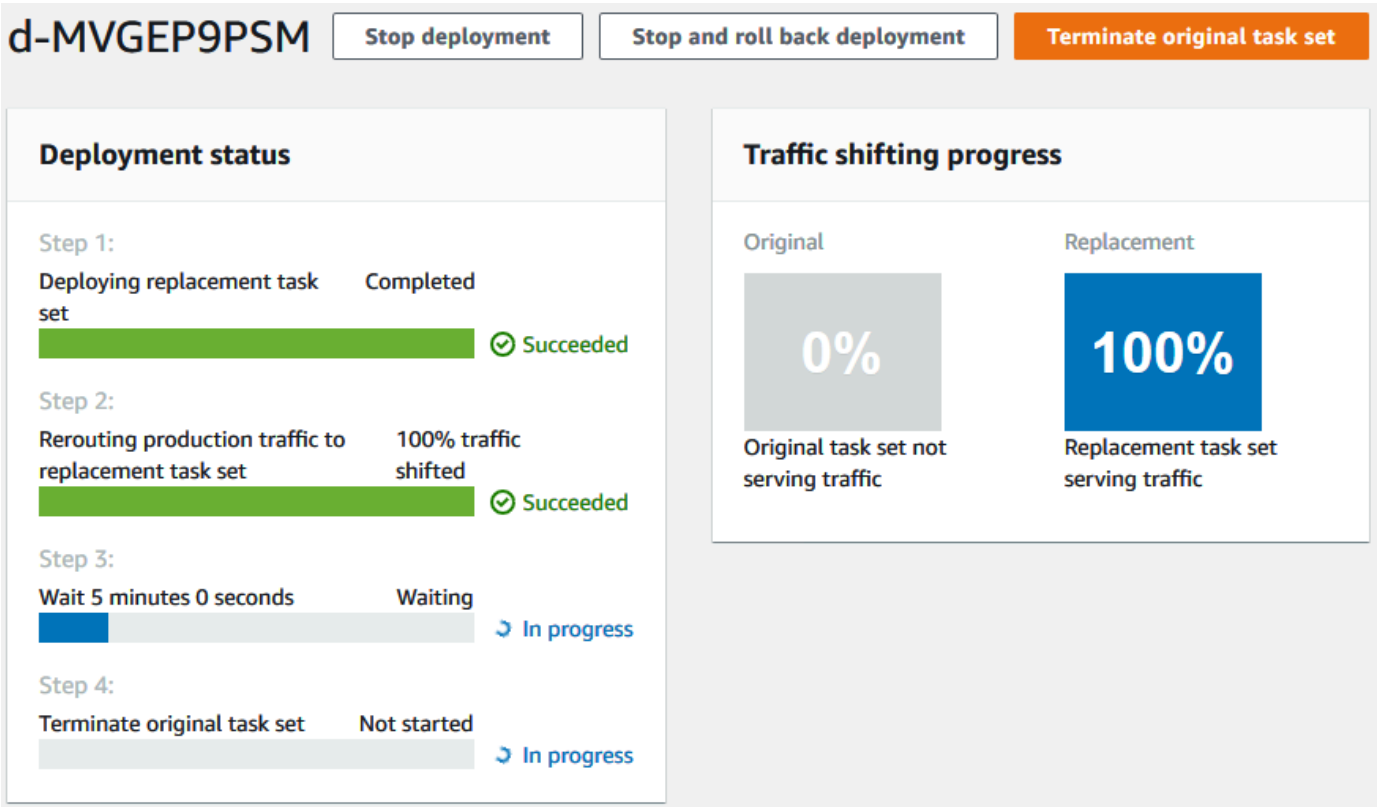
9. [Original revision termination (元のリビジョンの終了)] で、0 日、0 時間、5 分を選択します。これにより、デフォルトの（1 時間）を使用するよりも迅速にデプロイが完了したことがわかります。


Environment configuration
Choose an ECS cluster name
ecs-tutorial-cluster
Choose an ECS service name
ecs-demo-service
Load balancers
Choose a load balancer
ecs-demo-alb
Production listener port
HTTP: 80
Test listener port - optional
A test listener is required if you want to test your replacement version before traffic reroutes to it
Target group 1 name
ecs-demo-tg-1
Target group 2 name
ecs-demo-tg-2
Deployment settings
Traffic rerouting
Choose whether traffic reroutes to the replacement environment immediately or waits for you to start the rerouting process
Reroute traffic immediately
Specify when to reroute traffic
Deployment Configuration
CodeDeployDefault.ECSALLAtOnce
Original revision termination
Specify how long CodeDeploy waits before it terminates the original task set. After termination starts, you cannot rollback manually or automatically
Days
0
Hours
0
Minutes
5

10. デプロイグループの作成 を選択します。

Amazon ECS アプリケーションをデプロイするには

1. デプロイグループのコンソールページで、[デプロイの作成] を選択します。
2. [デプロイグループ] で、[ecs-demo-dg] を選択します。
3. [Revision type (リビジョンのタイプ)] の横の [My application is stored in Amazon S3 (Amazon S3 に保存されているアプリケーション)] を選択します。[リビジョンの場所] に、S3 バケットの名前を入力します。
4. [リビジョンファイルタイプ] で、必要に応じて[.json] または [.yaml] を選択します。
5. (オプション)[デプロイの説明] に、デプロイの説明を入力します。
6. [デプロイの作成] を選択します。
7. [Deployment status (デプロイのステータス)] で、デプロイをモニタリングできます。本稼働トラフィックの 100% が置き換えタスクセットにルーティングされた後、5 分の待機時間が期限切れになる前に、[Terminate original task set (元のタスクセットの終了)] を選択して元のタスクセットをすぐに終了できます。[Terminate original task set (元のタスクセットの終了)] を選択しない場合、指定した 5 分の待機時間が経過すると元のタスクセットが終了します。



ステップ 4: クリーンアップする

次のチュートリアル、チュートリアル: 検証テストを使用して Amazon ECS サービスをデプロイする で、このチュートリアルを基盤として構築し、作成した CodeDeploy アプリケーションとデプロイグループを使用します。そのチュートリアルのステップを実行する場合は、このステップをスキップし、作成したリソースを削除しないでください。

> **Note**
>
> AWS アカウントでは、作成した CodeDeploy リソースに対して料金は発生しません。

これらのステップのリソース名は、このチュートリアルで提案されている名前です (例えば、CodeDeploy アプリケーションの名前のための **ecs-demo-codedeploy-app** など) 。別の名前を使用した場合は、クリーンアップ時にそれらを使用してください。

1. delete-deployment-group コマンドを使用して、CodeDeploy デプロイグループを削除します。

```
aws deploy delete-deployment-group --application-name ecs-demo-codedeploy-app --
deployment-group-name ecs-demo-dg --region aws-region-id
```

2. delete-application コマンドを使用して、 CodeDeploy アプリケーションを削除します。

```
aws deploy delete-application --application-name ecs-demo-codedeploy-app --
region aws-region-id
```

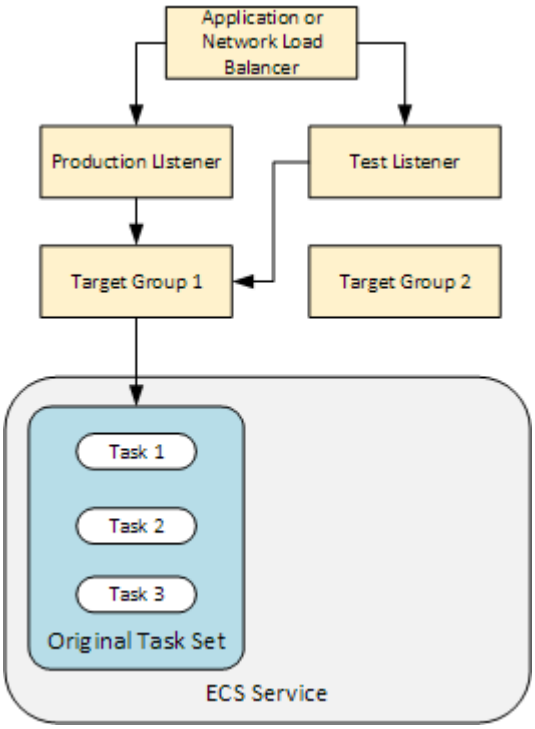
チュートリアル: 検証テストを使用して Amazon ECS サービスをデプロイする

このチュートリアルでは、Lambda 関数を使用して、更新された Amazon ECS アプリケーションのデプロイの一部を検証する方法について学びます。このチュートリアルでは、CodeDeploy アプリケーション、CodeDeploy デプロイグループ、および チュートリアル: Amazon ECS へアプリケーションをデプロイする で使用した Amazon ECS アプリケーションを使用します。そのチュートリアルを完了してからこのチュートリアルを開始してください。

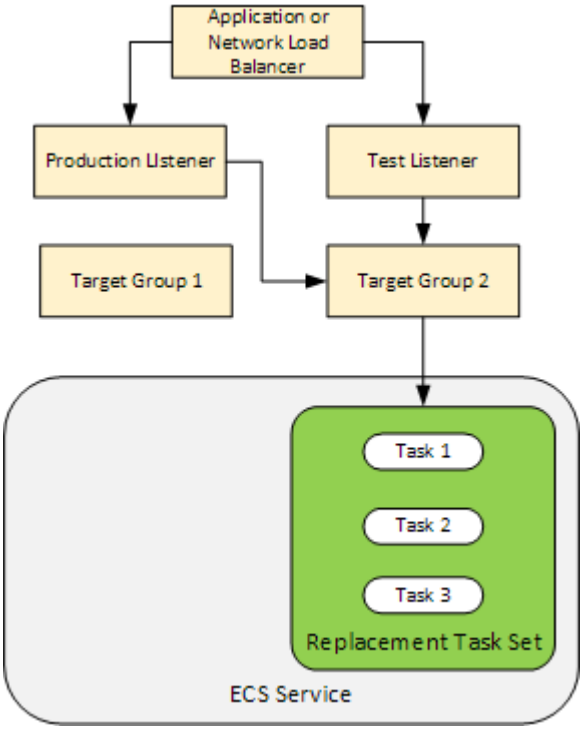
検証テストを追加するには、最初に Lambda 関数でテストを実装します。次に、デプロイ AppSpec ファイルで、テストしたい ライフサイクルフックのために Lambda 関数を指定します。検証テスト

が失敗した場合、デプロイは停止し、ロールバックされて、失敗とマークされます。テストが成功すると、デプロイは次のデプロイライフサイクルイベントまたはフックに進みます。

検証テストを使用した Amazon ECS デプロイ中、CodeDeploy は 2 つのターゲットグループ (1 つの本稼働トラフィックリスナーと 1 つのテストトラフィックリスナー) で構成されたロードバランサーを使用します。次の図表は、デプロイの前に、ロードバランサー、本稼働およびテストリスナー、ターゲットグループ、および Amazon ECS アプリケーションがどのように関連しているかを示しています。このチュートリアルでは、Application Load Balancer を使用します。Network Load Balancer を使用することもできます。



Amazon ECS デプロイ中、テスト用の 5 つのライフサイクルフックがあります。このチュートリアルでは、3 番目のライフサイクルデプロイフック、`AfterAllowTestTraffic` 中に 1 つのテストを実装します。詳細については、「Amazon ECS のデプロイ向けのライフサイクルイベントフックのリスト」を参照してください。デプロイが成功すると、本稼働トラフィックリスナーは新しい置き換えタスクセットにトラフィックを提供し、元のタスクセットは終了します。次の図は、デプロイが成功した後にリソースがどのように関連しているかを示しています。詳細については、「Amazon ECS デプロイ中の処理で起こっていること」を参照してください。



> **Note**
>
> このチュートリアルを完了すると、 AWS アカウントに料金が発生する可能性があります。これには、CodeDeploy、 AWS Lambda、CloudWatch に対して発生する可能性のある料金が含まれます。さらなる詳細については、AWS CodeDeploy の料金、AWS Lambda の料金、および Amazon CloudWatch の料金 を参照してください。

トピック

- 前提条件
- ステップ 1: テストリスナーを作成する
- ステップ 2: Amazon ECS アプリケーションを更新する
- ステップ 3: ライフサイクルフック Lambda 関数を作成する
- ステップ 4: AppSpec ファイルを更新する
- ステップ 5: CodeDeploy コンソールを使用して Amazon ECS サービスをデプロイする
- ステップ 6: CloudWatch Logs で Lambda フック関数の出力を表示する
- ステップ 7: クリーンアップする

前提条件

このチュートリアルを正常に完了するには、まず以下を行う必要があります。

- 前提条件 にある チュートリアル: Amazon ECS へアプリケーションをデプロイする の前提条件を完了します。
- 「チュートリアル: Amazon ECS へアプリケーションをデプロイする」のステップを完了します。以下を書き留めます。
  - ロードバランサーの名前。
  - ターゲットグループの名前。
  - ロードバランサーのリスナーが使用するポート。
  - ロードバランサーの ARN。これを使用して新しいリスナーを作成します。
  - いずれかのターゲットグループの ARN。これを使用して新しいリスナーを作成します。
  - 作成する CodeDeploy アプリケーションとデプロイグループ。
  - CodeDeploy デプロイで使用されることによって作成する AppSpec ファイル このチュートリアルでは、このファイルを編集します。

ステップ 1: テストリスナーを作成する

検証テストを使用する Amazon ECS デプロイには、2 番目のリスナーが必要です。このリスナーは、置き換えタスクセット中の更新された Amazon ECS アプリケーションにテストトラフィックを提供するために使用されます。検証テストは、テストトラフィックに対して実行されます。

テストトラフィックのリスナーは、いずれのターゲットグループも使用できます。create-listener AWS CLI コマンドを使用して、テストトラフィックをポート 8080 に転送するデフォルトルールを持つ 2 番目のリスナーを作成します。ロードバランサーの ARN といずれかのターゲットグループの ARN を使用します。

```
aws elbv2 create-listener --load-balancer-arn your-load-balancer-arn \
--protocol HTTP --port 8080 \
--default-actions Type=forward,TargetGroupArn=your-target-group-arn --region your-aws-region
```

ステップ 2: Amazon ECS アプリケーションを更新する

このセクションでは、タスク定義の新しいリビジョンを使用するために Amazon ECS アプリケーションを更新します。新しいリビジョンを作成し、タグを追加することでマイナー更新を追加します。

タスク定義を更新するには

1. Amazon ECS クラシックコンソール (https://console.aws.amazon.com/ecs/) を開きます。
2. ナビゲーションペインで、[タスク定義] を選択します。
3. Amazon ECS サービスで使用されるタスク定義のチェックボックスを選択します。
4. [Create new revision (新しいリビジョンを作成)] を選択します。
5. タグを追加して、タスク定義を少し更新します。ページの下部にある [タグ] で、新しいキーと値のペアを入力して新しいタグを作成します。
6. [作成] を選択します。タスク定義のリビジョン番号が 1 増えたことがわかります。
7. JSON タブを選択します。[taskDefinitionArn] の値を書き留めておきます。形式は `arn:aws:ecs:`*`aws-region`*`:`*`account-id`*`:task-definition/`*`task-definition-family`*`:`*`task-definition-revision`* です。これは、更新されたタスク定義の ARN です。

ステップ 3: ライフサイクルフック Lambda 関数を作成する

このセクションでは、Amazon ECS デプロイの `AfterAllowTestTraffic` のフック用に 1 つの Lambda 関数を実装します。Lambda 関数は、更新された Amazon ECS アプリケーションがインストールされる前に検証テストを実行します。このチュートリアルでは、Lambda 関数は `Succeeded` を返します。実際のデプロイ中、検証テストの結果に応じて、検証テストは `Succeeded` または `Failed` を返します。また、実際のデプロイ中に、他の Amazon ECS デプロイライフサイクルイベントフック (`BeforeInstall`、`AfterInstall`、`BeforeAllowTraffic`、および `AfterAllowTraffic`) に 1 つ以上の Lambda テスト関数を実装することもできます。詳細については、「Amazon ECS のデプロイ向けのライフサイクルイベントフックのリスト」を参照してください。

Lambda 関数を作成するには、IAM ロールが必要です。ロールは、CloudWatch Logs に書き込む Lambda 関数を許可し、CodeDeploy ライフサイクルフックのステータスを設定します。

IAM ロールを作成するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで [ロール] を選択し、[ロールを作成] を選択します。
3. 次のプロパティでロールを作成します。

   - 信頼されたエンティティ]: AWS Lambda。
   - [アクセス許可]: [AWSLambdaBasicExecutionRole]。これにより、CloudWatch Logs に書き込むアクセス許可が Lambda 関数に付与されます。
   - ロール名]: **lambda-cli-hook-role**。

   詳細については、AWS Lambda 「実行ロールの作成」を参照してください。
4. 作成したロールにアクセス許可 `codedeploy:PutLifecycleEventHookExecutionStatus` をアタッチします。これにより、デプロイ中に CodeDeploy ライフサイクルフックのステータスを設定する Lambda 関数の許可が付与されます。詳細については、AWS Identity and Access Management ユーザーガイド の Adding IAM identity permissions、および CodeDeploy API Reference の PutLifecycleEventHookExecutionStatus を参照してください。

`AfterAllowTestTraffic` のフック Lambda 関数を作成するには

1. 次の内容で、`AfterAllowTestTraffic.js` という名前のファイルを作成します。

```
'use strict';

 const AWS = require('aws-sdk');
 const codedeploy = new AWS.CodeDeploy({apiVersion: '2014-10-06'});

 exports.handler = (event, context, callback) => {

  console.log("Entering AfterAllowTestTraffic hook.");

  // Read the DeploymentId and LifecycleEventHookExecutionId from the event payload
  var deploymentId = event.DeploymentId;
  var lifecycleEventHookExecutionId = event.LifecycleEventHookExecutionId;
  var validationTestResult = "Failed";

  // Perform AfterAllowTestTraffic validation tests here. Set the test result
  // to "Succeeded" for this tutorial.
  console.log("This is where AfterAllowTestTraffic validation tests happen.")
```

```
 validationTestResult = "Succeeded";

 // Complete the AfterAllowTestTraffic hook by sending CodeDeploy the validation
status
 var params = {
  deploymentId: deploymentId,
  lifecycleEventHookExecutionId: lifecycleEventHookExecutionId,
  status: validationTestResult // status can be 'Succeeded' or 'Failed'
 };

 // Pass CodeDeploy the prepared validation test results.
 codedeploy.putLifecycleEventHookExecutionStatus(params, function(err, data) {
  if (err) {
   // Validation failed.
   console.log('AfterAllowTestTraffic validation tests failed');
   console.log(err, err.stack);
   callback("CodeDeploy Status update failed");
  } else {
   // Validation succeeded.
   console.log("AfterAllowTestTraffic validation tests succeeded");
   callback(null, "AfterAllowTestTraffic validation tests succeeded");
  }
 });
}
```

2. Lambda デプロイパッケージを作成する

```
zip AfterAllowTestTraffic.zip AfterAllowTestTraffic.js
```

3. `create-function` のコマンドを使用して、`AfterAllowTestTraffic` のフックのために Lambda 関数を作成します。

```
aws lambda create-function --function-name AfterAllowTestTraffic \
       --zip-file fileb://AfterAllowTestTraffic.zip \
       --handler AfterAllowTestTraffic.handler \
       --runtime nodejs10.x \
       --role arn:aws:iam::aws-account-id:role/lambda-cli-hook-role
```

4. `create-function` のレスポンスにある Lambda 関数の ARN を書き留めます。この ARN は、次のステップで CodeDeploy デプロイの AppSpec ファイルを更新するときに使用します。

ステップ 4: AppSpec ファイルを更新する

このセクションでは、Hooks のセクションを使用して AppSpec ファイルを更新します。Hooks のセクションで、AfterAllowTestTraffic のライフサイクルフックのための Lambda 関数を指定します。

AppSpec ファイルを更新するには

1. ステップ 2: AppSpec ファイルを作成します。 の チュートリアル: Amazon ECS へアプリケーションをデプロイする で作成した AppSpec ファイルファイルを開きます。
2. TaskDefinition プロパティを、「ステップ 2: Amazon ECS アプリケーションを更新する」でメモしたタスク定義 ARN で更新します。
3. コピーアンドペーストして、Hooks のセクションを AppSpec ファイルファイルに追加します。ARN を AfterAllowTestTraffic でメモした Lambda 関数の ARN を用いて ステップ 3: ライフサイクルフック Lambda 関数を作成する の後の ARN を更新します。

   JSON AppSpec

```
{
  "version": 0.0,
  "Resources": [
    {
      "TargetService": {
        "Type": "AWS::ECS::Service",
        "Properties": {
          "TaskDefinition": "arn:aws:ecs:aws-region-id:aws-account-id::task-
definition/ecs-demo-task-definition:revision-number",
          "LoadBalancerInfo": {
            "ContainerName": "sample-website",
            "ContainerPort": 80
          }
        }
      }
    }
  ],
  "Hooks": [
    {
      "AfterAllowTestTraffic": "arn:aws:lambda:aws-region-id:aws-account-
id:function:AfterAllowTestTraffic"
    }
  ]
```

```
}
```

YAML AppSpec

```
version: 0.0
Resources:
  - TargetService:
      Type: AWS::ECS::Service
      Properties:
        TaskDefinition: "arn:aws:ecs:aws-region-id:aws-account-id::task-
definition/ecs-demo-task-definition:revision-number"
        LoadBalancerInfo:
          ContainerName: "sample-website"
          ContainerPort: 80
Hooks:
  - AfterAllowTestTraffic: "arn:aws:lambda:aws-region-id:aws-account-
id:function:AfterAllowTestTraffic"
```

4. AppSpec ファイルを保存し、S3 バケットにアップロードします。

ステップ 5: CodeDeploy コンソールを使用して Amazon ECS サービスをデプロイする

このセクションでは、テストリスナーのポートを指定して、デプロイグループを更新します。これは、「ステップ 1: テストリスナーを作成する」で作成したリスナーです。デプロイ中、CodeDeploy は、テストリスナーを使用して置き換えタスクセットに提供されるテストトラフィックを使用して、AfterAllowTestTraffic のデプロイライフサイクルフックの間に、検証テストを実行します。検証テストによって結果 Succeeded が返されるため、デプロイは次のデプロイライフサイクルイベントに進みます。実際のシナリオでは、テスト関数は Succeeded または Failed を返します。

デプロイグループにテストリスナーを追加するには

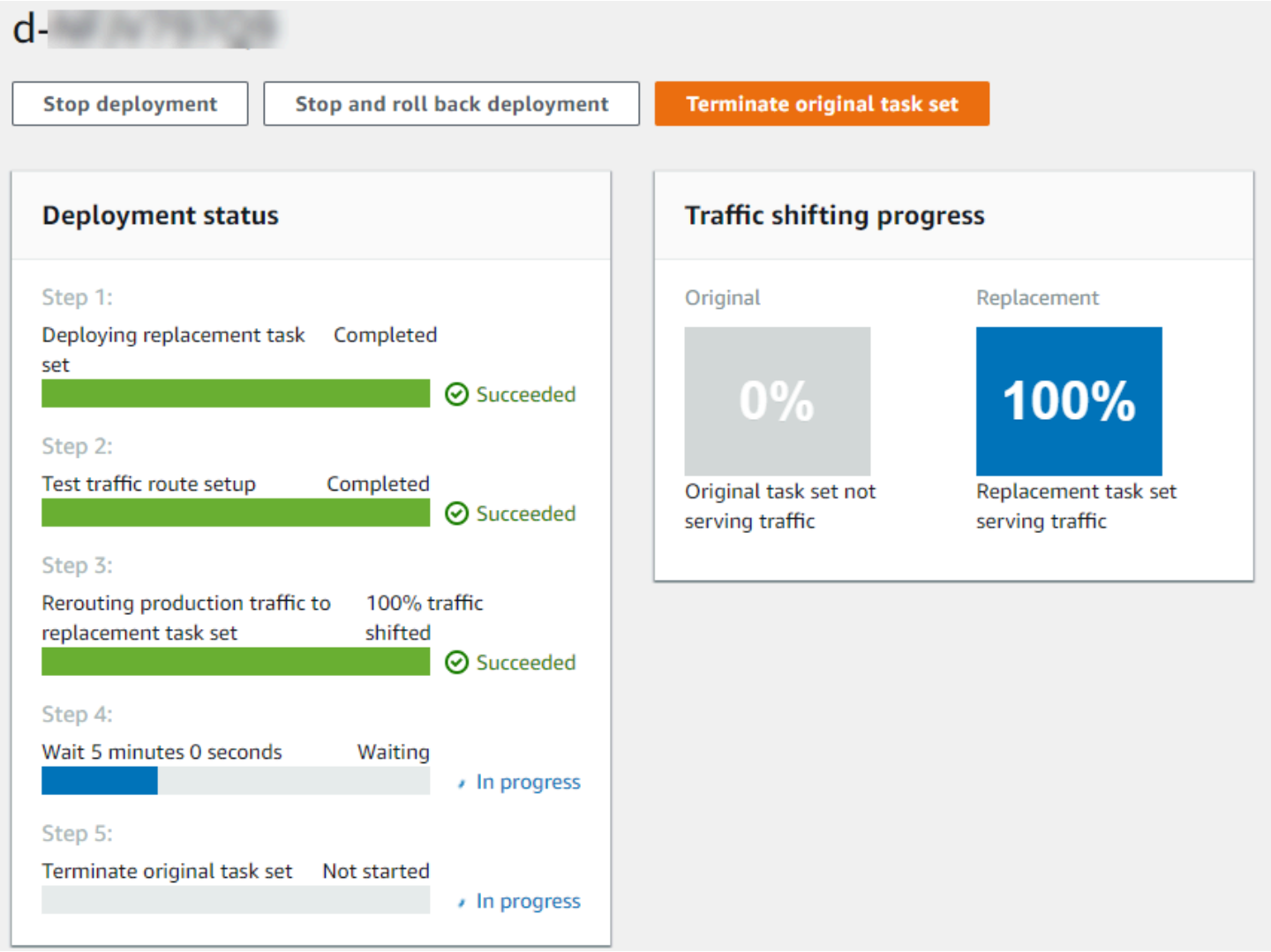
1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy/ で CodeDeploy コンソールを開きます。
2. ナビゲーションペインで、[アプリケーション] を選択します。
3. 「チュートリアル: Amazon ECS へアプリケーションをデプロイする」で作成したアプリケーションを選択します。提案された名前を使用した場合は、[ecs-demo-codedeploy-app] です。

4. [デプロイグループ] で、先ほど チュートリアル: Amazon ECS へアプリケーションをデプロイする で作成したデプロイグループを選択します。推奨された名前を使用した場合は [ecs-demo-dg] です。
5. [編集] を選択します。
6. [Test listener port (テストリスナーポート)] から、このチュートリアルで前に作成したテストリスナーのポートとプロトコルを選択します。これは [HTTP:8080] である必要があります。
7. [Save changes] (変更の保存) をクリックします。

Amazon ECS アプリケーションをデプロイするには

1. デプロイグループのコンソールページで、[デプロイの作成] を選択します。
2. [デプロイグループ] で、[ecs-demo-dg] を選択します。
3. [Revision type (リビジョンのタイプ)] の場合は、[My application is stored in Amazon S3 (Amazon S3 に保存されているアプリケーション)] を選択します。リビジョンの場所 に、S3 バケットの名前と AppSpec ファイル (例: **s3://my-s3-bucket/appspec.json**) を入力します。
4. [リビジョンファイルの種類] で、必要に応じて [.json] または [.yaml] を選択します。
5. (オプション)[デプロイの説明] に、デプロイの説明を入力します。
6. [デプロイの作成] を選択します。

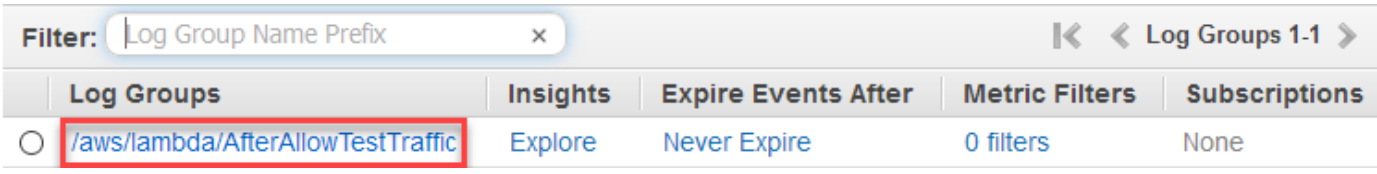
[Deployment status (デプロイのステータス)] で、デプロイをモニタリングできます。本稼働トラフィックの 100% が置き換えタスクセットにルーティングされた後、[Terminate original task set (元のタスクセットの終了)] を選択して元のタスクセットをすぐに終了できます。[Terminate original task set (元のタスクセットの終了)] を選択しない場合、元のタスクセットはデプロイグループの作成時に指定した期間後に終了します。



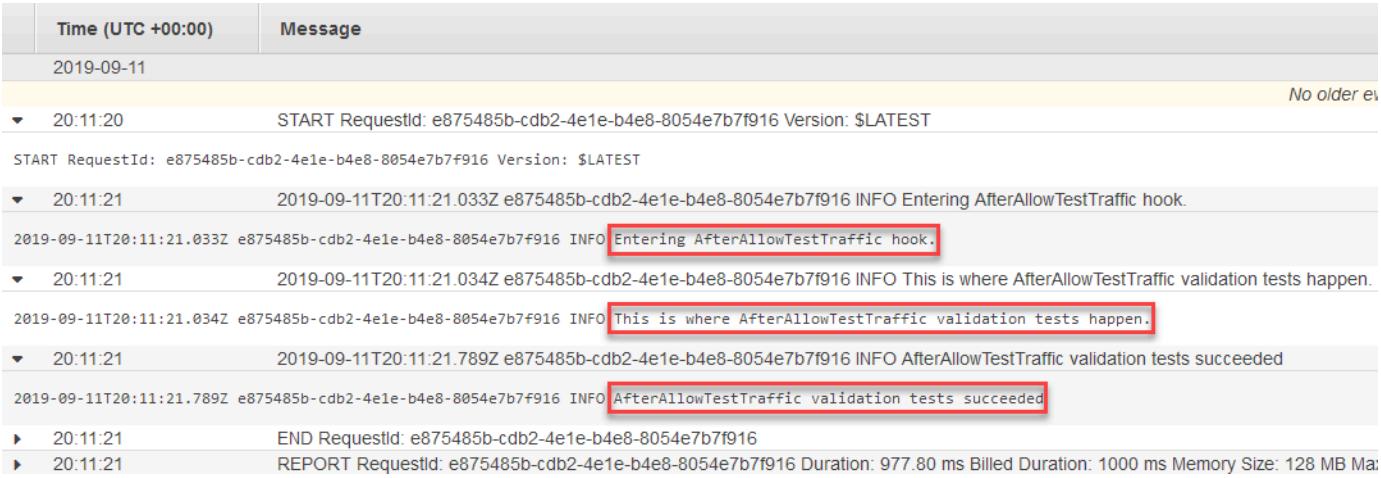
ステップ 6: CloudWatch Logs で Lambda フック関数の出力を表示する

CodeDeploy デプロイが成功すると、Lambda フック関数での検証テストも成功します。これを確認するには、CloudWatch Logs にあるフック関数のログを確認します。

1. CloudWatch コンソールの https://console.aws.amazon.com/cloudwatch/ を開いてください。
2. ナビゲーションペインで、[Logs (ログ)] を選択します。AppSpec ファイル で指定した Lambda フック関数の新しいロググループが 1 つ表示されます。



3. 新しいロググループを選択します。これは /aws/lambda/AfterAllowTestTrafficHook である必要があります。
4. ログストリームを選択します。複数のログストリームが表示された場合は、[最後のイベント時間] で最新の日付と時刻のログストリームを選択します。
5. ログストリームイベントを展開し、Lambda フック関数がログに成功メッセージを書き込んだことを確認します。以下は、 `AfterAllowTraffic` の Lambda フック関数が成功したことを示します。



ステップ 7: クリーンアップする

このチュートリアルが終了したら、使用していないリソースに対する料金が発生しないように、チュートリアルに関連付けられたリソースをクリーンアップします。このステップのリソース名は、このチュートリアルで提案されている名前です (例えば、CodeDeploy アプリケーションの名前のための **ecs-demo-codedeploy-app** など)。別の名前を使用した場合は、クリーンアップでそれらを使用してください。

チュートリアルリソースをクリーンアップするには

1. delete-deployment-group コマンドを使用して、CodeDeploy デプロイグループを削除します。

```
aws deploy delete-deployment-group --application-name ecs-demo-deployment-group --
deployment-group-name ecs-demo-dg --region aws-region-id
```

2. delete-application コマンドを使用して、 CodeDeploy アプリケーションを削除します。

```
aws deploy delete-application --application-name ecs-demo-deployment-group --
region aws-region-id
```

3. delete-function コマンドを使用して、Lambda フック関数を削除します。

```
aws lambda delete-function --function-name AfterAllowTestTraffic
```

4. delete-log-group コマンドを使用して、CloudWatch log グループを削除します。

```
aws logs delete-log-group --log-group-name /aws/lambda/AfterAllowTestTraffic
```

チュートリアル: CodeDeploy と AWS サーバーレスアプリケーションモデルを使用して更新された Lambda 関数をデプロイする

AWS SAM は、サーバーレスアプリケーションを構築するためのオープンソースフレームワークです。 AWS SAM テンプレート内の YAML 構文を CloudFormation 構文に変換して拡張し、Lambda 関数などのサーバーレスアプリケーションを構築します。詳細については、「 AWS サーバーレスアプリケーションモデルとは何ですか。」 を参照してください。

このチュートリアルでは、SAM AWS を使用して、以下を実行するソリューションを作成します。

- Lambda 関数を作成します。
- CodeDeploy アプリケーションとデプロイグループを作成します。
- CodeDeploy ライフサイクルフック中にデプロイの検証テストを実行する 2 つの Lambda 関数を作成します。
- Lambda 関数がいつ更新されたかを検出します。Lambda 関数の更新により、CodeDeploy によるデプロイがトリガーされます。これにより、本稼働トラフィックが Lambda 関数の元のバージョンから更新されたバージョンに段階的に移行されます。

> **Note**
>
> このチュートリアルでは、 AWS アカウントに課金される可能性のあるリソースを作成する必要があります。これには、CodeDeploy、Amazon CloudWatch、および に対して発生する可能性のある料金が含まれます AWS Lambda。詳細については、CodeDeploy pricing、Amazon CloudWatch pricing および AWS Lambda pricing を参照してください。

トピック

- 前提条件

- ステップ 1: インフラストラクチャをセットアップする
- ステップ 2: Lambda 関数を更新します
- ステップ 3: 更新された Lambda 関数をデプロイします。
- ステップ 4: デプロイの結果を表示する
- ステップ 5: クリーンアップ

前提条件

このチュートリアルを完了するには、まず以下を行う必要があります。

- 「CodeDeploy の開始方法」のステップを完了します。
- CLI AWS Serverless Application Model をインストールします。詳細については、「SAM AWS CLI のインストール」を参照してください。
- S3 バケットを作成します。 AWS SAM は、AWS SAM テンプレートで参照されているアーティファクトをこのバケットにアップロードします。

ステップ 1: インフラストラクチャをセットアップする

このトピックでは、 AWS SAM を使用して AWS SAM テンプレートと Lambda 関数のファイルを作成する方法について説明します。次に、 コマンド AWS SAM `package`と `deploy` コマンドを使用して、インフラストラクチャ内のコンポーネントを生成します。インフラストラクチャの準備ができたら、CodeDeploy アプリケーションとデプロイグループ、更新およびデプロイする Lambda 関数、Lambda 関数をデプロイするときに実行される検証テストを含む 2 つの Lambda 関数があります。完了したら、 CloudFormation を使用して Lambda コンソールまたは でコンポーネントを表示 AWS CLI し、Lambda 関数をテストできます。

トピック

- ファイルを作成する
- SAM AWS アプリケーションをパッケージ化する
- SAM AWS アプリケーションをデプロイする
- (オプション) インフラストラクチャの検査とテスト

ファイルを作成する

インフラストラクチャを作成するには、次のファイルを作成する必要があります。

- template.yml
- myDateTimeFunction.js
- beforeAllowTraffic.js
- afterAllowTraffic.js

トピック

- AWS SAM テンプレートを作成する
- Lambda 関数のファイルを作成します
- [BeforeAllowTraffic] Lambda 関数のファイルを作成する
- [AfterAllowTraffic] Lambda 関数のファイルを作成する

AWS SAM テンプレートを作成する

インフラストラクチャ内のコンポーネントを指定する AWS SAM テンプレートファイルを作成します。

AWS SAM テンプレートを作成するには

1. SAM-Tutorial という名前のディレクトリを作成します。
2. SAM-Tutorial ディレクトリに template.yml という名前のファイルを作成します。
3. 次の YAML コードを template.yml にコピーします。これが使用する AWS SAM テンプレートです。

```
AWSTemplateFormatVersion : '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: A sample SAM template for deploying Lambda functions.

Resources:
# Details about the myDateTimeFunction Lambda function
  myDateTimeFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: myDateTimeFunction.handler
      Runtime: nodejs18.x
# Instructs your myDateTimeFunction is published to an alias named "live".
      AutoPublishAlias: live
# Grants this function permission to call lambda:InvokeFunction
      Policies:
```

```
        - Version: "2012-10-17"
          Statement:
          - Effect: "Allow"
            Action:
              - "lambda:InvokeFunction"
            Resource: '*'
      DeploymentPreference:
# Specifies the deployment configuration
          Type: Linear10PercentEvery1Minute
# Specifies Lambda functions for deployment lifecycle hooks
          Hooks:
            PreTraffic: !Ref beforeAllowTraffic
            PostTraffic: !Ref afterAllowTraffic

# Specifies the BeforeAllowTraffic lifecycle hook Lambda function
  beforeAllowTraffic:
    Type: AWS::Serverless::Function
    Properties:
      Handler: beforeAllowTraffic.handler
      Policies:
        - Version: "2012-10-17"
# Grants this function permission to call
 codedeploy:PutLifecycleEventHookExecutionStatus
          Statement:
          - Effect: "Allow"
            Action:
              - "codedeploy:PutLifecycleEventHookExecutionStatus"
            Resource:
              !Sub 'arn:aws:codedeploy:${AWS::Region}:
${AWS::AccountId}:deploymentgroup:${ServerlessDeploymentApplication}/*'
        - Version: "2012-10-17"
# Grants this function permission to call lambda:InvokeFunction
          Statement:
          - Effect: "Allow"
            Action:
              - "lambda:InvokeFunction"
            Resource: !Ref myDateTimeFunction.Version
      Runtime: nodejs18.x
# Specifies the name of the Lambda hook function
      FunctionName: 'CodeDeployHook_beforeAllowTraffic'
      DeploymentPreference:
        Enabled: false
      Timeout: 5
      Environment:
```

```
        Variables:
          NewVersion: !Ref myDateTimeFunction.Version

# Specifies the AfterAllowTraffic lifecycle hook Lambda function
  afterAllowTraffic:
    Type: AWS::Serverless::Function
    Properties:
      Handler: afterAllowTraffic.handler
      Policies:
        - Version: "2012-10-17"
          Statement:
# Grants this function permission to call
 codedeploy:PutLifecycleEventHookExecutionStatus
          - Effect: "Allow"
            Action:
              - "codedeploy:PutLifecycleEventHookExecutionStatus"
            Resource:
              !Sub 'arn:aws:codedeploy:${AWS::Region}:
${AWS::AccountId}:deploymentgroup:${ServerlessDeploymentApplication}/*'
        - Version: "2012-10-17"
          Statement:
# Grants this function permission to call lambda:InvokeFunction
          - Effect: "Allow"
            Action:
              - "lambda:InvokeFunction"
            Resource: !Ref myDateTimeFunction.Version
      Runtime: nodejs18.x
# Specifies the name of the Lambda hook function
      FunctionName: 'CodeDeployHook_afterAllowTraffic'
      DeploymentPreference:
        Enabled: false
      Timeout: 5
      Environment:
        Variables:
          NewVersion: !Ref myDateTimeFunction.Version
```

このテンプレートは以下を指定します。詳細については、AWS SAM template concepts (テンプレートの概念) を参照してください。

myDateTimeFunction と呼ばれるLambda 関数

この Lambda 関数が発行されると、テンプレート中の AutoPublishAlias の行は、それを live という名前のエイリアスにリンクします。このチュートリアルの後半で、この関数の更新により、本番トラフィックを元のバージョンから更新されたバージョンに段階的に移行 AWS CodeDeploy する によるデプロイがトリガーされます。

2 つの Lambda デプロイ検証関数

次の Lambda 関数は、CodeDeploy ライフサイクルフック中に実行されます。関数には、更新された myDateTimeFunction のデプロイを検証するコードが含まれています。検証テストの結果は、PutLifecycleEventHookExecutionStatus のAPI メソッドを使用して CodeDeploy に渡されます。検証テストが失敗すると、デプロイは失敗し、ロールバックされます。

- CodeDeployHook_beforeAllowTraffic は、BeforeAllowTraffic フック中に実行します。
- CodeDeployHook_afterAllowTraffic は、AfterAllowTraffic フック中に実行します。

両方の関数の名前は CodeDeployHook_ で始まります。CodeDeployRoleForLambda のロールは、このプレフィックスで始まる名前の Lambda 関数でのみ、Lambda の invoke のメソッドへの呼び出しを許可します。詳細については、AWS Lambda デプロイの AppSpec 'hooks' セクション および CodeDeploy API Reference 中の PutLifecycleEventHookExecutionStatus を参照してください。

更新された Lambda 関数の自動検出

AutoPublishAlias 条件は、myDateTimeFunction 関数が変更されたときに検出し、live エイリアスを使用してデプロイするようにフレームワークに指示します。

デプロイ設定

デプロイ設定により、CodeDeploy アプリケーションが Lambda 関数の元のバージョンから新しいバージョンにトラフィックを移行するレートが決まります。このテンプレートは、事前定義されたデプロイ設定 Linear10PercentEvery1Minute を指定します。

> **Note**
>
> AWS SAM テンプレートでカスタムデプロイ設定を指定することはできません。詳細については、「Create a Deployment Configuration」を参照してください。

デプロイライフサイクルフック関数

Hooks セクションでは、ライフサイクルイベントフック中に実行する関数を指定します。PreTraffic は、PostTraffic フック中に実行する関数を指定します。BeforeAllowTraffic は、フック中に AfterAllowTraffic が実行する関数を指定します。

Lambda が別の Lambda 関数を呼び出すための許可

指定されたlambda:InvokeFunctionアクセス許可は、SAM AWS アプリケーションで使用されるロールに Lambda 関数を呼び出すアクセス許可を付与します。これは、CodeDeployHook_beforeAllowTraffic および CodeDeployHook_afterAllowTraffic の関数が、検証テスト中に、デプロイされた Lambda 関数を呼び出す場合に必要です。

Lambda 関数のファイルを作成します

このチュートリアルの後半で更新してデプロイする関数のファイルを作成します。

> **Note**
>
> Lambda 関数は、 AWS Lambdaでサポートされている任意のランタイムを使用できます。詳細については、「AWS Lambda ランタイム」を参照してください。

Lambda 関数を作成するには

1. テキストファイルを作成し、myDateTimeFunction.js という名前で SAM-Tutorial ディレクトリに保存します。
2. 次の Node.js コードを myDateTimeFunction.js にコピーします。

```
'use strict';

    exports.handler = function(event, context, callback) {

      if (event.body) {
        event = JSON.parse(event.body);
      }

      var sc; // Status code
```

```
    var result = ""; // Response payload

    switch(event.option) {
      case "date":
        switch(event.period) {
          case "yesterday":
            result = setDateResult("yesterday");
            sc = 200;
            break;
          case "today":
            result = setDateResult();
            sc = 200;
            break;
          case "tomorrow":
            result = setDateResult("tomorrow");
            sc = 200;
            break;
          default:
            result = {
              "error": "Must specify 'yesterday', 'today', or 'tomorrow'."
            };
            sc = 400;
            break;
        }
        break;

  /*      Later in this tutorial, you update this function by uncommenting
          this section. The framework created by AWS SAM detects the update
          and triggers a deployment by CodeDeploy. The deployment shifts
          production traffic to the updated version of this function.

          case "time":
          var d = new Date();
          var h = d.getHours();
          var mi = d.getMinutes();
          var s = d.getSeconds();

          result = {
            "hour": h,
            "minute": mi,
            "second": s
          };
          sc = 200;
          break;
```

```
    */
          default:
            result = {
              "error": "Must specify 'date' or 'time'."
            };
            sc = 400;
          break;
      }

      const response = {
        statusCode: sc,
        headers: { "Content-type": "application/json" },
        body: JSON.stringify( result )
      };

      callback(null, response);

      function setDateResult(option) {

        var d = new Date(); // Today
        var mo; // Month
        var da; // Day
        var y; // Year

        switch(option) {
          case "yesterday":
            d.setDate(d.getDate() - 1);
            break;
          case "tomorrow":
            d.setDate(d.getDate() + 1);
          default:
           break;
        }

        mo = d.getMonth() + 1; // Months are zero offset (0-11)
        da = d.getDate();
        y = d.getFullYear();

        result = {
          "month": mo,
          "day": da,
          "year": y
        };
```

```
        return result;
      }
    };
```

Lambda 関数は、昨日、今日、または明日に対して年月日を返します。このチュートリアルの後半では、指定した日または時間に関する情報(年月日または現在の時、分、秒など)を返す関数を更新するコードをコメント解除します。によって作成されたフレームワークは、 関数の更新バージョン AWS SAM を検出してデプロイします。

> **Note**
>
> この Lambda 関数は、 AWS Cloud9 チュートリアルでも使用されます。 はクラウドベースの統合開発環境 AWS Cloud9 です。でこの関数を作成、実行、更新、デバッグする方法については AWS Cloud9、AWS Lambda 「」のチュートリアル AWS Cloud9を参照してください。

[BeforeAllowTraffic] Lambda 関数のファイルを作成する

`beforeAllowTraffic` のフック Lambda 関数のファイルを作成します。

1. テキストファイルを作成し、`beforeAllowTraffic.js` という名前で `SAM-Tutorial` ディレクトリに保存します。
2. 次の Node.js コードを `beforeAllowTraffic.js` にコピーします。この関数は、デプロイの `BeforeAllowTraffic` フック中に実行されます。

```
'use strict';

    const AWS = require('aws-sdk');
    const codedeploy = new AWS.CodeDeploy({apiVersion: '2014-10-06'});
    var lambda = new AWS.Lambda();

    exports.handler = (event, context, callback) => {

     console.log("Entering PreTraffic Hook!");

     // Read the DeploymentId and LifecycleEventHookExecutionId from the event
 payload
      var deploymentId = event.DeploymentId;
     var lifecycleEventHookExecutionId = event.LifecycleEventHookExecutionId;
```

```
var functionToTest = process.env.NewVersion;
console.log("BeforeAllowTraffic hook tests started");
console.log("Testing new function version: " + functionToTest);

// Create parameters to pass to the updated Lambda function that
// include the newly added "time" option. If the function did not
// update, then the "time" option is invalid and function returns
// a statusCode of 400 indicating it failed.
var lambdaParams = {
 FunctionName: functionToTest,
 Payload: "{\"option\": \"time\"}",
 InvocationType: "RequestResponse"
};

var lambdaResult = "Failed";
// Invoke the updated Lambda function.
lambda.invoke(lambdaParams, function(err, data) {
 if (err){ // an error occurred
  console.log(err, err.stack);
  lambdaResult = "Failed";
 }
 else{ // successful response
  var result = JSON.parse(data.Payload);
  console.log("Result: " +  JSON.stringify(result));
     console.log("statusCode: " + result.statusCode);

     // Check if the status code returned by the updated
     // function is 400. If it is, then it failed. If
     // is not, then it succeeded.
  if (result.statusCode != "400"){
       console.log("Validation succeeded");
   lambdaResult = "Succeeded";
     }
     else {
       console.log("Validation failed");
     }

  // Complete the PreTraffic Hook by sending CodeDeploy the validation status
  var params = {
   deploymentId: deploymentId,
   lifecycleEventHookExecutionId: lifecycleEventHookExecutionId,
   status: lambdaResult // status can be 'Succeeded' or 'Failed'
  };
```

```
      // Pass CodeDeploy the prepared validation test results.
      codedeploy.putLifecycleEventHookExecutionStatus(params, function(err, data)
{
       if (err) {
        // Validation failed.
        console.log("CodeDeploy Status update failed");
        console.log(err, err.stack);
        callback("CodeDeploy Status update failed");
       } else {
        // Validation succeeded.
        console.log("CodeDeploy status updated successfully");
        callback(null, "CodeDeploy status updated successfully");
       }
      });
     }
    });
   }
```

[AfterAllowTraffic] Lambda 関数のファイルを作成する

`afterAllowTraffic` のフック Lambda 関数のファイルを作成します。

1. テキストファイルを作成し、`afterAllowTraffic.js` という名前で `SAM-Tutorial` ディレクトリに保存します。
2. 次の Node.js コードを `afterAllowTraffic.js` にコピーします。この関数は、デプロイの `AfterAllowTraffic` フック中に実行されます。

```
'use strict';

   const AWS = require('aws-sdk');
   const codedeploy = new AWS.CodeDeploy({apiVersion: '2014-10-06'});
   var lambda = new AWS.Lambda();

   exports.handler = (event, context, callback) => {

    console.log("Entering PostTraffic Hook!");

    // Read the DeploymentId and LifecycleEventHookExecutionId from the event
 payload
     var deploymentId = event.DeploymentId;
    var lifecycleEventHookExecutionId = event.LifecycleEventHookExecutionId;
```

```
var functionToTest = process.env.NewVersion;
console.log("AfterAllowTraffic hook tests started");
console.log("Testing new function version: " + functionToTest);

// Create parameters to pass to the updated Lambda function that
// include the original "date" parameter. If the function did not
// update as expected, then the "date" option might be invalid. If
// the parameter is invalid, the function returns
// a statusCode of 400 indicating it failed.
var lambdaParams = {
 FunctionName: functionToTest,
 Payload: "{\"option\": \"date\", \"period\": \"today\"}",
 InvocationType: "RequestResponse"
};

var lambdaResult = "Failed";
// Invoke the updated Lambda function.
lambda.invoke(lambdaParams, function(err, data) {
 if (err){ // an error occurred
  console.log(err, err.stack);
  lambdaResult = "Failed";
 }
 else{ // successful response
  var result = JSON.parse(data.Payload);
  console.log("Result: " +  JSON.stringify(result));
     console.log("statusCode: " + result.statusCode);

     // Check if the status code returned by the updated
     // function is 400. If it is, then it failed. If
     // is not, then it succeeded.
  if (result.statusCode != "400"){
       console.log("Validation of time parameter succeeded");
   lambdaResult = "Succeeded";
     }
     else {
       console.log("Validation failed");
     }

  // Complete the PostTraffic Hook by sending CodeDeploy the validation status
  var params = {
   deploymentId: deploymentId,
   lifecycleEventHookExecutionId: lifecycleEventHookExecutionId,
   status: lambdaResult // status can be 'Succeeded' or 'Failed'
```

```
      };

      // Pass CodeDeploy the prepared validation test results.
      codedeploy.putLifecycleEventHookExecutionStatus(params, function(err, data)
{
       if (err) {
        // Validation failed.
        console.log("CodeDeploy Status update failed");
        console.log(err, err.stack);
        callback("CodeDeploy Status update failed");
       } else {
        // Validation succeeded.
        console.log("CodeDeploy status updated successfully");
        callback(null, "CodeDeploy status updated successfully");
       }
      });
     }
    });
   }
```

SAM AWS アプリケーションをパッケージ化する

これで、SAM-Tutorial ディレクトリに次の 4 つのファイルがあるはずです。

- beforeAllowTraffic.js
- afterAllowTraffic.js
- myDateTimeFunction.js
- template.yml

これで、 AWS SAM sam package コマンドを使用して、Lambda 関数と CodeDeploy アプリケーションのアーティファクトを作成してパッケージ化する準備ができました。アーティファクトは S3 バケットにアップロードされます。コマンドの出力は、package.yml という新しいファイルです。このファイルは、次のステップで AWS SAM sam deploy コマンドによって使用されます。

> **Note**
>
> sam package のコマンドのさらなる詳細については、AWS SAM デベロッパーガイド の AWS Serverless Application Model CLI command reference を参照してください。

SAM-Tutorial ディレクトリで、以下を実行します。

```
sam package \
  --template-file template.yml \
  --output-template-file package.yml \
  --s3-bucket amzn-s3-demo-bucket
```

s3-bucket のパラメータには、このチュートリアルの前提条件として作成した Amazon S3 バケットを指定します。は、 AWS SAM sam deploy コマンドで使用される新しいファイルの名前output-template-fileを指定します。

SAM AWS アプリケーションをデプロイする

ファイルで AWS SAM sam deploy コマンドpackage.ymlを使用して、 を使用して Lambda 関数と CodeDeploy アプリケーションおよびデプロイグループを作成します CloudFormation。

> **Note**
>
> sam deploy のコマンドのさらなる詳細については、AWS Serverless Application Model デベロッパーガイド の AWS SAM CLI command reference を参照してください。

SAM-Tutorial ディレクトリで、次のコマンドを実行します。

```
sam deploy \
  --template-file package.yml \
  --stack-name my-date-time-app \
  --capabilities CAPABILITY_IAM
```

IAM ロールの作成を --capabilities CAPABILITY_IAM に許可するには、 CloudFormation のパラメータが必要です。

(オプション) インフラストラクチャの検査とテスト

このトピックでは、インフラストラクチャコンポーネントを表示し、Lambda 関数をテストする方法を示します。

sam deploy の実行後にスタックの結果を表示するには

1. https://console.aws.amazon.com/cloudformation で CloudFormation コンソールを開きます。

2. ナビゲーションペインで、[Stacks] を選択します。my-date-time-app スタックが上部に表示されます。
3. [イベント] タブを選択して、完了したイベントを確認します。スタックの作成の進行中に、イベントを表示できます。スタックの作成が完了すると、すべてのスタック作成イベントを表示できます。
4. スタックを選択した状態で、[リソース] を選択します。タイプ 列に、Lambda関数、myDateTimeFunction 、CodeDeployHook_beforeAllowTraffic および CodeDeployHook_afterAllowTraffic が表示されます。Lambda 関数の Physical ID の各列には、Lambda コンソールで関数を表示するためのリンクが含まれています。

   > **Note**
   >
   > myDateTimeFunction Lambda 関数の名前には CloudFormation スタックの名前が付加され、識別子が追加されているため、 のようになりますmy-date-time-app-myDateTimeFunction-123456ABCDEF。

5. https://console.aws.amazon.com/codedeploy/ で、CodeDeploy コンソールを開きます。
6. ナビゲーションペインで [デプロイ] を展開し、[アプリケーション] を選択します。
7. で作成された新しい CodeDeploy アプリケーションは、 で始まる名前 CloudFormation で表示されますmy-date-time-app-ServerlessDeploymentApplication。このアプリケーションを選択します。
8. my-date-time-app-myDateTimeFunctionDeploymentGroup で始まる名前のデプロイグループが表示されます。このデプロイグループを選択します。

   [デプロイ設定] に、[CodeDeployDefault.LambdaLinear10PercentEvery1Minute] が表示されます。

(オプション) 関数をテストするには (コンソール)

1. https://console.aws.amazon.com/lambda/ で AWS Lambda コンソールを開きます。
2. ナビゲーションペインで、my-date-time-app-myDateTimeFunction 関数を選択します。コンソールでは、名前に識別子が含まれているため、my-date-time-app-myDateTimeFunction-123456ABCDEF のようになります。
3. [テスト] を選択します。
4. [イベント名] にテストイベントの名前を入力します。

5. テストイベントに以下を入力し、[作成] を選択します。

```
{
  "option": "date",
  "period": "today"
}
```

6. [テスト] を選択します。テストイベントのリストには、テストイベントのみが表示されます。

   [実行結果] に [成功] と表示されます。

7. [実行結果] で、[詳細] を展開して結果を表示します。現在の年月日が表示されます。

(オプション) 関数をテストするには (AWS CLI)

1. Lambda 関数の ARN を配置します。関数を表示しているときに、Lambda コンソールの上部に表示されます。
2. 以下のコマンドを実行してください。[*your-function-arn*] を関数 ARN に置き換えます。

```
aws lambda invoke \
--function your-function-arn \
--cli-binary-format raw-in-base64-out \
--payload "{\"option\": \"date\", \"period\": \"today\"}" out.txt
```

3. `out.txt` を開き、結果に現在の年月日が含まれていることを確認します。

ステップ 2: Lambda 関数を更新します

このトピックでは、`myDateTimeFunction.js` ファイルを更新します。次のステップでは、このファイルを使用して、更新された関数をデプロイします。これは、本稼働トラフィックを Lambda 関数の現在のバージョンから更新されたバージョンに移行することによってデプロイするために、CodeDeploy をトリガーします。

Lambda 関数を更新するには

1. `myDateTimeFunction.js` を開きます。
2. 2 つのコメントマーカー(「`/*`」と「`*/`」)および `case` ブロック内の `time` という名前の `switch` の開始と終了にある説明テキストを削除します。

コメントされていないコードを使用すると、新しいパラメータ time を関数に渡すことができます。time を更新された関数に渡すと、現在の hour、minute、および second が返されます。

3. myDateTimeFunction.js を保存します。次のようになります。

```
'use strict';

exports.handler = function(event, context, callback) {

  if (event.body) {
    event = JSON.parse(event.body);
  }

  var sc; // Status code
  var result = ""; // Response payload

  switch(event.option) {
    case "date":
      switch(event.period) {
        case "yesterday":
          result = setDateResult("yesterday");
          sc = 200;
          break;
        case "today":
          result = setDateResult();
          sc = 200;
          break;
        case "tomorrow":
          result = setDateResult("tomorrow");
          sc = 200;
          break;
        default:
          result = {
            "error": "Must specify 'yesterday', 'today', or 'tomorrow'."
          };
          sc = 400;
          break;
      }
      break;
      case "time":
        var d = new Date();
        var h = d.getHours();
```

```
        var mi = d.getMinutes();
        var s = d.getSeconds();

        result = {
          "hour": h,
          "minute": mi,
          "second": s
        };
        sc = 200;
        break;

      default:
        result = {
          "error": "Must specify 'date' or 'time'."
        };
        sc = 400;
      break;
  }

  const response = {
    statusCode: sc,
    headers: { "Content-type": "application/json" },
    body: JSON.stringify( result )
  };

  callback(null, response);

  function setDateResult(option) {

    var d = new Date(); // Today
    var mo; // Month
    var da; // Day
    var y; // Year

    switch(option) {
      case "yesterday":
        d.setDate(d.getDate() - 1);
        break;
      case "tomorrow":
        d.setDate(d.getDate() + 1);
      default:
       break;
    }
```

```
    mo = d.getMonth() + 1; // Months are zero offset (0-11)
    da = d.getDate();
    y = d.getFullYear();

    result = {
      "month": mo,
      "day": da,
      "year": y
    };

    return result;
  }
};
```

ステップ 3: 更新された Lambda 関数をデプロイします。

このステップでは、更新された myDateTimeFunction.js を使用して、Lambda 関数のデプロイを更新して開始します。CodeDeploy または AWS Lambda コンソールでデプロイの進行状況をモニタリングできます。

AWS SAM テンプレートの AutoPublishAlias: live行により、インフラストラクチャはliveエイリアスを使用する関数の更新を検出します。関数を更新すると、CodeDeploy によるデプロイがトリガーされます。これにより、本稼働トラフィックが関数の元のバージョンから更新されたバージョンに移行されます。

sam package と sam deploy のコマンドは、Lambda 関数のデプロイを更新およびトリガーするために使用されます。これらのコマンドは、SAM AWS アプリケーションをパッケージ化する および SAM AWS アプリケーションをデプロイする で実行しました。

更新された Lambda 関数をデプロイするには

1. SAM-Tutorial ディレクトリで、次のコマンドを実行します。

```
sam package \
  --template-file template.yml \
  --output-template-file package.yml  \
  --s3-bucket amzn-s3-demo-bucket
```

これにより、S3 バケットで更新された Lambda 関数を参照する新しいアーティファクトのセットが作成されます。

2. SAM-Tutorial ディレクトリで、次のコマンドを実行します。

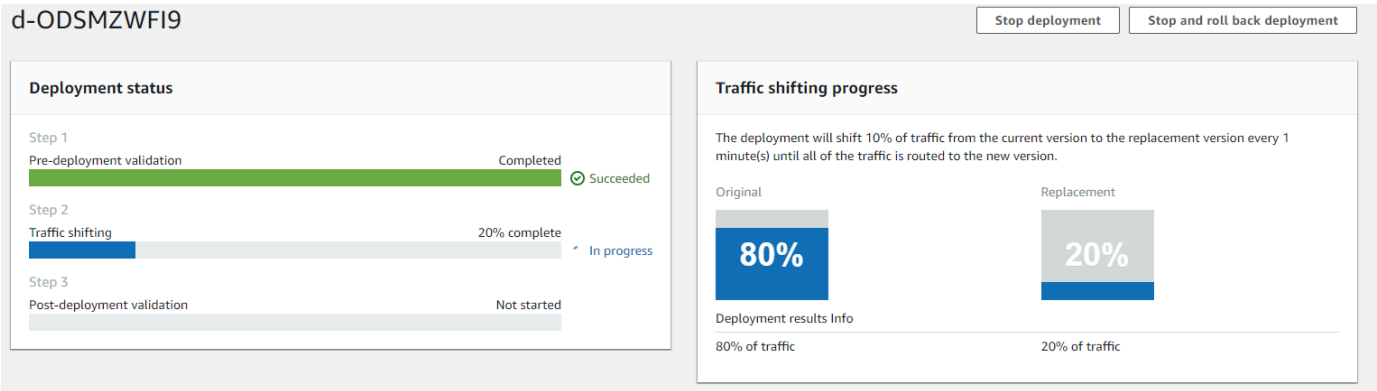
```
sam deploy \
  --template-file package.yml \
  --stack-name my-date-time-app \
  --capabilities CAPABILITY_IAM
```

スタック名はまだ であるためmy-date-time-app、 はこれがスタックの更新である CloudFormation ことを認識します。更新されたスタックを表示するには、 CloudFormation コンソールを返し、ナビゲーションペインから スタックを選択します。

(オプション)デプロイ中にトラフィックを表示するには(CodeDeploy コンソール)

1. https://console.aws.amazon.com/codedeploy/ で、CodeDeploy コンソールを開きます。
2. ナビゲーションペインで、[アプリケーション] を展開し、[my-date-time-app-ServerlessDeploymentApplication] アプリケーションを選択します。
3. [デプロイグループ] で、アプリケーションのデプロイグループを選択します。ステータスは [進行中]になります。
4. [Deployment group history (デプロイグループ履歴)] で、進行中のデプロイを選択します。

   [トラフィックの移行] の進行状況バーと、このページの [Original] ボックスと 置換] ボックスの割合に、進行状況が表示されます。

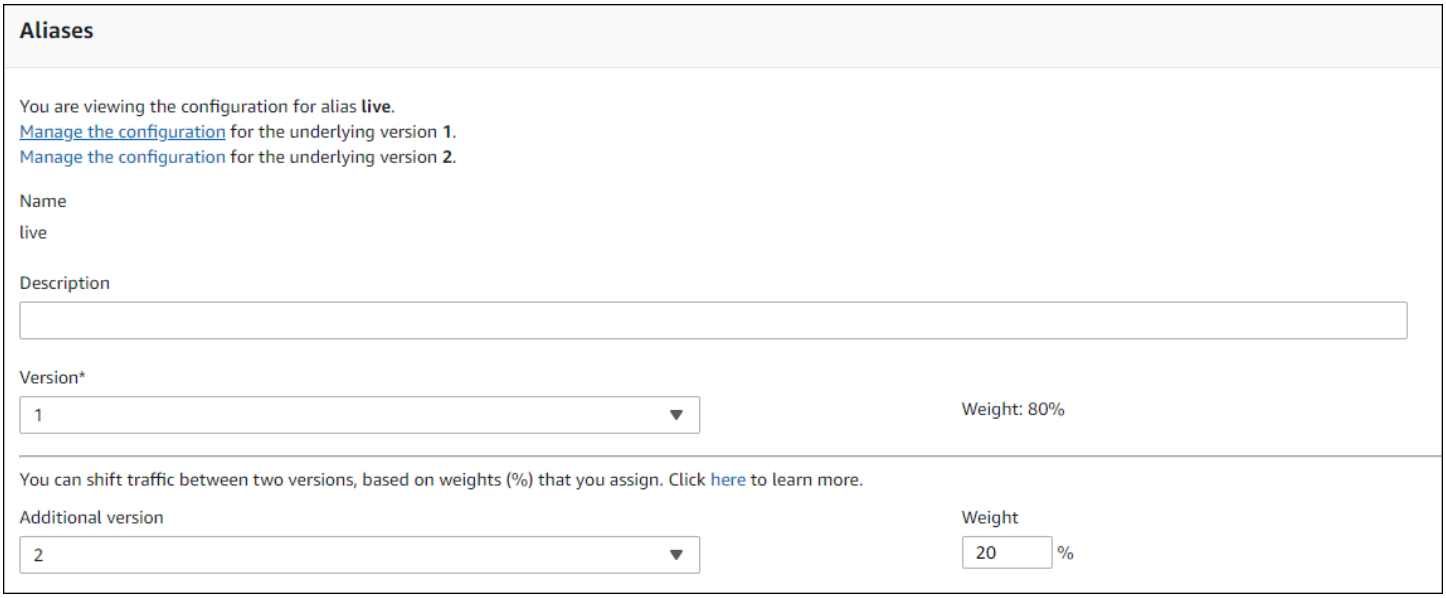


(オプション) デプロイ中にトラフィックを表示するには(Lambda コンソール)

1. https://console.aws.amazon.com/lambda/ で AWS Lambda コンソールを開きます。

2. ナビゲーションペインで、my-date-time-app-myDateTimeFunction 関数を選択します。コンソールでは、名前に識別子が含まれているため、my-date-time-app-myDateTimeFunction-123456ABCDEF のようになります。
3. Aliases、live の順に選択します。

元の関数バージョン(バージョン 1)と更新された関数バージョン(バージョン 2)の横の重みは、この AWS Lambda コンソールページがロードされた時点で各バージョンに提供されているトラフィック量を示します。ページでは、時間が経過しても重みは更新されません。ページを 1 分に 1 回更新すると、バージョン 1 の重みは 10% 減り、バージョン 2 の重みは 10% 増加します。



ステップ 4: デプロイの結果を表示する

このステップでは、デプロイの結果を表示します。デプロイが成功すると、更新された Lambda 関数が本稼働トラフィックを受信することを確認できます。デプロイが失敗した場合、CloudWatch Logs を使用して、デプロイのライフサイクルフックの間に実行される Lambda 関数の検証テストの出力を表示できます。

トピック

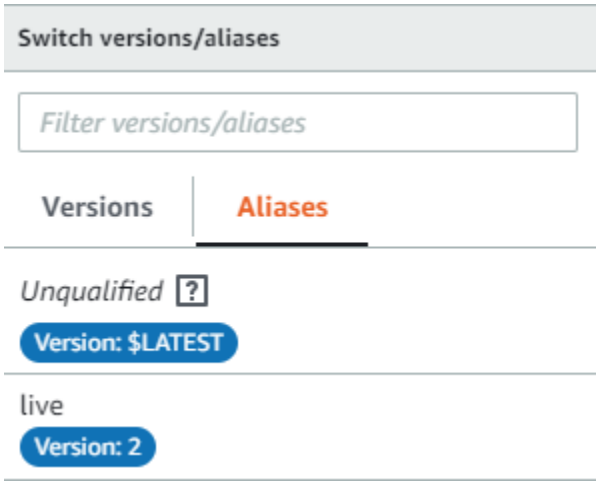
- デプロイされた関数をテストする
- CloudWatch Logs でのフックイベントを表示します。

デプロイされた関数をテストする

sam deploy のコマンドは、`my-date-time-app-myDateTimeFunction` のLambda 関数を更新します。関数のバージョンが 2 に更新され、`live` エイリアスに追加されます。

Lambda コンソール中の更新を見るには

1. https://console.aws.amazon.com/lambda/ で AWS Lambda コンソールを開きます。
2. ナビゲーションペインで、`my-date-time-app-myDateTimeFunction` 関数を選択します。コンソールでは、名前に識別子が含まれているため、`my-date-time-app-myDateTimeFunction-123456ABCDEF` のようになります。
3. [Qualifiers (修飾子)]、[エイリアス] の順に選択します。デプロイが完了した後(約 10 分)、`live` エイリアスに [バージョン: 2] と表示されます。



4. [関数コード] で、関数のソースコードを表示します。変更が表示されます。
5. (オプション)ステップ 2: Lambda 関数を更新します のテスト手順を使用して、更新された関数をテストできます。次のペイロードを使用して新しいテストイベントを作成し、結果に現在の時、分、秒が含まれていることを確認します。

```
{
    "option": "time"
  }
```

を使用して更新された関数を AWS CLI テストするには、次のコマンドを実行し、 `out.txt`を開いて、結果に現在の時間、分、秒が含まれていることを確認します。

```
aws lambda invoke --function your-function-arn --payload "{\"option\": \"time\"}"
 out.txt
```

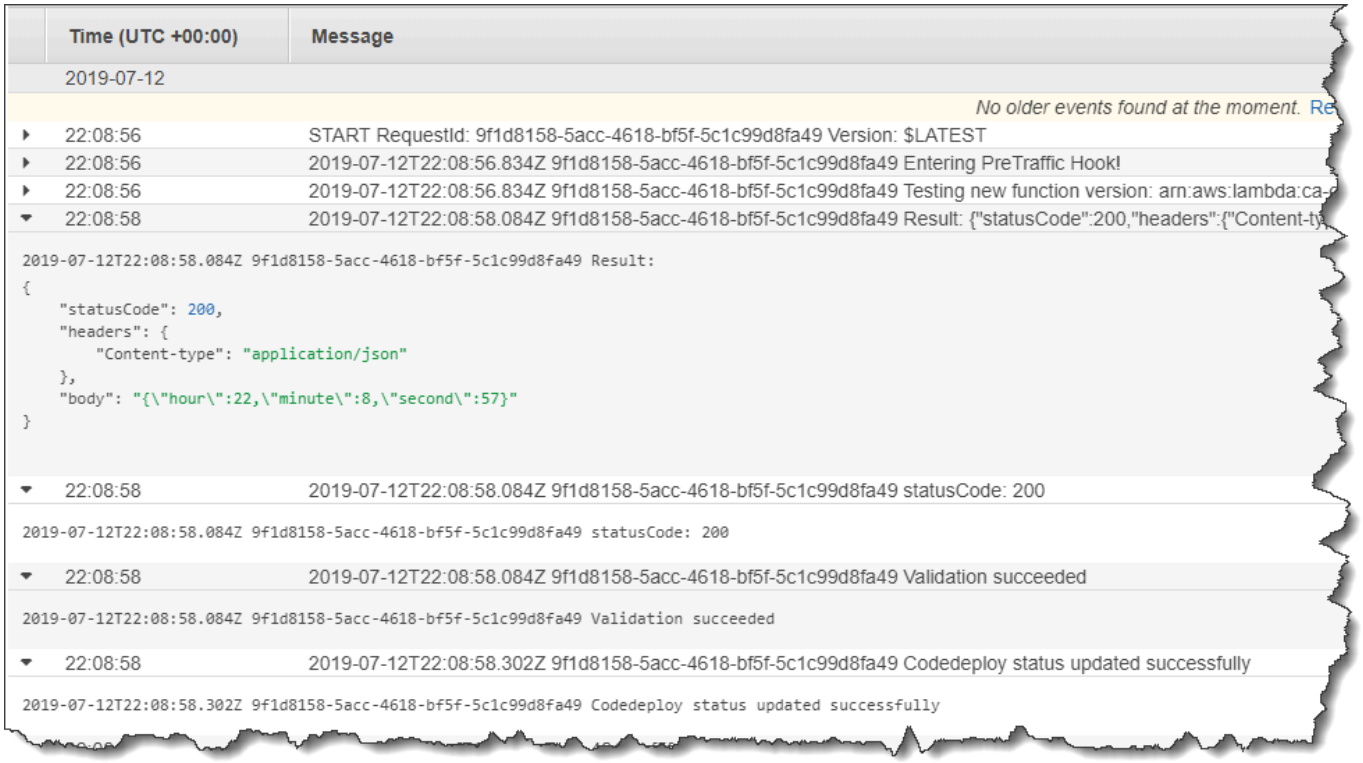
> **Note**
>
> を使用してデプロイが完了する前に関数を AWS CLI テストすると、予期しない結果が表示されることがあります。これは、CodeDeploy がトラフィックの 10% を毎分更新バージョンに段階的に移行するためです。デプロイ中、一部のトラフィックは引き続き元のバージョンを指すため、aws lambda invoke は元のバージョンを使用する場合があります。10 分後には、デプロイが完了し、すべてのトラフィックが関数の新しいバージョンを指し示します。

CloudWatch Logs でのフックイベントを表示します。

BeforeAllowTraffic のフック中に、CodeDeploy は CodeDeployHook_beforeAllowTraffic の Lambda 関数を実行します。AfterAllowTraffic のフック中に、CodeDeploy は CodeDeployHook_afterAllowTraffic の Lambda 関数を実行します。各関数は、新しい time パラメータを使用して関数の更新バージョンを呼び出す検証テストを実行します。Lambda 関数の更新が成功した場合、time のオプションによるエラーは発生せず、検証は成功します。関数が更新されなかった場合、認識されないパラメータによってエラーが発生し、検証が失敗します。これらの検証テストはデモンストレーションのみを目的としています。デプロイを検証するには独自のテストを記述します。CloudWatch Logs コンソールを使用して、検証テストを表示できます。

CodeDeploy フックイベントを表示するには

1. CloudWatch コンソールの https://console.aws.amazon.com/cloudwatch/ を開いてください。
2. ナビゲーションペインで、[Logs (ログ)] を選択します。
3. ロググループのリストから、[/aws/lambda/CodeDeployHook_beforeAllowTraffic] または [/aws/lambda/CodeDeployHook_afterAllowTraffic] を選択します。
4. ログストリームを選択します。1 つのみ表示されます。
5. イベントを展開して詳細を表示します。



ステップ 5: クリーンアップ

このチュートリアルで使用したリソースの追加料金を回避するには、 AWS SAM テンプレートによって作成されたリソースと、Lambda 検証関数によって作成された CloudWatch ログを削除します。

CloudFormation スタックを削除するには

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/cloudformation で CloudFormation コンソールを開きます。
2. [スタック] 列で、`my-date-time-app` スタックを選択し、[削除] を選択します。
3. プロンプトが表示されたら、[スタックの削除] を選択します。Lambda 関数、CodeDeploy アプリケーションとデプロイグループ、および によって作成された IAM ロール AWS SAM は削除されます。

CloudWatch Logs でログを削除するには

1. CloudWatch コンソールの https://console.aws.amazon.com/cloudwatch/ を開いてください。
2. ナビゲーションペインで、[Logs (ログ)] を選択します。

3. ロググループのリストから、[/aws/lambda/CodeDeployHook_beforeAllowTraffic] の横にあるボタンを選択します。
4. [アクション] から、[ロググループを削除する] を選択し、次に [はい、削除します] を選択します。
5. ロググループのリストから、[/aws/lambda/CodeDeployHook_afterAllowTraffic] の横にあるボタンを選択します。
6. [アクション] から、[ロググループを削除する] を選択し、次に [はい、削除します] を選択します。

CodeDeploy エージェントの使用

AWS CodeDeploy エージェントは、インスタンスにインストールして設定すると、そのインスタンスを CodeDeploy デプロイで使用できるようにするソフトウェアパッケージです。

AWS は、CodeDeploy エージェントの最新のマイナーバージョンをアクティブにサポートしています。最新のリリースバージョンについては、「」を参照してくださいCodeDeploy エージェントのバージョン履歴。

> **Note**
>
> CodeDeploy エージェントは、EC2/オンプレミスのコンピューティングプラットフォームにデプロイする場合にのみ必要です。Amazon ECS または AWS Lambda コンピューティングプラットフォームを使用するデプロイには、このエージェントは必要ありません。

エージェントがインストールされている場合、設定ファイルはインスタンスに配置されます。このファイルは、エージェントの動作を指定するために使用されます。この設定ファイルでは、 がインスタンスとやり取り AWS CodeDeploy するときに使用するディレクトリパスやその他の設定を指定します。ファイルの一部の設定オプションは変更できます。CodeDeploy エージェント設定ファイルの使用の詳細については、「CodeDeploy エージェント設定リファレンス」を参照してください。

インストール、更新、バージョンの確認のステップなど、CodeDeploy エージェントの使用に関する詳細については、「CodeDeploy エージェントのオペレーションの管理」を参照してください。

トピック

- CodeDeploy エージェントで対応するオペレーティングシステム
- CodeDeploy エージェントの通信プロトコルとポート
- CodeDeploy エージェントのバージョン履歴
- CodeDeploy プロセスの管理
- アプリケーションリビジョンとログファイルのクリーンアップ
- CodeDeploy エージェントでインストールされるファイル
- CodeDeploy エージェントのオペレーションの管理

CodeDeploy エージェントで対応するオペレーティングシステム

対応する Amazon EC2 AMI オペレーティングシステム

CodeDeploy エージェントは、以下の Amazon EC2 AMI オペレーティングシステムでテスト済みです。

- Amazon Linux 2023 (x86_64、aarch64)
- Amazon Linux 2 (x86_64、aarch64)
- Red Hat Enterprise Linux (RHEL) 8、9、10 (x86_64、aarch64)
- Oracle Linux 8、9、10 (x86_64、aarch64)
- Rocky Linux 9、10 (x86_64、aarch64)
- CentOS ストリーム 9、10 (x86_64、aarch64)
- SLES 15 (x86_64、aarch64)
- Debian 11、12、13 (x86_64、aarch64)
- Ubuntu Server 16.04 LTS、18.04 LTS、20.04 LTS、22.04 LTS、24.04 LTS、25.04、26.04 LTS (x86_64、aarch64)
- Windows Server 2016、2019、2022、2025 (x86_64)
- Apple シリコンを搭載した EC2 Mac インスタンス上の macOS 15、26、27 (aarch64)

> **Note**
>
> CodeDeploy エージェントバージョン 2.0.x 以降でのみ、次のプラットフォームがテストされています。Windows Server 2025、RHEL 10、Oracle Linux 8、9、10、Rocky Linux 9、10、CentOS Stream 9、10、SLES 15、Debian 11、12、13、および Ubuntu Server 24.04、25.04、26.04。
> macOS は CodeDeploy エージェントバージョン 2.1.2 以降でのみサポートされています。エージェントは、Apple シリコンを使用した macOS 15、26、27 をサポートしています。

CodeDeploy エージェントは、ニーズに適応できるようオープンソースとして利用できます。他の Amazon EC2 AMI オペレーションシステムでも使用できます。詳細については、GitHub の CodeDeploy エージェント リポジトリを参照してください。

サポートされているオンプレミスオペレーションシステム

CodeDeploy エージェントは次のオンプレミスオペレーティングシステムでテスト済みです。

- Windows Server 2016、2019、2022、2025 (x86_64)
- Red Hat Enterprise Linux (RHEL) 8、9、10 (x86_64、aarch64)
- Oracle Linux 8、9、10 (x86_64、aarch64)
- Rocky Linux 9、10 (x86_64、aarch64)
- CentOS ストリーム 9、10 (x86_64、aarch64)
- SLES 15 (x86_64、aarch64)
- Debian 11、12、13 (x86_64、aarch64)
- Ubuntu Server 16.04 LTS、18.04 LTS、20.04 LTS、22.04 LTS、24.04 LTS、25.04、26.04 LTS (x86_64、aarch64)
- macOS 15、26、27 と Apple シリコン (aarch64)

> **Note**
>
> CodeDeploy エージェントバージョン 2.0.x 以降でのみ、次のプラットフォームがテストされています。Windows Server 2025、RHEL 10、Oracle Linux 8、9、10、Rocky Linux 9、10、CentOS Stream 9、10、SLES 15、Debian 11、12、13、および Ubuntu Server 24.04、25.04、26.04。
> macOS は CodeDeploy エージェントバージョン 2.1.2 以降でのみサポートされています。エージェントは、Apple シリコンを使用した macOS 15、26、27 をサポートしています。

CodeDeploy エージェントは、ニーズに適応できるようオープンソースとして利用できます。他のオンプレミスインスタンスオペレーティングシステムで使用できます。詳細については、GitHub の CodeDeploy エージェント リポジトリを参照してください。

CodeDeploy エージェントの通信プロトコルとポート

CodeDeploy エージェントは、ポート 443 上の HTTPS を使用してアウトバウンド通信を行います。

CodeDeploy エージェントが EC2 インスタンスで実行されると、EC2 メタデータ エンドポイントを使用して、インスタンス関連の情報を取得します。詳細については、「インスタンスメタデータサービスの制限」を参照してください。

CodeDeploy エージェントのバージョン履歴

インスタンスでは、 CodeDeploy エージェントのサポートされているバージョンが実行されている必要があります。現在サポートされている最小バージョンは 2.1.x です。

> **Note**
>
> 最新バージョンの CodeDeploy エージェントを使用することをお勧めします。問題が発生した場合は、 AWS サポートに連絡する前に最新バージョンに更新してください。アップグレード情報については、「CodeDeploy エージェントを更新する」を参照してください。

> **バージョン 2.x のロールアウトステータス**
>
> CodeDeploy エージェントのバージョン 2.x は 全体に展開されており AWS リージョン 、一部のリージョンではまだ利用できない場合があります。バージョン 2.x はオプトインアップグレードです。古いリビジョンからバージョン 2.x にアップグレードするには、「」を参照してくださいCodeDeploy エージェントをインストールする。インストールできるバージョンを確認するには、「」を参照してくださいで利用可能な最新のエージェントバージョンを確認する AWS リージョン。

次の表は、CodeDeploy エージェントのすべてのリリースと、各バージョンに含まれる機能や強化された機能の一覧です。

バージョン	リリース日	詳細
2.1.2	2026 年 10 月 13 日	追加: Apple シリコンを使用した macOS 15、26、27 のサポート。手順については、「macOS 用の CodeDeploy エージェントをインストールする」を参照してください。 追加: Amazon S3 からのアプリケーションリビジョンの並列範囲のダウンロード。デフォルトでは無効になっています。詳細については、「CodeDeploy エージェント設定リファレンス」を参照してください。

バージョン	リリース日	詳細
		修正: ライフサイクルイベントスクリプトの出力は、デプロイ ID のプレフィックスが付いた行`codedeploy-agent-deployments.log` ごとに書き込まれます。
2.1.1	2026 年 10 月 2 日	修正: エージェントは、 AWS 欧州主権クラウドの正しい CodeDeploy エンドポイントに接続するようになりました。
2.1.0	2026 年 9 月 7 日	追加: `RESTART`デプロイモードのネイティブサポート。これにより、デプロイグループの最後に成功したデプロイのリビジョンを使用して、ターゲットインスタンスでアプリケーションを再起動します。エージェントは、インスタンスに既にインストールされているリビジョンのコピーから をインストールします。そのコピーが欠落しているか不完全である場合、エージェントはリビジョンをダウンロードするため、デプロイは完了します。 デプロイモード自体については、 AWS CodeDeploy API リファレンスの deploymentMode を参照してください。 変更 (セキュリティ): エージェントは、アプリケーションリビジョンディレクトリの外部で AppSpec ファイルパスが解決されるデプロイを拒否するようになりました。など、リビジョン内にネストされたパス`configs/appspec.yml` は引き続きサポートされます。
2.0.1	2026 年 8 月 24 日	修正: Windows では、バックスラッシュ (\) または複数の区切り文字 (など`//my-folder`) で始まる AppSpec ファイルパスが、ドライブのルートではなくリビジョン内で解決されるようになりました。 サポートされている`source`値の例については、「」を参照してくださいAppSpec の「ファイル」セクション (EC2/オンプレミスデプロイのみ)。

バージョン	リリース日	詳細
2.0.0	2026 年 7 月 15 日	変更: CodeDeploy エージェントは Rust ベースになり、単一の自己完結型ネイティブバイナリとして出荷されます。Ruby ランタイムの依存関係が削除されました。エージェントをインストールまたは実行するために Ruby が不要になりました。エージェントはデフォルトでバージョン 1.8.x と動作互換性があります。既存の設定ファイル、AppSpec 処理、ライフサイクルフックセマンティクス、ディスク上のレイアウト、およびデプロイ結果は、以下に記載がない限り変更されません。 追加: Windows Server 2025、RHEL 10、Ubuntu Server 26.04 などのオペレーティングシステムのサポートが拡張され、追加の Linux ディストリビューションと ARM (aarch64) バリアントが追加されました。このバージョンでテストされたプラットフォームの完全なリストについては、「」を参照してくださいCodeDeploy エージェントで対応するオペレーティングシステム。 追加: オンデマンドエージェントの自己更新のcodedeploy-agent update コマンド。 追加: ローカルコマンドポート。これは、ループバックアドレス (127.0.0.1) のみをリッスンしてエージェントのステータスをクエリし、コマンドをローカルに挿入できるオプションの TCP 管理インターフェイスです。デフォルトでは無効になっており、 で有効になっていますenable_command_port 。 変更: ローカルデプロイコマンドが になりましたcodedeploy-agent deploy-local 。codedeploy-local コマンドは下位互換性のために引き続き使用できます。すべてのコマンドオプションは保持され、ローカルパス、Amazon S3 (s3://)、GitHub バンドルソースは引き続きサポートされます。

バージョン	リリース日	詳細
		変更: Linux では、エージェントはネイティブ systemd ユニット (codedeploy-agent.service) によって管理されます。SysV init.d スクリプトはインストールされなくなります。systemctl を使用してエージェントを管理します。 変更 (セキュリティ): TLS 証明書の検証は常に有効になっており、コアダンプはデフォルトで無効になっています。 追加: 独自のセキュリティ要件を満たすために有効にできるオプションの強化設定。各設定はデフォルトでオフになっており、個別に有効にします。シンボリックリンク、パストラバーサル、または安全でないアクセス許可を含むバンドルを拒否したり、ライフサイクルイベントフックの環境を制限したり、エージェントディレクトリとログファイルのアクセス許可を制限したりできます。利用できるタイムゾーンの一覧については、「CodeDeploy エージェント設定リファレンス」を参照してください。 変更: コマンドラインインストールスクリプトが Bash (以前の Ruby) で記述されるようになりました。curl または が必要ですwget。 変更: バージョン 2.0.x のインストールスクリプトは、リージョンaws-codedeploy- *region-identifier* バケットの latestv2/ プレフィックスで公開されます。latest/ プレフィックスは、引き続きバージョン 1.8.x のインストールスクリプトを提供します。 Changed: AWS バージョン 2.0.x を AWSCodeDeployAgent V2 Systems Manager Distributor パッケージとして公開しました。AWSCodeDeployAgent パッケージは引き続きバージョン 1.8.x を提供します。 削除: ssl_verify_peer 設定と Ruby ProcessManager 設定キー。

バージョン	リリース日	詳細
		重要: バージョン 1.8.x から 2.0.0 への自動更新パスは現在ありません。2.0.0 にアップグレードするには、`AWSCodeDeployAgentV2` Systems Manager Distributor パッケージでインストールするか、リージョン`latestv2/install` スクリプトを手動で実行します。手順については、「CodeDeploy エージェントをインストールする」を参照してください。
1.8.1	2026 年 2 月 3 日	修正: S3 エンドポイントのバグ修正。 追加: Windows CA リストへの Sectigo CA 証明書。
1.8.0	2025 年 7 月 31 日	変更: Windows 用 CodeDeploy エージェントでバンドルされた Ruby を 3.2 にアップグレードしました。
1.7.1	2024 年 11 月 14 日	変更: セキュリティパッチ用に依存関係を更新しました。
1.7.0	2024 年 3 月 6 日	追加: CodeDeploy エージェント設定ファイルの `:disable_imds_v1:` 設定。この設定を使用して、IMDSv2 エラーが発生したときに IMDSv1 へのフォールバックを無効にします。デフォルトは `false` です (フォールバックを有効にします)。詳細については、「CodeDeploy エージェント設定リファレンス」を参照してください。 追加: Red Hat Enterprise Linux 9 (RHEL 9) オペレーティングシステムのサポート。 追加: Ubuntu Server での Ruby バージョン 3.1 および 3.2 のサポート。 修正: CodeDeploy エージェント設定ファイルのロードに失敗すると、CodeDeploy エージェントはユーザーフレンドリーなエラーを生成するようになりました。 変更: Windows 用 CodeDeploy エージェントで Ruby を 2.7.8-1 にアップグレードしました。

バージョン	リリース日	詳細
1.6.0	2023 年 3 月 30 日	追加: Ruby 3.1、3.2 に対応しました。 追加: Amazon Linux 2023 に対応しました。 追加: Windows Server 2022 に対応しました。 変更: Windows Server インスタンスでは、デフォルトの `verbose` の設定は `false` になりました。Windows で引き続きデバッグメッセージをログファイルに出力するには、`verbose` を `true` に設定する必要があります。 削除: Windows Server 2016 および Windows Server 2012 R2 のサポートが削除されました。 削除: Amazon Linux 2018.03.x が削除されました。
1.5.0	2023 年 3 月 3 日	追加: Ruby 3 に対応しました。 追加: Ubuntu 22.04 に対応しました。 修正: 起動後すぐに CodeDeploy エージェントを再起動すると、エージェントがハングアップする問題を修正しました。 変更: CodeDeploy エージェントは、フックスクリプトの実行中にエージェントサービスが予期せず再起動した場合、エージェント起動時にホストデプロイに失敗するようになりました。この修正により、70 分のタイムアウト時間を待ってからデプロイを再試行する必要がなくなりました。 廃止通知: CodeDeploy エージェント 1.5.0 は Windows Server 2016 と Windows Server 2012 R2 をサポートする最後のリリースです。 削除: Ubuntu 14.04 LTS、Windows Server 2008 R2、および Windows Server 2008 R2 32 ビットでの CodeDeploy エージェントのサポートが削除されました。

バージョン	リリース日	詳細
1.4.1	2022 年 12 月 6 日	修正: ログ記録に関連するセキュリティの脆弱性を修正しました。 強化: ホストコマンドのポーリング時のログ記録を改善しました。

バージョン	リリース日	詳細
1.4.0	2022 年 8 月 31 日	追加: Red Hat Enterprise Linux 8 に対応しました。 追加: Windows 用の CodeDeploy エージェントでの長いファイルパスに対応しました。長いファイルパスを有効にするには、適切な Windows レジストリキーを設定し、エージェントを再起動する必要があります。詳細については、「ファイルパスが長いと、「そのようなファイルまたはディレクトリはありません」というエラーが発生します」を参照してください。 修正: ディスクがいっぱいになったときの解凍オペレーションに関する問題を修正しました。CodeDeploy エージェントは、ディスクがいっぱいであることを示す解凍の終了コード 50 を検出し、部分的に抽出されたファイルを削除して、例外を発生させて失敗を CodeDeploy サーバーに送信するようになりました。このエラーメッセージはライフサイクルイベントのエラーメッセージとして表示され、ホストレベルのデプロイはスタックしたりタイムアウトしたりすることなく停止します。 修正: エージェントが失敗する原因となる問題を修正しました。 修正: エッジケースの競合状態時にフックがタイムアウトする問題を修正しました。スクリプトのないフックは引き続き動作するようになり、障害やタイムアウトが発生しなくなりました。 変更: CodeDeploy エージェントの `bin` ディレクトリにある `update` スクリプトは、使用されなくなったため削除されました。 変更: Windows Server 用の CodeDeploy エージェントは Ruby 2.7 がバンドルされました。

バージョン	リリース日	詳細
		変更: デプロイバンドルのソース (Amazon S3 または GitHub) に応じてフックスクリプトが使用する新しい環境変数が追加されました。 詳細については、「フックの環境変数の可用性」を参照してください。 **Important** 廃止通知: CodeDeploy エージェント 1.4.0 は、32 ビット Windows Server 用のインストーラーが含まれる最後のリリースです。 廃止通知: CodeDeploy エージェント 1.4.0 は Windows Server 2008 R2 をサポートする最後のリリースです。 削除: Amazon Linux 2014.09、2016.03、2016.09、2017.03 の Amazon EC2 AMI での CodeDeploy エージェントに対応しました。

バージョン	リリース日	詳細
1.3.2	2021 年 5 月 6 日	**Important** CodeDeploy エージェント 1.3.2 は、エージェントを実行している Windows ホストに影響を与える CVE-2018-1000201 に対応しています。CVE は、CodeDeploy エージェントの依存関係である ruby-ffi を挙げています。エージェントが Amazon EC2 Systems Manager (SSM) とともにインストールされ、自動的に更新するように設定されている場合、アクションは必要ありません。それ以外の場合は、エージェントを手動で更新するためのアクションが必要になります。エージェントをアップグレードするには、「Windows サーバーで CodeDeploy エージェントを更新する」の手順に従います。 修正: CodeDeploy エージェントを Ubuntu 20.04 以降にインストールする際の問題。 修正: 圧縮ファイルを抽出する際に、相対パスが正しく処理されていないために発生する断続的な問題。 追加: Windows インスタンスの AWS PrivateLink および VPC エンドポイント の対応。 追加: AppSpec ファイルの改善について、以下に説明します。 • ローカルデプロイを作成する時に、AppSpec ファイルにカスタムファイル名を指定できるようになりました。詳細については、「ローカルのデプロイを作成する。」を参照してください。 • AppSpec ファイルに `.yaml` 拡張子をつけることができるようになりました。

バージョン	リリース日	詳細
		• AppSpec ファイルの新しいオプションの `file_exists_behavior` 設定を使用して、デプロイされたファイルを上書きできるようになりました。詳細については、「AppSpec の「ファイル」セクション (EC2/オンプレミスデプロイのみ)」を参照してください。 アップグレード: CodeDeploy が、Ruby 3.0 用の AWS SDK を使用するようになりました。
1.3.1	2020 年 12 月 22 日	修正: オンプレミスのインスタンスが起動しない 1.3.0 の問題。
1.3.0	2020 年 11 月 10 日	**Important** このバージョンは非推奨です。 修正: 使用されなくなった期限切れの証明書を削除しました。 修正: が使用するエージェントのアンインストールスクリプトからプロンプトメッセージを削除し AWS Systems Manager、ホストまたはフリートを以前のバージョンのエージェントにダウングレードしやすくしました。

バージョン	リリース日	詳細
1.2.1	2020 年 9 月 23 日	変更: v2 から v3 へ AWS SDK for Ruby 依存関係をアップグレードしました。 追加: IMDSv2 に対応しました。IMDsv2 http リクエストが失敗した場合、IMDSv1 へのサイレントフォールバックを含みます。 変更: セキュリティパッチ用に Rake と Rubyzip の依存関係を更新しました。 修正: 空の PID ファイルが、`No CodeDeploy Agent Running` のステータスを返すことを確認し、エージェントの起動時に PID ファイルをクリーンアップします。

バージョン	リリース日	詳細
1.1.2	2020 年 8 月 4 日	追加: Ubuntu Server 19.10 および 20.04 に対応しました。 注意: バージョン 19.10 は使用期限を迎えたため、Ubuntu および CodeDeploy のサポートは終了しました。 追加: Linux と Ubuntu のメモリ効率を改善し、予約メモリをよりタイムリーにリリースできるようになりました。 追加: Windows Server の [サイレントクリーンクリーンアップ] との互換性より、エージェントが応答しなくなることがありました。 追加: デプロイ時の失敗を避けるために、クリーンアップ中に空でないディレクトリを無視します。 追加: ロサンゼルス (LA) の AWS ローカルゾーンのサポート。 追加: インスタンスメタデータから AZ を抽出して、 AWS ローカルゾーンの互換性を提供します。 追加: ユーザーはサブディレクトリにアーカイブを提供できるようになり、ルートディレクトリに保存する必要はありません。 追加: Rubyzip でメモリリークが発生する可能性のある問題を検出しました。Rubyzip を使用する前に、まずシステムにインストールされている unzip ユーティリティの使用を試みるように、unzip コマンドを更新しました。 追加: エージェント構成設定としての `:enable_auth_policy:` 。 変更: Unzip の警告が無視されるようになり、デプロイが継続されるようになりました。

バージョン	リリース日	詳細
1.1.0	2020 年 6 月 30 日	変更: CodeDeploy エージェントのバージョニングが Ruby の標準のバージョニング規約に従うようになりました。 追加: コマンドラインから特定のエージェントバージョンをインストールできる、インストールおよび更新コマンドの新しいパラメータ。 削除: Linux および Ubuntu 用の CodeDeploy エージェント Auto Update を削除しました。CodeDeploy エージェントの自動更新を設定するには、「 を使用して CodeDeploy エージェントをインストールする AWS Systems Manager」を参照してください。
1.0.1.1597	2018 年 11 月 15 日	機能強化: CodeDeploy は Ubuntu 18.04 を対応しました。 機能強化: CodeDeploy は Ruby 2.5 をサポートします。 機能強化: CodeDeploy は FIPS エンドポイントをサポートします。FIPS エンドポイントの詳細については、「FIPS 140-2 の概要」を参照してください。CodeBuild で使用できるエンドポイントについては、CodeDeploy リージョンとエンドポイント」を参照してください。
1.0.1.1518	2018 年 6 月 12 日	機能強化: ポールリクエストの受け入れ中に CodeDeploy エージェントを閉じたときにエラーが発生する問題が修正されました。 機能強化: デプロイの進行中に CodeDeploy エージェントを閉じられないようにするデプロイ追跡機能を追加しました。 機能強化: ファイルを削除する際のパフォーマンスが改善されました。

バージョン	リリース日	詳細
1.0.1.1458	2018 年 3 月 6 日	注: このバージョンは現在サポートされていません。このバージョンを使用すると、デプロイに失敗することがあります。 機能強化: より多くの信頼された機関をサポートするため、証明書の検証を改善しました。 機能強化: ローカル CLI が BeforeInstall ライフサイクルイベントを含むデプロイに失敗していた問題を修正しました。 機能強化: CodeDeploy エージェントの更新された時に、アクティブなデプロイが失敗する可能性がある問題を修正しました。
1.0.1.1352	2017 年 11 月 16 日	注: このバージョンは現在サポートされていません。このバージョンを使用すると、デプロイに失敗することがあります。 機能導入: CodeDeploy エージェントがインストールされているローカルマシンまたはインスタンスで、EC2/オンプレミスのデプロイをテストおよびデバッグする新機能が導入されました。
1.0.1.1106	2017 年 5 月 16 日	注: このバージョンは現在サポートされていません。このバージョンを使用すると、デプロイに失敗することがあります。 特徴: 前回の成功したデプロイのアプリケーションリビジョンの一部ではない、デプロイ先のコンテンツを処理する新しいサポートを導入しました。既存のコンテンツのデプロイオプションとして、コンテンツの保持、コンテンツの上書き、またはデプロイの失敗が追加されました。 機能強化: CodeDeploy エージェントをバージョン 2.9.2 AWS SDK for Ruby (aws-sdk-core 2.9.2) と互換性を持たせました。

バージョン	リリース日	詳細
1.0.1.1095	2017 年 3 月 29 日	注: このバージョンは現在サポートされていません。このバージョンを使用すると、デプロイに失敗することがあります。 機能強化: 中国 (北京) リージョンに CodeDeploy エージェントのサポートが導入されました。 機能強化: ライフサイクルイベントフックから呼び出されたときに Windows Server インスタンスで Puppet が実行されるようになりました。 機能強化: `untar` オペレーションの処理が改善されました。
1.0.1.1067	2017 年 1 月 6 日	注: このバージョンは現在サポートされていません。このバージョンを使用すると、デプロイに失敗することがあります。 機能強化: 多くのエラーメッセージを改訂し、デプロイの失敗に関するより具体的な原因を含めました。 機能強化: CodeDeploy エージェントが一部のデプロイ中にデプロイする正しいアプリケーションリビジョンを特定できない問題を修正しました。 機能強化: `untar` オペレーションの前後の `pushd` と `popd` の使用を元に戻しました。
1.0.1.1045	2016 年 11 月 21 日	注: このバージョンは現在サポートされていません。このバージョンを使用すると、デプロイに失敗することがあります。 機能強化: CodeDeploy エージェントをバージョン 2.6.11 AWS SDK for Ruby (aws-sdk-core 2.6.11) と互換性を持たせました。

バージョン	リリース日	詳細
1.0.1.1037	2016 年 10 月 19 日	注: このバージョンは現在サポートされていません。このバージョンを使用すると、デプロイに失敗することがあります。 Amazon Linux、RHEL、および Ubuntu Server インスタンスの CodeDeploy エージェントを更新して、以下の変更を反映しました。Windows Server インスタンスの場合、最新バージョンは 1.0.1.998 のままです。 機能強化: エージェントは、インスタンスにインストールされている Ruby のバージョンを特定し、そのバージョンを使用して `codedeploy-agent` スクリプトを呼び出すことができるようになりました。
1.0.1.1011.1	2016 年 8 月 17 日	注: このバージョンは現在サポートされていません。このバージョンを使用すると、デプロイに失敗することがあります。 機能強化: シェルのサポートの問題により、バージョン 1.0.1.1011 で導入された変更を削除しました。このバージョンのエージェントは、2016 年 7 月 11 日にリリースされたバージョン 1.0.1.998 と機能的に同じものです。

バージョン	リリース日	詳細
1.0.1.1011	2016 年 8 月 15 日	注: このバージョンは現在サポートされていません。このバージョンを使用すると、デプロイに失敗することがあります。 Amazon Linux、RHEL、および Ubuntu Server インスタンスの CodeDeploy エージェントを更新して、以下の変更を反映しました。Windows Server インスタンスの場合、最新バージョンは 1.0.1.998 のままです。 機能導入: systemd init システムが使用されているオペレーションシステムで、Bash シェルを使用して CodeDeploy エージェントを呼び出すサポートが追加されました。 機能強化: CodeDeploy エージェントおよび CodeDeploy エージェントアップデータで Ruby 2.x のすべてのバージョンに対するサポートが有効になりました。更新された CodeDeploy エージェントは、Ruby 2.0 のみに依存しなくなりました (CodeDeploy エージェントインストーラの deb および rpm バージョンでは、Ruby 2.0 が引き続き必要です)。
1.0.1.998	2016 年 7 月 11 日	注: このバージョンは現在サポートされていません。このバージョンを使用すると、デプロイに失敗することがあります。 機能強化: CodeDeploy エージェントを ルート 以外のユーザープロファイルで実行する場合のサポートを修正しました。環境変数の競合を回避するため、USER いう名前の変数は CODEDEPLOY_USER で置き換えられました。

バージョン	リリース日	詳細
1.0.1.966	2016 年 6 月 16 日	注: このバージョンは現在サポートされていません。このバージョンを使用すると、デプロイに失敗することがあります。 機能導入: CodeDeploy エージェントを ルート 以外のユーザープロファイルで実行する場合のサポートを導入しました。 機能強化: デプロイグループに対して CodeDeploy エージェントでアーカイブするアプリケーションリビジョン数を指定するサポートを修正しました。 機能強化: CodeDeploy エージェントを バージョン 2.3 AWS SDK for Ruby (aws-sdk-core 2.3) と互換性を持たせました。 機能強化: デプロイ中の UTF-8 エンコードに関する問題が修正されました。 機能強化: プロセス名を確認する際の精度が向上しました。
1.0.1.950	2016 年 3 月 24 日	注: このバージョンは現在サポートされていません。このバージョンを使用すると、デプロイに失敗することがあります。 機能: インストールプロキシのサポートを追加しました。 機能強化: 最新のバージョンがすでにインストールされている場合は、CodeDeploy エージェントをダウンロードしないように、インストールスクリプトを更新しました。
1.0.1.934	2016 年 2 月 11 日	注: このバージョンは現在サポートされていません。このバージョンを使用すると、デプロイに失敗することがあります。 機能導入: デプロイグループに対して CodeDeploy エージェントでアーカイブするアプリケーションリビジョン数を指定するサポートが導入されました。

バージョン	リリース日	詳細
1.0.1.880	2016 年 1 月 11 日	注: このバージョンは現在サポートされていないため、デプロイに失敗する可能性があります。 機能強化: CodeDeploy エージェントを バージョン 2.2 AWS SDK for Ruby (aws-sdk-core 2.2) と互換性を持たせました。バージョン 2.1.2 は引き続きサポートされます。
1.0.1.854	2015 年 11 月 17 日	注: このバージョンは現在サポートされていません。このバージョンを使用すると、デプロイに失敗することがあります。 特徴: SHA-256 ハッシュアルゴリズムのサポートが導入されました。 機能: `.version` ファイルでバージョンの追跡サポートが導入されました。 特徴: 環境変数の使用を通じて、デプロイグループ ID を利用できるようになりました。 機能強化: Amazon CloudWatch Logs を使用した CodeDeploy エージェントのログを監視するためのサポートを追加しました。

関連情報については、以下を参照してください。

- CodeDeploy エージェントのバージョンを特定します。
- CodeDeploy エージェントをインストールする

CodeDeploy エージェントのバージョン履歴については、「GitHub のリリースリポジトリ」を参照してください。

CodeDeploy プロセスの管理

CodeDeploy エージェントは systemd を使用して、サポートされているすべての Linux ディストリビューションでエージェントプロセスを管理します。

バージョン 2.0.x 以降では、エージェントはネイティブ systemd ユニット (codedeploy-agent.service) のみを出荷します。/etc/init.d/codedeploy-agent スクリプトはありません。systemctl を使用してエージェントを管理するか、start、、、stoprestartまたは statusサブコマンドを使用してバイナリを直接呼び出します。

バージョン 1.8.x 以前では、rpm ディストリビューションと deb ディストリビューションの両方にスタートアップスクリプトが に付属しています/etc/init.d/codedeploy-agent。systemd をサポートするシステムでは、 service コマンドを使用すると、systemd がプロセスを管理するのではなく、誤って init.d スクリプトを実行する可能性があります。この問題を回避するには、エージェントオペレーションsystemctlに を使用します。

CodeDeploy エージェントを管理するには、 systemctlユーティリティを使用します。

```
sudo systemctl start codedeploy-agent
sudo systemctl stop codedeploy-agent
sudo systemctl restart codedeploy-agent
sudo systemctl status codedeploy-agent
```

macOS では、エージェントは で定義されている起動されたデーモン として実行されます/Library/LaunchDaemons/com.amazon.aws.codedeploy-agent.plist。 を起動するとcom.amazon.aws.codedeploy-agent、起動時にエージェントが開始され、クラッシュした場合は再起動されます。macOS でエージェントを管理するには、 launchctlユーティリティを使用します。

```
sudo launchctl bootstrap system /Library/LaunchDaemons/com.amazon.aws.codedeploy-
agent.plist
sudo launchctl bootout system/com.amazon.aws.codedeploy-agent
sudo launchctl kickstart -k system/com.amazon.aws.codedeploy-agent
sudo launchctl print system/com.amazon.aws.codedeploy-agent
```

これらのコマンドは、それぞれエージェントのステータスを開始、停止、再起動、表示します。

アプリケーションリビジョンとログファイルのクリーンアップ

CodeDeploy エージェントは、インスタンス上にリビジョンとログファイルをアーカイブします。CodeDeploy エージェントは、ディスクスペースを節約するために、これらのアーティファクトをクリーンアップします。

アプリケーションリビジョンのデプロイログ: エージェント設定ファイルの :max_revisions: オプションを使用して、正の整数を入力することで、アーカイブするアプリケーションリビジョン数を指定

できます。CodeDeploy は、これらのリビジョンのログファイルもアーカイブします。その他すべては、最後に成功したデプロイのログファイルを除いて削除されます。失敗したデプロイの数が、保持されているバージョンの数を超えた場合でも、そのログファイルは常に保持されます。値を指定しない場合、CodeDeploy は現在デプロイされたリビジョンに加えて 5 つの最新のリビジョンを保持します。

CodeDeploy ログ: Amazon Linux、Ubuntu Server、RHEL、macOS インスタンスの場合、CodeDeploy エージェントは `/var/log/aws/codedeploy-agent`フォルダのログファイルをローテーションします。ログファイルは、毎日 00:00:00 (インスタンス時間) 、またはファイルが 64 MB に達したときのいずれか早い方にローテーションされます。同じ日に 2 回目のローテーションが発生した場合、エージェントは数値サフィックス (などcodedeploy-agent.20260715.1.log) を追加します。エージェントは 7 日後にログファイルを削除します。ローテーションされたログファイルの名前付けパターンは `codedeploy-agent.`*`YYYYMMDD`*`.log` です。

CodeDeploy エージェントでインストールされるファイル

CodeDeploy エージェントはリビジョン、デプロイ履歴、デプロイスクリプトをインスタンスのルートディレクトリに保存します。このディレクトリのデフォルトの名前と場所:

`'/opt/codedeploy-agent/deployment-root'` Amazon Linux、Ubuntu Server、RHEL、macOS インスタンス用。

Windows Server インスタンス用の `'C:\ProgramData\Amazon\CodeDeploy'`。

CodeDeploy エージェントの設定ファイルにある root_dir 設定を使用して、ディレクトリの名前と場所を設定することができます。詳細については、「CodeDeploy エージェント設定リファレンス」を参照してください。

次の例は、ルートディレクトリ内のファイルとディレクトリの構造を示しています。この構造は N 件のデプロイグループがあることを前提とし、各デプロイグループには N 件のデプロイが含まれています。

```
|--deployment-root/
|-- deployment group 1 ID
|    |-- deployment 1 ID
|    |    |-- Contents and logs of the deployment's revision
|    |-- deployment 2 ID
|    |    |-- Contents and logs of the deployment's revision
```

```
|    |-- deployment N ID
|    |    |-- Contents and logs of the deployment's revision
|-- deployment group 2 ID
|    |-- deployment 1 ID
|    |    |-- bundle.tar
|    |    |-- deployment-archive
|    |    |    | -- contents of the deployment's revision
|    |    |-- logs
|    |    |    | -- scripts.log
|    |-- deployment 2 ID
|    |    |-- bundle.tar
|    |    |-- deployment-archive
|    |    |    | -- contents of the deployment's revision
|    |    |-- logs
|    |    |    | -- scripts.log
|    |-- deployment N ID
|    |    |-- bundle.tar
|    |    |-- deployment-archive
|    |    |    | -- contents of the deployment's revision
|    |    |-- logs
|    |    |    | -- scripts.log
|-- deployment group N ID
|    |-- deployment 1 ID
|    |    |-- Contents and logs of the deployment's revision
|    |-- deployment 2 ID
|    |    |-- Contents and logs of the deployment's revision
|    |-- deployment N ID
|    |    |-- Contents and logs of the deployment's revision
|-- deployment-instructions
|    |-- [deployment group 1 ID]_cleanup
|    |-- [deployment group 2 ID]_cleanup
|    |-- [deployment group N ID]_cleanup
|    |-- [deployment group 1 ID]_install.json
|    |-- [deployment group 2 ID]_install.json
|    |-- [deployment group N ID]_install.json
|    |-- [deployment group 1 ID]_last_successful_install
|    |-- [deployment group 2 ID]_last_successful_install
|    |-- [deployment group N ID]_last_successful_install
|    |-- [deployment group 1 ID]_most_recent_install
|    |-- [deployment group 2 ID]_most_recent_install
|    |-- [deployment group N ID]_most_recent_install
|-- deployment-logs
|    |-- codedeploy-agent-deployments.log
```

- Deployment Group ID フォルダは各デプロイグループを示しています。デプロイグループのディレクトリ名は、その ID です (例: `acde1916-9099-7caf-fd21-012345abcdef`)。各デプロイグループのディレクトリには、そのデプロイグループで試みた各デプロイのサブディレクトリ 1 つが含まれています。

  batch-get-deployments コマンドを使用してデプロイグループ ID を検索できます。
- デプロイ ID フォルダはデプロイグループの各デプロイを示します。各デプロイディレクトリの名前はその ID です。各フォルダには以下が含まれています。
  - bundle.tar はデプロイのリビジョンのコンテンツを含む圧縮ファイルです。リビジョンを表示する場合は、zip 圧縮解除ユーティリティを使用してください。
  - deployment-archive はデプロイのリビジョンのコンテンツを含むディレクトリです。
  - logs は `scripts.log` ファイルを含むディレクトリです。このファイルには、デプロイの AppSpec ファイルで指定されたスクリプトすべての出力が一覧表示されます。

  デプロイのフォルダを見つけたいが、そのデプロイ ID またはデプロイグループ ID がわからない場合は、AWS CodeDeploy コンソールまたは AWS CLI を使用して見つけることができます。詳細については、「CodeDeploy デプロイの詳細を表示する」を参照してください。

  デプロイグループでアーカイブできるデプロイのデフォルト最大数は 5 件です。最大数に達すると、その後のデプロイがアーカイブされ、一番古いアーカイブは削除されます。CodeDeploy エージェントの設定ファイルで max_revisions 設定を使用すればデフォルトを変更できます。詳細については、「CodeDeploy エージェント設定リファレンス」を参照してください。

  > **Note**
  >
  > アーカイブしたデプロイが使用したハードディスク容量を復元するには、max_revisions 設定を 1 や 2 といった低い数値に変更してください。次のデプロイでは、アーカイブされたデプロイが削除され、その数が指定した数と等しくなります。
- deployment-instructions には各デプロイグループのテキストファイル 4 件が含まれています。
  - [デプロイグループ ID]-クリーンアップ。デプロイ中に実行される各コマンドの元に戻すバージョンを含むテキストファイル。サンプルファイルの名前は `acde1916-9099-7caf-fd21-012345abcdef-cleanup` です。
  - [Deployment Group ID]-install.json は最新のデプロイ中に作成された JSON ファイルです。これにはデプロイ中に実行するコマンドが含まれています。サンプルファイルの名前は `acde1916-9099-7caf-fd21-012345abcdef-install.json` です。

  - [デプロイグループ ID]_last_successful_install。最後に成功したデプロイのアーカイブディレクトリを一覧表示するテキストファイル。これは CodeDeploy エージェントがデプロイアプリケーションのファイルすべてをインスタンスにコピーした時に作成されたファイルです。次回のデプロイで、ApplicationStop スクリプトと BeforeInstall スクリプトのどちらを実行するか決定するために CodeDeploy エージェントを使用します。サンプルファイルの名前は acde1916-9099-7caf-fd21-012345abcdef_last_successful_install です。
  - [Deployment Group ID]_most_recent_install は、最新のデプロイのアーカイブディレクトリ名をリストにしたテキストファイルです。このファイルはデプロイ内のファイルが正常にダウンロードされた時に作成されます。[デプロイグループ ID]_last_successful_install ファイルは、ダウンロードしたファイルが最終送信先にコピーされたときに、このファイルの後に作成されます。サンプルファイルの名前は acde1916-9099-7caf-fd21-012345abcdef_most_recent_install です。
- deployment-logs には次のログファイルが含まれています。
  - デプロイがある日ごとに codedeploy-agent.yyyymmdd.log ファイルが作成されます。各ログファイルには、その日のデプロイに関する情報が含まれています。アクセス権限の問題などをデバッグする場合に、こうしたログファイルが役に立ちます。初期状態のログファイル名は codedeploy-agent.log です。翌日、デプロイの日付がファイル名に挿入されます。たとえば、今日の日付が 2018 年 1 月 3 日だとします。この場合、その日のデプロイすべてに関する情報は codedeploy-agent.log で見ることができます。そして翌日の 2018 年 1 月 4 日に、ログファイル名は codedeploy-agent.20180103.log に変更されます。
  - codedeploy-agent-deployments.log は各デプロイの scripts.log ファイルのコンテンツをコンパイルします。scripts.log ファイルは logs サブフォルダ (各 Deployment ID フォルダ内) にあります。このファイル内のエントリにはデプロイ ID が付いています。たとえば、"[d-ABCDEF123]LifecycleEvent - BeforeInstall" はデプロイ中に d-ABCDEF123 の ID を使用して書き込みを実行します。codedeploy-agent-deployments.log が最大サイズに達した場合、CodeDeploy エージェントは古いコンテンツを削除しながら引き続き書き込みを実行します。

CodeDeploy エージェントのオペレーションの管理

このセクションの手順では、CodeDeploy エージェントをインストール、アンインストール、再インストール、または更新する方法、および CodeDeploy エージェントが実行されていることを確認する方法について説明します。

> **Tip**
>
> このページの手順は、エージェントのバージョンによって異なります。インストールされているバージョンが不明な場合は、 `/opt/codedeploy-agent/bin/codedeploy-agent --version` (Linux または macOS) または `& 'C:\ProgramData\Amazon\CodeDeploy\bin\codedeploy-agent.exe' --version` (Windows) を実行します。そのコマンドが認識されない場合は、バージョン 1.8.x 以前を実行しています。詳細については、「CodeDeploy エージェントのバージョンを特定します。」を参照してください。

トピック

- CodeDeploy エージェントが実行されていることの確認
- CodeDeploy エージェントのバージョンを特定します。
- CodeDeploy エージェントをインストールする
- CodeDeploy エージェントを更新する
- CodeDeploy エージェントをアンインストールする
- CodeDeploy エージェントログを CloudWatch に送信する

CodeDeploy エージェントが実行されていることの確認

このセクションでは、インスタンスで CodeDeploy エージェントが実行を停止した可能性がある場合に実行するコマンドについて説明します。

トピック

- Amazon Linux または RHEL 用の CodeDeploy エージェントが実行されていることを確認します。
- Ubuntu サーバーの CodeDeploy エージェントが実行されていることを確認します。
- Windows サーバーの CodeDeploy エージェントが実行されていることを確認します。
- macOS 用の CodeDeploy エージェントが実行されていることを確認します。

Amazon Linux または RHEL 用の CodeDeploy エージェントが実行されていることを確認します。

CodeDeploy エージェントがインストールされていて実行していることを確認するには、インスタンスにサインインし、次のコマンドを実行します。

```
systemctl status codedeploy-agent
```

コマンドがエラーを返す場合、CodeDeploy エージェントはインストールされていません。「Amazon Linux または RHEL 用の CodeDeploy エージェントをインストールする」で説明されているようにインストールします。

CodeDeploy エージェントがインストールされて実行されている場合は、 `The AWS CodeDeploy agent is running.` (終了コード 0) のようなメッセージが表示されます。

(`The AWS CodeDeploy agent is not running.`終了コード 3) のようなメッセージが表示された場合は、サービスを起動し、次の 2 つのコマンドを一度に 1 つずつ実行します。

```
systemctl start codedeploy-agent
```

```
systemctl status codedeploy-agent
```

バージョン 2.0.x 以降では、`codedeploy-agent status`コマンドは LSB 互換の終了コード `0` (実行中)、 `3` (非実行中)、 `4` (不明) を返します。を使用する場合`systemctl status`、単位の状態が表示されます (例: `Active: active (running)`)。

Ubuntu サーバーの CodeDeploy エージェントが実行されていることを確認します。

CodeDeploy エージェントがインストールされていて実行していることを確認するには、インスタンスにサインインし、次のコマンドを実行します。

```
systemctl status codedeploy-agent
```

コマンドがエラーを返す場合、CodeDeploy エージェントはインストールされていません。「Ubuntu サーバー用の CodeDeploy エージェントをインストールする」で説明されているようにインストールします。

CodeDeploy エージェントがインストールされて実行されている場合は、 `The AWS CodeDeploy agent is running.` (終了コード 0) のようなメッセージが表示されます。

(`The AWS CodeDeploy agent is not running.`終了コード 3) のようなメッセージが表示された場合は、サービスを起動し、次の 2 つのコマンドを一度に 1 つずつ実行します。

```
systemctl start codedeploy-agent
```

```
systemctl status codedeploy-agent
```

Windows サーバーの CodeDeploy エージェントが実行されていることを確認します。

CodeDeploy エージェントがインストールされていて実行していることを確認するには、インスタンスにサインインし、次のコマンドを実行します。

```
powershell.exe -Command Get-Service -Name codedeployagent
```

次のような出力が表示されます:

```
Status   Name               DisplayName
------   ----               -----------
Running codedeployagent    CodeDeploy Host Agent Service
```

コマンドがエラーを返す場合、CodeDeploy エージェントはインストールされていません。「Windows サーバー用の CodeDeploy エージェントです。」で説明されているようにインストールします。

`Status` で `Running` 以外の何かが表示される場合は、次のコマンドでサービスを開始します。

```
powershell.exe -Command Start-Service -Name codedeployagent
```

次のコマンドを使ってサービスを再起動できます。

```
powershell.exe -Command Restart-Service -Name codedeployagent
```

次のコマンドを使ってサービスを停止できます。

```
powershell.exe -Command Stop-Service -Name codedeployagent
```

macOS 用の CodeDeploy エージェントが実行されていることを確認します。

CodeDeploy エージェントがインストールされていて実行していることを確認するには、インスタンスにサインインし、次のコマンドを実行します。

```
sudo /opt/codedeploy-agent/bin/codedeploy-agent status
```

ファイルが見つからないためにコマンドがエラーを返した場合、CodeDeploy エージェントはインストールされません。「macOS 用の CodeDeploy エージェントをインストールする」で説明されているようにインストールします。

CodeDeploy エージェントがインストールされて実行されている場合は、 `The AWS CodeDeploy agent is running.` (終了コード 0) のようなメッセージが表示されます。

(`The AWS CodeDeploy agent is not running.`終了コード 3) のようなメッセージが表示された場合は、次のコマンドを使用してエージェントを起動します。

```
sudo launchctl kickstart system/com.amazon.aws.codedeploy-agent
```

そのコマンドがサービスが見つからなかったと報告した場合、起動されたデーモンはロードされません。ロードして、次のコマンドで開始します。

```
sudo launchctl bootstrap system /Library/LaunchDaemons/com.amazon.aws.codedeploy-agent.plist
```

起動されたデーモンの状態を確認するには、以下を実行します。

```
sudo launchctl print system/com.amazon.aws.codedeploy-agent
```

CodeDeploy エージェントのバージョンを特定します。

インスタンスで実行中の CodeDeploy エージェントのバージョンは、2 つの方法で特定できます。

まず、バージョン 2.0.x 以降では、すべてのプラットフォームで `codedeploy-agent --version` コマンドを使用します。

次に、インスタンスの `.version` ファイルでバージョンを表示できます。Amazon Linux、RHEL、Ubuntu Server、macOS では、ファイルは にあり`/opt/codedeploy-agent/.version`、Windows Server では にあります`C:\ProgramData\Amazon\CodeDeploy\.version`。ファイルの内容はバージョンによって異なります。

- バージョン 2.0.x 以降では、ファイルには 形式のペアバージョン番号が含まれています`agent_version: 2.0.0`。この値は、エージェントが CodeDeploy サービスにレポートするバージョンと一致します。
- バージョン 1.8.x 以前では、バージョン文字列は 形式 `OFFICIAL_1.0.1.854_rpm` (または `_deb`、) を使用します`_msi`。

トピック

- Amazon Linux または RHEL でバージョンを確認する
- Ubuntu サーバーでバージョンを確認する
- Windows サーバーでバージョンをを確認する
- macOS でバージョンを確認する
- で利用可能な最新のエージェントバージョンを確認する AWS リージョン

Amazon Linux または RHEL でバージョンを確認する

インスタンスにサインインし、次のコマンドを実行します。

```
sudo /opt/codedeploy-agent/bin/codedeploy-agent --version
```

2.0.0 より前のバージョンでは、以下を使用することもできます。

```
sudo yum info codedeploy-agent
```

Ubuntu サーバーでバージョンを確認する

インスタンスにサインインし、次のコマンドを実行します。

```
sudo /opt/codedeploy-agent/bin/codedeploy-agent --version
```

2.0.0 より前のバージョンでは、以下を使用することもできます。

```
sudo dpkg -s codedeploy-agent
```

Windows サーバーでバージョンをを確認する

インスタンスにサインインし、次のコマンドを実行します。

```
& 'C:\ProgramData\Amazon\CodeDeploy\bin\codedeploy-agent.exe' --version
```

macOS でバージョンを確認する

インスタンスにサインインし、次のコマンドを実行します。

```
sudo /opt/codedeploy-agent/bin/codedeploy-agent --version
```

インストールされたパッケージのバージョンも確認できます。

```
pkgutil --pkg-info com.amazon.aws.codedeploy-agent
```

で利用可能な最新のエージェントバージョンを確認する AWS リージョン

AWS リージョン デプロイ先の で利用可能な最新バージョンを確認するには、そのリージョンの CodeDeploy リソースキットバケットをクエリします。各リージョンバケットには、そのリージョンで現在リリースされているエージェントパッケージに名前を付けるlatestv2/LATEST_VERSIONファイルがあります。ファイルはパブリックに読み取れるため、取得するための認証情報は必要ありません。

```
curl https://bucket-name.s3.region-identifier.amazonaws.com/latestv2/LATEST_VERSION
```

bucket-name と *region-identifier* を、デプロイ先のリージョンのリソースキットバケットとリージョン識別子に置き換えます。リストについては、「リージョン別リソースキットバケット名」を参照してください。

このコマンドにより、以下のような出力が返されます。

```
{
    "rpm_x86_64": "releases/codedeploy-agent-2.1.0-9745926189.x86_64.rpm",
    "rpm_aarch64": "releases/codedeploy-agent-2.1.0-9745926189.aarch64.rpm",
    "deb_x86_64": "releases/codedeploy-agent_2.1.0-9745926189_amd64.deb",
    "deb_aarch64": "releases/codedeploy-agent_2.1.0-9745926189_arm64.deb",
    "msi": "releases/codedeploy-agent-2.1.0-9745926189.msi"
}
```

各キーはパッケージタイプに名前を付け、RPM および DEB の場合はインスタンスアーキテクチャにも名前を付けます。各値は、そのパッケージの Amazon S3 オブジェクトキーであり、エージェントのバージョンとビルド番号はファイル名の一部であり、形式は です*agent-version*-*build-number*。前述の出力では、 リージョンで利用可能な最新バージョンは 2.1.0 です。

バージョン 2.1.2 以降では、ファイルには macOS インストーラパッケージという名前のpkg_aarch64キーもあります。

このファイル名をインストールまたはバージョンに更新するには、CodeDeploy エージェントをインストールする「」および「」を参照してくださいCodeDeploy エージェントを更新する。各バージョンに含まれる機能については、「」を参照してくださいCodeDeploy エージェントのバージョン履歴。

CodeDeploy エージェントをインストールする

EC2 インスタンスまたはオンプレミスサーバーで CodeDeploy を使用するには、まず CodeDeploy エージェントをインストールする必要があります。CodeDeploy エージェントは、 AWS Systems Manager または コマンドラインを使用してインストールできます。Systems Manager のさらなる詳細については、「AWS Systems Managerとは」 を参照してください。

インストールする推奨バージョンについては、「CodeDeploy エージェントのバージョン履歴」を参照してください。

トピック

- を使用して CodeDeploy エージェントをインストールする AWS Systems Manager
- コマンドラインを使用して CodeDeploy エージェントをインストールする

を使用して CodeDeploy エージェントをインストールする AWS Systems Manager

AWS マネジメントコンソール または を使用して AWS CLI 、CodeDeploy エージェントを Amazon EC2 またはオンプレミスインスタンスにインストールできます AWS Systems Manager。特定のバージョンをインストールするか、常に最新バージョンのエージェントをインストールするかを選択できます。の詳細については AWS Systems Manager、「 とは AWS Systems Manager」を参照してください。

また、Amazon S3 バケットか　らCodeDeploy エージェントをインストールすることもできます。Amazon S3 ダウンロードリンクの使用については、「 コマンドラインを使用して CodeDeploy エージェントをインストールする 」を参照してください。

トピック

- 前提条件
- CodeDeploy エージェントをインストールする

前提条件

CodeDeploy の開始方法 のステップに従って、IAM アクセス許可と AWS CLIを設定します。

Systems Manager を使用してオンプレミスサーバーに CodeDeploy エージェントをインストールする場合は、オンプレミスサーバーを Amazon EC2 Systems Manager に登録する必要があります。詳細については、AWS Systems Manager ユーザーガイド の ハイブリッド環境での Systems Manager のセットアップ を参照してください。

CodeDeploy エージェントをインストールする

Systems Manager を使用して CodeDeploy エージェントをインストールする前に、Systems Manager に合わせてインスタンスが適切に設定されていることを確認する必要があります。

SSM Agent のインストールまたは更新

Amazon EC2 インスタンスでは、CodeDeploy エージェントを使用するにはバージョン 2.3.274.0 以降のインスタンスが実行されている必要があります。CodeDeploy エージェントをインストールする前に、SSM Agent を更新するか、インスタンスにインストールしてください (まだの場合)。

SSM エージェントは、 が提供する一部の Amazon EC2 AMIs にプリインストールされています AWS。詳細については、「SSM Agent がプリインストールされた Amazon Machine Images (AMIs)」を参照してください。

> **Note**
>
> インスタンスのオペレーティング システムも CodeDeploy エージェントでサポートされていることを確認します。詳細については、「CodeDeploy エージェントで対応するオペレーティングシステム」を参照してください。

Linux を実行しているインスタンスでの SSM Agent のインストールまたは更新については、AWS Systems Manager ユーザーガイド の Linux インスタンスでの SSM Agent のインストールおよび設定 を参照してください。

Windows Server を実行しているインスタンスでの SSM Agent のインストールまたは更新については、AWS Systems Manager ユーザーガイド の Windows インスタンスでの SSM Agent のインストールおよび設定 を参照してください。

EC2 Mac インスタンスに SSM エージェントをインストールまたは更新する方法については、AWS Systems Manager 「 ユーザーガイド」のmacOS 用の EC2 インスタンスに SSM エージェントを手動でインストールおよびアンインストールする」を参照してください。

(オプション) Systems Manager の前提条件を確認します。

Systems Manager Run Command を使用して CodeDeploy エージェントをインストールする前に、インスタンスが Systems Manager の最低要件を満たしていることを確認してください。詳細については、AWS Systems Managerユーザーガイド の「AWS Systems Manager のセットアップ」を参照してください。

CodeDeploy エージェントをインストールする

SSM では、CodeDeploy を一度インストールするか、新しいバージョンをインストールするスケジュールを設定できます。

CodeDeploy エージェントをインストールするには、必要なエージェントバージョンの Systems Manager Distributor パッケージを選択し、AWS Systems Manager 「Distributor でパッケージをインストールまたは更新する」のステップに従います。パッケージはエージェントバージョンによって異なります。

- バージョン 2.0.x 以降では、`AWSCodeDeployAgentV2`パッケージを選択します。これは、バージョン 2.1.2 以降の macOS をサポートする唯一のパッケージです。
- バージョン 1.8.x 以前では、`AWSCodeDeployAgent`パッケージを選択します。

コマンドラインを使用して CodeDeploy エージェントをインストールする

以下のトピックに従い、コマンドラインを使用して CodeDeploy エージェントをインストールおよび実行します。

トピック

- Amazon Linux または RHEL 用の CodeDeploy エージェントをインストールする
- Ubuntu サーバー用の CodeDeploy エージェントをインストールする
- Windows サーバー用の CodeDeploy エージェントです。
- macOS 用の CodeDeploy エージェントをインストールする

Amazon Linux または RHEL 用の CodeDeploy エージェントをインストールする

インスタンスにサインインし、次のコマンドを一度に 1 つずつ実行します。コマンド `sudo yum update` を最初に実行するのが、`yum` を使用してパッケージをインストールするときのベストプラクティスと考えられていますが、すべてのパッケージを更新しない場合はこのコマンドをスキップできます。

```
sudo yum update
```

バージョン 2.0.x 以降の場合:

```
sudo yum install wget
```

バージョン 1.8.x 以前の場合:

```
sudo yum install ruby wget
```

(オプション) 以前のエージェントキャッシュ情報の AMI を消去するには、次のスクリプトを実行します。

```
#!/bin/bash
CODEDEPLOY_BIN="/opt/codedeploy-agent/bin/codedeploy-agent"
$CODEDEPLOY_BIN stop
yum erase codedeploy-agent -y
```

ホームディレクトリに移動します。

```
cd /home/ec2-user
```

> **Note**
>
> 以前のコマンドで、/home/ec2-user は、Amazon Linux または RHEL Amazon EC2 インスタンスのデフォルトのユーザー名を表しています。インスタンスがカスタム AMI を使用して作成された場合、AMI 所有者は別のデフォルトのユーザー名を指定している可能性があります。

CodeDeploy エージェントインストーラをダウンロードします。latestv2/ プレフィックスはバージョン 2.0.x インストーラーを提供し、 latest/ プレフィックスはバージョン 1.8.x インストーラーを提供します。

バージョン 2.0.x 以降の場合:

```
wget https://bucket-name.s3.region-identifier.amazonaws.com/latestv2/install
```

バージョン 1.8.x 以前の場合:

```
wget https://bucket-name.s3.region-identifier.amazonaws.com/latest/install
```

bucket-name は、お住まいの地域用の CodeDeploy リソースキットファイルが含まれている Amazon S3 バケットの名前です。*########* は、お住まいの地域の識別子です。

たとえば、バージョン 2.0.x 以降の場合:

```
https://aws-codedeploy-us-east-2.s3.us-east-2.amazonaws.com/latestv2/
install
```

バージョン 1.8.x 以前の場合:

```
https://aws-codedeploy-us-east-2.s3.us-east-2.amazonaws.com/latest/install
```

バケット名とリージョン識別子のリストについては、「リージョン別リソースキットバケット名」を参照してください。

install ファイルに実行権限を設定します。

```
chmod +x ./install
```

CodeDeploy エージェントの最新バージョンをインストールするには:

-
```
sudo ./install auto
```

CodeDeploy エージェントの特定のバージョンをインストールするには:

- リージョンで使用可能なバージョンを一覧表示します。

```
aws s3 ls s3://aws-codedeploy-region-identifier/releases/ --region region-identifier
 | grep '\.rpm$'
```

- 以下のいずれかのバージョンをインストールします。

  バージョン 2.0.x 以降では、パッケージはアーキテクチャ固有です。

```
sudo ./install auto -v releases/codedeploy-agent-version.arch.rpm
```

  インスタンスアーキテクチャaarch64に応じて、*arch* は x86_64または です。

  バージョン 1.8.x 以前では、パッケージは を使用しますnoarch。

```
sudo ./install auto -v releases/codedeploy-agent-version.noarch.rpm
```

最新のリリースバージョンについては、「」を参照してくださいCodeDeploy エージェントのバージョン履歴。

サービスが実行されているかどうか確認するには、次のコマンドを実行します。

```
systemctl status codedeploy-agent
```

バージョン 2.0.x 以降では、エージェントが実行中の場合、次のようなメッセージが表示されます。 `The AWS CodeDeploy agent is running.`

バージョン 1.8.x 以前では、エージェントが実行されている場合、次のようなメッセージが表示されます。 `The AWS CodeDeploy agent is running.`

エージェントが実行されていない場合は、サービスを開始し、ステータスを確認します。

```
systemctl start codedeploy-agent
```

```
systemctl status codedeploy-agent
```

Ubuntu サーバー用の CodeDeploy エージェントをインストールする

CodeDeploy エージェントを Ubuntu サーバーにインストールするには

1. インスタンスにサインインします。
2. 次のコマンドを順々に入力します。

   ```
   sudo apt update
   ```

   バージョン 2.0.x 以降の場合:

   ```
   sudo apt install wget
   ```

   バージョン 1.8.x 以前の場合:

   ```
   sudo apt install ruby-full wget
   ```

3. 次のコマンドを入力します。

```
cd /home/ubuntu
```

/home/ubuntu は、Ubuntu Server インスタンスのデフォルトのユーザー名を表しています。インスタンスがカスタム AMI を使用して作成された場合、AMI 所有者は別のデフォルトのユーザー名を指定している可能性があります。

4. 次のコマンドを入力します。latestv2/ プレフィックスはバージョン 2.0.x インストーラーを提供し、 latest/ プレフィックスはバージョン 1.8.x インストーラーを提供します。

   バージョン 2.0.x 以降の場合:

   ```
   wget https://bucket-name.s3.region-identifier.amazonaws.com/latestv2/install
   ```

   バージョン 1.8.x 以前の場合:

   ```
   wget https://bucket-name.s3.region-identifier.amazonaws.com/latest/install
   ```

   bucket-name は、お住まいの地域用の CodeDeploy リソースキットファイルが含まれている Amazon S3 バケットの名前です。*######## * は、お住まいの地域の識別子です。

   たとえば、バージョン 2.0.x 以降の場合:

   ```
   https://aws-codedeploy-us-east-2.s3.us-east-2.amazonaws.com/latestv2/
   install
   ```

   バージョン 1.8.x 以前の場合:

   ```
   https://aws-codedeploy-us-east-2.s3.us-east-2.amazonaws.com/latest/
   install
   ```

   バケット名とリージョン識別子のリストについては、「リージョン別リソースキットバケット名」を参照してください。

5. 次のコマンドを入力します。

   ```
   chmod +x ./install
   ```

6. 次のいずれかを行います:

   - CodeDeploy エージェントの最新バージョンをインストールするには:

```
sudo ./install auto
```

- CodeDeploy エージェントの特定のバージョンをインストールするには:
  - リージョンで使用可能なバージョンを一覧表示します。

    ```
    aws s3 ls s3://aws-codedeploy-region-identifier/releases/ --region region-identifier | grep '\.deb$'
    ```

  - 以下のいずれかのバージョンをインストールします。

    バージョン 2.0.x 以降では、パッケージはアーキテクチャ固有です。

    ```
    sudo ./install auto -v releases/codedeploy-agent_version_arch.deb
    ```

    インスタンスアーキテクチャarm64に応じて、*arch* は amd64または です。

    バージョン 1.8.x 以前では、パッケージは別の命名規則を使用します。

    ```
    sudo ./install auto -v releases/codedeploy-agent_version_all.deb
    ```

    最新のリリースバージョンについては、「」を参照してくださいCodeDeploy エージェントのバージョン履歴。

サービスが実行されていることをチェックするには

1. 次のコマンドを入力します。

   ```
   systemctl status codedeploy-agent
   ```

   CodeDeploy エージェントがインストールされて実行されている場合:

   バージョン 2.0.x 以降では、以下が表示されます。 `The AWS CodeDeploy agent is running.`

   バージョン 1.8.x 以前では、以下が表示されます。 `The AWS CodeDeploy agent is running.`
2. エージェントが実行されていない場合は、サービスを開始し、ステータスを確認します。

```
systemctl start codedeploy-agent
```

```
systemctl status codedeploy-agent
```

Windows サーバー用の CodeDeploy エージェントです。

Windows サーバーのインスタンスでは、次のいずれかの方法を使用して、CodeDeploy エージェントをダウンロードしてインストールできます。

- 使用 AWS Systems Manager (推奨)
- 一連の Windows PowerShell コマンドを実行。
- 直接ダウンロードリンクを選択。
- Amazon S3 コピーコマンドを実行してください。

> **Note**
>
> CodeDeploy エージェントがインストールされているフォルダは `C:\ProgramData\Amazon\CodeDeploy` です。このパスにディレクトリジャンクションまたはシンボリックリンクを配置しないでください。バージョン 2.0.x 以降では、エージェントはこの要件を適用し、サービスバイナリが に含まれていない場合は開始を拒否しますC:\ProgramData\Amazon\CodeDeploy\。

トピック

- 使用アイテム Systems Manager
- Windows PowerShell の使用
- 直接接続の使用
- Amazon S3 コピーコマンドの使用

使用アイテム Systems Manager

「を使用して CodeDeploy エージェントをインストールする AWS Systems Manager」の手順に従って CodeDeploy エージェントをインストールします。

Windows PowerShell の使用

インスタンスにサインインし、Windows PowerShell で次のコマンドを実行します。

1. インターネットからダウンロードされたすべてのスクリプトと設定ファイルが、信頼された発行元によって署名されていることを要求します。実行ポリシーの変更を求められた場合は「**Y**」と入力します。

   ```
   Set-ExecutionPolicy RemoteSigned
   ```

2. をロードします AWS Tools for Windows PowerShell。

   ```
   Import-Module AWSPowerShell
   ```

3. CodeDeploy エージェントのインストールファイルがダウンロードされた場所で、ディレクトリを作成します。

   ```
   New-Item -Path "c:\temp" -ItemType "directory" -Force
   ```

4. `Set-AWSCredential` および `Initialize-AWSDefaultConfiguration` コマンドを使用して AWS 認証情報を設定します。詳細については、AWS Tools for PowerShell ユーザーガイドにある「AWS 認証情報の使用」を参照してください。

5. CodeDeploy エージェントのインストールファイルをダウンロードします。

   > **Note**
   >
   > 最新のリリースバージョンについては、「」を参照してくださいCodeDeploy エージェントのバージョン履歴。

   CodeDeploy エージェントの最新バージョンをインストールするには。`latestv2/` プレフィックスはバージョン 2.x (2.0.x 以降) インストーラーを提供し、`latest/`プレフィックスはバージョン 1.8.x インストーラーを提供します。

   - バージョン 2.0.x 以降の場合:

     ```
     powershell.exe -Command Read-S3Object -BucketName bucket-name -Key latestv2/
     codedeploy-agent.msi -File c:\temp\codedeploy-agent.msi
     ```

バージョン 1.8.x 以前の場合:

```
powershell.exe -Command Read-S3Object -BucketName bucket-name -Key latest/
codedeploy-agent.msi -File c:\temp\codedeploy-agent.msi
```

CodeDeploy エージェントの特定のバージョンをインストールするには:

-
```
powershell.exe -Command Read-S3Object -BucketName bucket-name -Key releases/
codedeploy-agent-###.msi -File c:\temp\codedeploy-agent.msi
```

は、リージョンの CodeDeploy リソースキットファイルが含まれている Amazon S3 バケットの名前です。例えば、米国東部 (オハイオ) リージョンの場合、***#####*** を aws-codedeploy-us-east-2 に置き換えます。バケット名のリストについては、「リージョン別リソースキットバケット名」を参照してください。

6. CodeDeploy エージェントのインストールファイルを実行します。

```
c:\temp\codedeploy-agent.msi /quiet /l c:\temp\host-agent-install-log.txt
```

サービスが実行されているかどうか確認するには、次のコマンドを実行します。

```
powershell.exe -Command Get-Service -Name codedeployagent
```

CodeDeploy エージェントがインストールされただけで、まだ開始されていない場合は、Get-Service コマンドを実行すると、[ステータス] に「**Start...**」と表示されます。

```
Status     Name               DisplayName
------     ----               -----------
Start...   codedeployagent    CodeDeploy Host Agent Service
```

CodeDeploy エージェントが既に実行されている場合は、Get-Service コマンドを実行すると、[ステータス] に「**Running**」と表示されます。

```
Status     Name               DisplayName
```

```
------     ----                 -----------
Running    codedeployagent    CodeDeploy Host Agent Service
```

直接接続の使用

Windows サーバーのインスタンスのブラウザのセキュリティ設定でアクセス権限 (例えば、`https://s3.*.amazonaws.com` への) が提供されている場合は、リージョンの直接接続を使用して CodeDeploy エージェントをダウンロードし、インストーラを手動で実行できます。

`latestv2/` プレフィックスはバージョン 2.x (2.0.x 以降) インストーラーを提供し、`latest/`プレフィックスはバージョン 1.8.x インストーラーを提供します。

バージョン 2.0.x 以降では、リンクは次のとおりです。

```
https://s3.region.amazonaws.com/aws-codedeploy-region/latestv2/codedeploy-agent.msi
```

バージョン 1.8.x 以前の場合:

```
https://s3.region.amazonaws.com/aws-codedeploy-region/latest/codedeploy-agent.msi
```

*...#####*はアプリケーションをデプロイする AWS リージョンです。

たとえば、バージョン 2.0.x 以降の場合:

```
https://s3.af-south-1.amazonaws.com/aws-codedeploy-af-south-1/latestv2/codedeploy-agent.msi
```

> **Important**
>
> CodeDeploy アプリケーションと同じリージョンから `.msi` ファイルを取得します。別のリージョンを選択すると、`.msi` ファイルを実行したときに `codedeploy-agent-log` ファイルに `inconsistent region` 障害が発生する可能性があります。

Amazon S3 コピーコマンドの使用

AWS CLI がインスタンスにインストールされている場合は、Amazon S3 cp コマンドを使用して CodeDeploy エージェントをダウンロードし、インストーラを手動で実行できます。詳細について

は、「Microsoft Windows AWS Command Line Interface に をインストールする」を参照してください。

latestv2/ プレフィックスはバージョン 2.x (2.0.x 以降) インストーラーを提供し、latest/プレフィックスはバージョン 1.8.x インストーラーを提供します。

バージョン 2.0.x 以降では、Amazon S3 コマンドは次のとおりです。

```
aws s3 cp s3://aws-codedeploy-region/latestv2/codedeploy-agent.msi codedeploy-agent.msi
 --region region
```

バージョン 1.8.x 以前の場合:

```
aws s3 cp s3://aws-codedeploy-region/latest/codedeploy-agent.msi codedeploy-agent.msi
 --region region
```

...*#####*はアプリケーションをデプロイする AWS リージョンです。

たとえば、バージョン 2.0.x 以降の場合:

```
aws s3 cp s3://aws-codedeploy-af-south-1/latestv2/codedeploy-agent.msi codedeploy-
agent.msi --region af-south-1
```

macOS 用の CodeDeploy エージェントをインストールする

インスタンスにサインインし、次のコマンドを一度に 1 つずつ実行します。

ホームディレクトリに移動します。

```
cd ~
```

CodeDeploy エージェントインストーラパッケージをダウンロードします。

```
curl -O https://bucket-name.s3.region-identifier.amazonaws.com/latestv2/codedeploy-
agent.aarch64.pkg
```

bucket-name は、お住まいの地域用の CodeDeploy リソースキットファイルが含まれている Amazon S3 バケットの名前です。*########* は、お住まいの地域の識別子です。

例えば、次のようになります。

`https://aws-codedeploy-us-east-2.s3.us-east-2.amazonaws.com/latestv2/codedeploy-agent.aarch64.pkg`

バケット名とリージョン識別子のリストについては、「リージョン別リソースキットバケット名」を参照してください。

CodeDeploy エージェントの最新バージョンをインストールするには:

- ```
  sudo installer -pkg codedeploy-agent.aarch64.pkg -target /
  ```

CodeDeploy エージェントの特定のバージョンをインストールするには:

- リージョンで使用可能なバージョンを一覧表示します。

  ```
  aws s3 ls s3://aws-codedeploy-region-identifier/releases/ --region region-identifier
   | grep '\.pkg$'
  ```

- 次のいずれかのバージョンをダウンロードしてインストールします。

  ```
  aws s3 cp s3://aws-codedeploy-region-identifier/releases/codedeploy-
  agent-version.aarch64.pkg codedeploy-agent.aarch64.pkg --region region-identifier
  ```

  ```
  sudo installer -pkg codedeploy-agent.aarch64.pkg -target /
  ```

  最新のリリースバージョンについては、「」を参照してくださいCodeDeploy エージェントのバージョン履歴。

サービスが実行されているかどうか確認するには、次のコマンドを実行します。

```
sudo /opt/codedeploy-agent/bin/codedeploy-agent status
```

エージェントが実行中の場合は、次のようなメッセージが表示されます。 `The AWS CodeDeploy agent is running.`

エージェントが実行されていない場合は、サービスを開始し、ステータスを確認します。

```
sudo launchctl kickstart system/com.amazon.aws.codedeploy-agent
```
```

```
sudo /opt/codedeploy-agent/bin/codedeploy-agent status
```

CodeDeploy エージェントを更新する

AWS Systems Managerを使用して、すべてのサポートされているオペレーティングシステムで CodeDeploy エージェントの自動更新スケジュールを設定できます。また、インスタンスでコマンドを実行して、サポートされているすべてのオペレーティングシステムで更新を強制することもできます。

トピック

- Amazon Linux または RHEL で CodeDeploy エージェントを更新する
- Ubuntu サーバーで CodeDeploy エージェントを更新する
- Windows サーバーで CodeDeploy エージェントを更新する
- macOS で CodeDeploy エージェントを更新する

Amazon Linux または RHEL で CodeDeploy エージェントを更新する

を使用して CodeDeploy エージェントの自動スケジュール更新を設定するには、「 を使用して CodeDeploy エージェントをインストールする AWS Systems Manager」の手順 AWS Systems Managerに従います。

バージョン 2.0.x 以降では、更新を強制的に実行するには、以下を実行します。

```
sudo /opt/codedeploy-agent/bin/codedeploy-agent update
```

または、リージョン`latest/v2/install`スクリプトを再ダウンロードして実行します。インストール手順については、「Amazon Linux または RHEL 用の CodeDeploy エージェントをインストールする」を参照してください。

バージョン 1.8.x 以前では、更新を強制するには、以下を実行します。

```
sudo /opt/codedeploy-agent/bin/install auto
```

> **Important**
>
> 現在、バージョン 1.8.x から 2.0.0 への自動更新パスは存在しません。2.0.0 にアップグレードするには、`AWSCodeDeployAgentV2`Systems Manager Distributor パッケージでインス

トールするか、リージョンlatestv2/installスクリプトを手動で実行します。手順については、「CodeDeploy エージェントをインストールする」を参照してください。

Ubuntu サーバーで CodeDeploy エージェントを更新する

を使用して CodeDeploy エージェントの自動スケジュール更新を設定するには、「 を使用して CodeDeploy エージェントをインストールする AWS Systems Manager」の手順 AWS Systems Managerに従います。

バージョン 2.0.x 以降では、更新を強制的に実行するには、以下を実行します。

```
sudo /opt/codedeploy-agent/bin/codedeploy-agent update
```

または、リージョンlatestv2/installスクリプトを再ダウンロードして実行します。インストール手順については、「Ubuntu サーバー用の CodeDeploy エージェントをインストールする」を参照してください。

バージョン 1.8.x 以前では、更新を強制するには、以下を実行します。

```
sudo /opt/codedeploy-agent/bin/install auto
```

Important

現在、バージョン 1.8.x から 2.0.0 への自動更新パスは存在しません。2.0.0 にアップグレードするには、AWSCodeDeployAgentV2Systems Manager Distributor パッケージでインストールするか、リージョンlatestv2/installスクリプトを手動で実行します。手順については、「CodeDeploy エージェントをインストールする」を参照してください。

Windows サーバーで CodeDeploy エージェントを更新する

CodeDeploy エージェントの自動更新は、 を使用して有効にできます AWS Systems Manager。Systems Manager では、Systems Manager State Manager との関連付けを作成して、Amazon EC2 またはオンプレミスインスタンスの更新スケジュールを設定できます。現在のバージョンをアンインストールして新しいバージョンをインストールすることで、CodeDeploy エージェントを手動で更新することもできます。

トピック

- を使用して CodeDeploy エージェントの自動更新を設定する AWS Systems Manager
- CodeDeploy エージェントを手動で更新する

を使用して CodeDeploy エージェントの自動更新を設定する AWS Systems Manager

Systems Manager を設定し、CodeDeploy エージェントの自動更新を有効にするには、「 を使用して CodeDeploy エージェントをインストールする AWS Systems Manager」の手順に従います。

CodeDeploy エージェントを手動で更新する

CodeDeploy エージェントを手動で更新するには、CLI または Systems Manager を使用して最新バージョンをインストールします。「CodeDeploy エージェントをインストールする」の手順に従います。CodeDeploy エージェントのアンインストール の手順に従って、古いバージョンの CodeDeploy エージェントをアンインストールすることをお勧めします。

macOS で CodeDeploy エージェントを更新する

を使用して CodeDeploy エージェントの自動スケジュール更新を設定するには、 AWS Systems Manager「」の手順に従ってAWSCodeDeployAgentV2パッケージを使用して CodeDeploy エージェントをインストールする AWS Systems Managerを選択します。

更新を強制するには、以下を実行します。

```
sudo /opt/codedeploy-agent/bin/codedeploy-agent update
```

または、最新の.pkgファイルをダウンロードし、既存のエージェントにインストールします。インストール手順については、「macOS 用の CodeDeploy エージェントをインストールする」を参照してください。

CodeDeploy エージェントをアンインストールする

不要になった場合や、新しいインストールを実行する場合は、インスタンスから CodeDeploy エージェントを削除できます。

Amazon Linux または RHEL から CodeDeploy エージェントをアンインストールする

CodeDeploy エージェントをアンインストールするには、インスタンスにサインインし、次のコマンドを実行します。

```
sudo yum erase codedeploy-agent
```

Ubuntu Server から CodeDeploy エージェントをアンインストール

CodeDeploy エージェントをアンインストールするには、インスタンスにサインインし、次のコマンドを実行します。

```
sudo dpkg --purge codedeploy-agent
```

Windows サーバーから CodeDeploy エージェントをアンインストールする

CodeDeploy エージェントをアンインストールするには、インスタンスにサインインし、次の 3 つのコマンドを 1 つずつ実行します。

```
wmic
product where name="CodeDeploy Host Agent" call uninstall /nointeractive
exit
```

> **Note**
>
> Windows Server 2025 以降では、 wmicは廃止されました。代わりに次の PowerShell コマンドを使用します。
>
> ```
> Get-ItemProperty 'HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall*' |
>   Where-Object DisplayName -eq 'CodeDeploy Host Agent' |
>   ForEach-Object { Start-Process msiexec.exe -ArgumentList "/x
>  $($_.PSChildName) /quiet" -Wait }
> ```

または、インスタンスにサインインし、[Control Panel (コントロールパネル)] で [Programs and Features (プログラムと機能)] を開き、[CodeDeploy Host Agent (CodeDeploy ホストエージェント)] を選択してから [Uninstall (アンインストール)] を選択します。

macOS から CodeDeploy エージェントをアンインストールする

CodeDeploy エージェントをアンインストールするには、インスタンスにサインインし、次のコマンドを実行します。

```
sudo /opt/codedeploy-agent/bin/uninstall.sh
```

CodeDeploy エージェントログを CloudWatch に送信する

CodeDeploy エージェントのメトリクスとログデータを CloudWatch に送信するには、統合した CloudWatch エージェント、またはより簡単に CloudWatch エージェントを使用します。

以下の手順に従って CloudWatch エージェントをインストールし、CodeDeploy エージェントで使用するように設定します。

前提条件

開始する前に、以下のタスクを完了します。

- CodeDeploy エージェントをインストールし、実行されていることを確認します。詳細については、「CodeDeploy エージェントをインストールする」および「CodeDeploy エージェントが実行されていることの確認」を参照してください。
- CloudWatch エージェントをインストールします。詳細については、「CloudWatch エージェントのインストール」を参照してください。
- 次のアクセス権限を CodeDeploy IAM インスタンスプロファイルに追加します。
  - CloudWatchLogsFullAccess
  - CloudWatchAgentServerPolicy

  CodeDeploy インスタンスプロファイルの詳細については、CodeDeploy の開始方法 の「ステップ 4: Amazon EC2 インスタンス用の IAM インスタンスプロファイルを作成する」を参照してください。

CodeDeploy ログを収集するための CloudWatch エージェントの設定

CloudWatch エージェントは、ウィザードを使用するか、設定ファイルを手動で作成または編集することで設定できます。

ウィザードを使用して CloudWatch エージェントを設定する (Linux)

1. 「CloudWatch エージェント設定ウィザードを実行する」の説明に従ってウィザードを実行します。
2. ウィザードで、`Do you want to monitor any log files?` と表示されたら、**1** と入力します。
3. CodeDeploy エージェントのログファイルを次のように指定します。

1. `Log file path` には、CodeDeploy ログファイルのパス (例: **/var/log/aws/codedeploy-agent/codedeploy-agent.log**) を入力します。
2. `Log group name` には、ロググループ名 (例: **codedeploy-agent-log**) を入力します。
3. `Log stream name` には、ログストリーム名 (例: **{instance_id}-codedeploy-agent-log**) を入力します。

4. `Do you want to specify any additional log files?` と表示されたら、**1** と入力します。
5. CodeDeploy エージェントのデプロイログを次のように指定します。

   1. `Log file path` には、CodeDeploy デプロイログファイルのパス (例: **/opt/codedeploy-agent/deployment-root/deployment-logs/codedeploy-agent-deployments.log**) を入力します。
   2. `Log group name` には、ロググループ名 (例: **codedeploy-agent-deployment-log**) を入力します。
   3. `Log stream name` には、ログストリーム名 (例: **{instance_id}-codedeploy-agent-deployment-log**) を入力します。

6. `Do you want to specify any additional log files?` と表示されたら、**1** と入力します。
7. CodeDeploy エージェントのアップデータログを次のように指定します。

   1. に`Log file path`、CodeDeploy アップデーターログファイルのパスを入力します。パスはエージェントのバージョンによって異なります。
      - バージョン 2.0.x 以降では、 を使用します**/var/log/aws/codedeploy-agent/codedeploy-agent-updater.log**。
      - バージョン 1.8.x 以前では、 を使用します**/tmp/codedeploy-agent.update.log**。
   2. `Log group name` には、ロググループ名 (例: **codedeploy-agent-updater-log**) を入力します。
   3. `Log stream name` には、ログストリーム名 (例: **{instance_id}-codedeploy-agent-updater-log**) を入力します。

ウィザードを使用して CloudWatch エージェントを設定する (Windows)

1. 「CloudWatch エージェント設定ウィザードを実行する」の説明に従ってウィザードを実行します。

2. ウィザードで、`Do you want to monitor any customized log files?` と表示されたら、**1** と入力します。
3. CodeDeploy ログファイルを次のように指定します。
   1. に`Log file path`、CodeDeploy エージェントログファイルのパスを入力します。パスはエージェントのバージョンによって異なります。
      - バージョン 2.0.x 以降では、 を使用します**C:\ProgramData\Amazon\CodeDeploy\log\codedeploy-agent.log**。
      - バージョン 1.8.x 以前では、 を使用します**C:\ProgramData\Amazon\CodeDeploy\log\codedeploy-agent-log.txt**。
   2. `Log group name` には、ロググループ名 (例: **codedeploy-agent-log**) を入力します。
   3. `Log stream name` には、ログストリーム名 (例: **{instance_id}-codedeploy-agent-log**) を入力します。
4. `Do you want to specify any additional log files?` と表示されたら、**1** と入力します。
5. CodeDeploy エージェントのデプロイログを次のように指定します。
   1. には`Log file path`、CodeDeploy デプロイログファイルのパス を入力します**C:\ProgramData\Amazon\CodeDeploy\deployment-logs\codedeploy-agent-deployments.log**。
   2. `Log group name` には、ロググループ名 (例: **codedeploy-agent-deployment-log**) を入力します。
   3. `Log stream name` には、ログストリームの名前 (例: **{instance_id}-codedeploy-agent-deployment-log**) を入力します。

設定ファイルを手動で作成または編集して CloudWatch エージェントを設定する (Linux)

1. 「CloudWatch エージェント設定ファイルを手動で作成または編集する」の説明に従って CloudWatch エージェント設定ファイルを作成または編集します。
2. ファイル名は `/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json` で、次のコードが含まれていることを確認してください。

   アップデータログパスはバージョンによって異なります。バージョン 2.0.x 以降では、 を使用します`/var/log/aws/codedeploy-agent/codedeploy-agent-updater.log`。バージョン 1.8.x 以前では、 を使用します`/tmp/codedeploy-agent.update.log`。

```
...
"logs": {
    "logs_collected": {
        "files": {
            "collect_list": [
                {
                    "file_path": "/var/log/aws/codedeploy-agent/codedeploy-
agent.log",
                    "log_group_name": "codedeploy-agent-log",
                    "log_stream_name": "{instance_id}-agent-log"
                },
                {
                    "file_path": "/opt/codedeploy-agent/deployment-root/deployment-
logs/codedeploy-agent-deployments.log",
                    "log_group_name": "codedeploy-agent-deployment-log",
                    "log_stream_name": "{instance_id}-codedeploy-agent-deployment-
log"
                },
                {
                    "file_path": "/var/log/aws/codedeploy-agent/codedeploy-agent-
updater.log",
                    "log_group_name": "codedeploy-agent-updater-log",
                    "log_stream_name": "{instance_id}-codedeploy-agent-updater-log"
                }
            ]
        }
    }
}
...
```

設定ファイルを手動で作成または編集して CloudWatch エージェントを設定する (Windows)

1. 「CloudWatch エージェント設定ファイルを手動で作成または編集する」の説明に従って CloudWatch エージェント設定ファイルを作成または編集します。
2. ファイル名は `C:\ProgramData\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent.json` で、次のコードが含まれていることを確認してください。

   バージョン 2.0.x 以降では、エージェントログファイル名は ですcodedeploy-agent.log。バージョン 1.8.x 以前では、エージェントログファイル名は ですcodedeploy-agent-log.txt。デプロイログパスは、両方のバージョンで同じです。

```
...
"logs": {
        "logs_collected": {
            "files": {
                "collect_list": [
                    {
                        "file_path": "C:\\ProgramData\\Amazon\\CodeDeploy\\log\
\codedeploy-agent.log",
                        "log_group_name": "codedeploy-agent-log",
                        "log_stream_name": "{instance_id}-codedeploy-agent-log"
                    },
                    {
                        "file_path": "C:\\ProgramData\\Amazon\\CodeDeploy\
\deployment-logs\\codedeploy-agent-deployments.log",
                        "log_group_name": "codedeploy-agent-deployment-log",
                        "log_stream_name": "{instance_id}-codedeploy-agent-
deployment-log"
                    }
                ]
            },
            ...
        }
    },
...
```

CloudWatch エージェントを再起動する

変更を加えたら、「CloudWatch エージェントを起動する」の説明に従って CloudWatch エージェントを再起動します。

CodeDeploy のためにインスタンスを用いた操作

CodeDeploy は Amazon Linux、Ubuntu Server、Red Hat Enterprise Linux (RHEL)、Windows Server を実行しているインスタンスへのデプロイをサポートします。

CodeDeploy を使って、Amazon EC2 インスタンスとオンプレミスインスタンスの両方にデプロイできます。オンプレミスインスタンスは、CodeDeploy エージェントを実行してパブリック AWS サービスエンドポイントに接続できる Amazon EC2 インスタンスではない物理デバイスです。CodeDeploy を使って、クラウド中の Amazon EC2 インスタンスと、オフィスのデスクトップ PC や自身のデータセンターのサーバーに、アプリケーションを同時にデプロイできます。

Amazon EC2 インスタンスとオンプレミスインスタンスの比較

次の表は、Amazon EC2 インスタンスとオンプレミスインスタンスの比較を示しています。

件名	Amazon EC2 インスタンス	オンプレミスインスタンス
インスタンス上で実行中のオペレーティングシステムと互換性がある CodeDeploy エージェントのバージョンをインストールおよび実行する必要があります。	はい	はい
インスタンスは、CodeDeploy に接続できる必要があります。	はい	はい
IAM インスタンスプロファイルがインスタンスに添付される必要があります。IAM インスタンスプロファイルには、CodeDeploy デプロイに参加する許可が必要です。詳細については、「ステップ 4: Amazon EC2 インスタンス用の IAM インスタンスプロファ	はい	いいえ

件名	Amazon EC2 インスタンス	オンプレミスインスタンス
イルを作成する」を参照してください。		
次のいずれかの操作を行って認証を行い、インスタンスを登録する必要があります。 • AWS Security Token Serviceを通して生成され、定期的に更新される一時的な認証情報を取得するための各インスタンス上で、IAM ユーザーによって推定され得る IAM ロールを作成します。 • 各インスタンスの IAM ユーザーを作成し、インスタンス上に簡易なテキストで IAM ユーザーのアカウント認証情報を保存します。	いいえ	はい
それへ実行できる前に、CodeDeploy を用いて各インスタンスを登録する必要があります。	いいえ	はい
CodeDeploy がデプロイを実行できる前に、各インスタンスにタグを付ける必要があります。	はい	はい
CodeDeploy デプロイの一環として、Amazon EC2 Auto Scaling および Elastic Load Balancing のシナリオに参加できます。	はい	いいえ

件名	Amazon EC2 インスタンス	オンプレミスインスタンス
Amazon S3 バケットと GitHub リポジトリからデプロイされることができます。	はい	はい
指定されたイベントがデプロイまたはインスタンスで発生したときに、SMS または E メール通知の送信を求めるトリガーをサポートできる。	はい	はい
関連デプロイへの請求対象である。	いいえ	はい

CodeDeploy のためのインスタンスタスク

デプロイで使用するインスタンスを起動または設定するには、以下の手順から選択します。

新しい Amazon Linux または Windows Server の Amazon EC2 インスタンスを起動する場合。	最小限の労力で Amazon EC2 インスタンスを起動するには、CodeDeploy 用の Amazon EC2 インスタンスを作成する (CloudFormation テンプレート) を参照してください。 主に自分で Amazon EC2 インスタンスを起動するには、CodeDeploy (AWS CLI または Amazon EC2 コンソール) 用の Amazon EC2 インスタンスを作成する を参照してください。
新しい Ubuntu Server または RHEL の Amazon EC2 インスタンスを起動する場合。	「CodeDeploy (AWS CLI または Amazon EC2 コンソール) 用の Amazon EC2 インスタンスを作成する」を参照してください。
Amazon Linux、Windows サーバー、Ubuntu サーバー、または RHEL Amazon EC2 インスタンスを構成する場合。	「CodeDeploy と共に動作するように Amazon EC2 インスタンスを構成します」を参照してください。

Windows サーバー、Ubuntu サーバー、または RHEL オンプレミスインスタンス (Amazon EC2 インスタンスではない物理デバイス) を構成する場合。	「Working with On-Premises Instances」を参照してください。
blue/green のデプロイ中に、CodeDeploy にインスタンスの置換フリートをプロビジョンするための CodeDeploy が必要な場合。	「CodeDeploy でのデプロイグループの使用」を参照してください。

Amazon EC2 Auto Scaling グループ中の Amazon EC2 インスタンスを準備するには、追加のステップに従う必要があります。詳細については、「CodeDeploy と Amazon EC2 Auto Scaling の統合」を参照してください。

トピック

- Tagging Instances for Deployments
- Working with Amazon EC2 Instances
- Working with On-Premises Instances
- View Instance Details
- Instance Health

CodeDeploy 中のデプロイグループのためのインスタンスのタグ付け

Amazon EC2 インスタンスとオンプレミスインスタンスを管理するために、タグを使用して独自のメタデータを各リソースに割り当てることができます。タグを使用すると、インスタンスをさまざまな方法 (目的、所有者、環境など) で分類することができます。これはインスタンスが多数ある場合に便利です。割り当てられたタグに基づいて、インスタンスやインスタンスグループを迅速に識別できます。各タグはキーとオプションの値で構成され、どちらもお客様側が定義します。詳細については、「Amazon EC2 リソースにタグを付ける」を参照してください。

どのインスタンスを CodeDeploy デプロイグループ中に含めるかを指定するには、1 つ以上の tag groups にタグを指定します。タグ条件を満たすインスタンスは、そのデプロイグループへのデプロイが作成されたときにアプリケーションの最新のリビジョンがインストール済みのものです。

> **Note**
>
> また、Amazon EC2 Auto Scaling グループをデプロイグループに含めることもできますが、これらはインスタンスに適用されたタグではなく名前によって識別されます。詳細については、「CodeDeploy と Amazon EC2 Auto Scaling の統合」を参照してください。

デプロイグループのインスタンスの条件は、単一のタググループの単一のタグと同じほど簡単です。これは、最大 3 つのタググループそれぞれに 10 個のタグという複雑なものにもできます。

単一のタググループを使用する場合は、グループ内の少なくとも 1 つのタグによって識別されたインスタンスがデプロイグループに含まれます。複数のタググループを使用する場合は、タググループそれぞれの少なくとも 1 つのタグによって識別されたインスタンスのみが含まれます。

次の例は、タグとタググループを使用してデプロイグループのインスタンスを選択する方法を説明します。

トピック

- 例 1: 単一タググループ、単一タグ
- 例 2: 単一タググループ、複数タグ
- 例 3: 複数タググループ、複数タグ
- 例 4: 複数タググループ、複数タグ

例 1: 単一タググループ、単一タグ

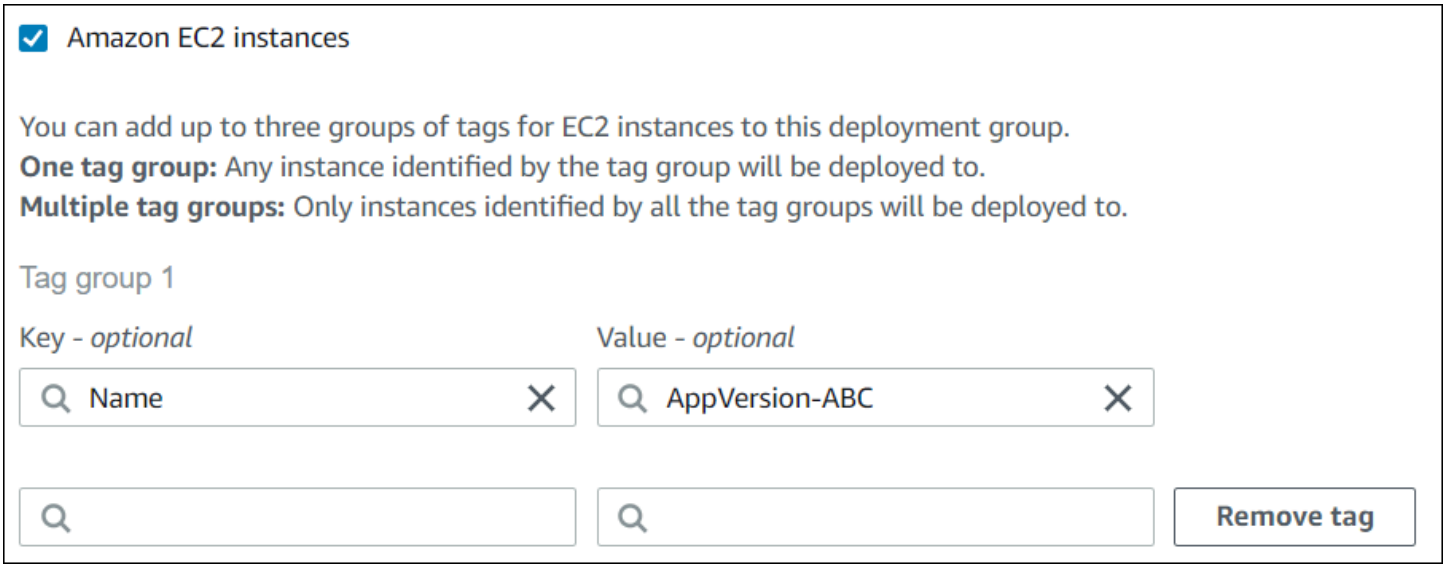
単一のタグを単一のタググループに指定できます。

タググループ 1

キー	値
名前	AppVersion-ABC

`Name=AppVersion-ABC` というタグが付いている各インスタンスは、他のタグがついていても、デプロイグループの一部になります。

CodeDeploy コンソールのセットアップビュー。



JSON の構造:

```
"ec2TagSet": {
         "ec2TagSetList": [
            [
               {
                  "Type": "KEY_AND_VALUE",
                  "Key": "Name",
                  "Value": "AppVersion-ABC"
               }
            ]
         ]
      },
```

例 2: 単一タググループ、複数タグ

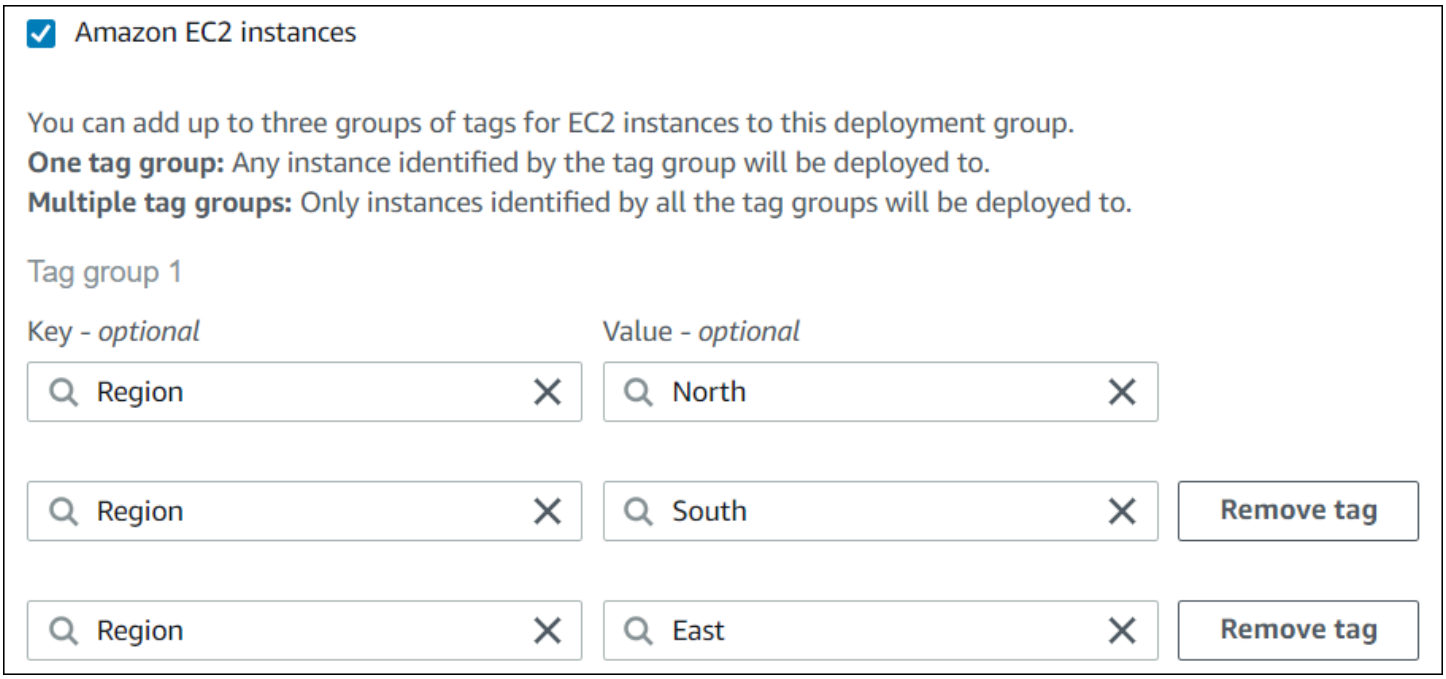
単一のタグを複数のタググループに指定することもできます。

タググループ 1

キー	値
リージョン	North
リージョン	South
リージョン	East

この 3 つのタグのうちいずれかが付いているインスタンスは、他のタグがついていても、デプロイグループの一部になります。たとえば、Region=West というタグが付いている他のインスタンスがある場合、それらはデプロイグループに含まれません。

CodeDeploy コンソールのセットアップビュー。



JSON の構造:

```
"ec2TagSet": {
         "ec2TagSetList": [
            [
               {
                  "Type": "KEY_AND_VALUE",
                  "Key": "Region",
                  "Value": "North"
               },
                              {
                  "Type": "KEY_AND_VALUE",
                  "Key": "Region",
                  "Value": "South"
               },
                              {
                  "Type": "KEY_AND_VALUE",
                  "Key": "Region",
                  "Value": "East"
               }
```

```
            ]
         ]
      },
```

例 3: 複数タググループ、複数タグ

それぞれに単一のキーと値のペアを持つタググループの複数セットを使用して、デプロイグループのインスタンスの条件を指定することもできます。デプロイグループで複数のタググループを使用する場合は、すべてのタググループによって識別されたインスタンスのみがデプロイグループに含まれます。

タググループ 1

キー	値
名前	AppVersion-ABC

タググループ 2

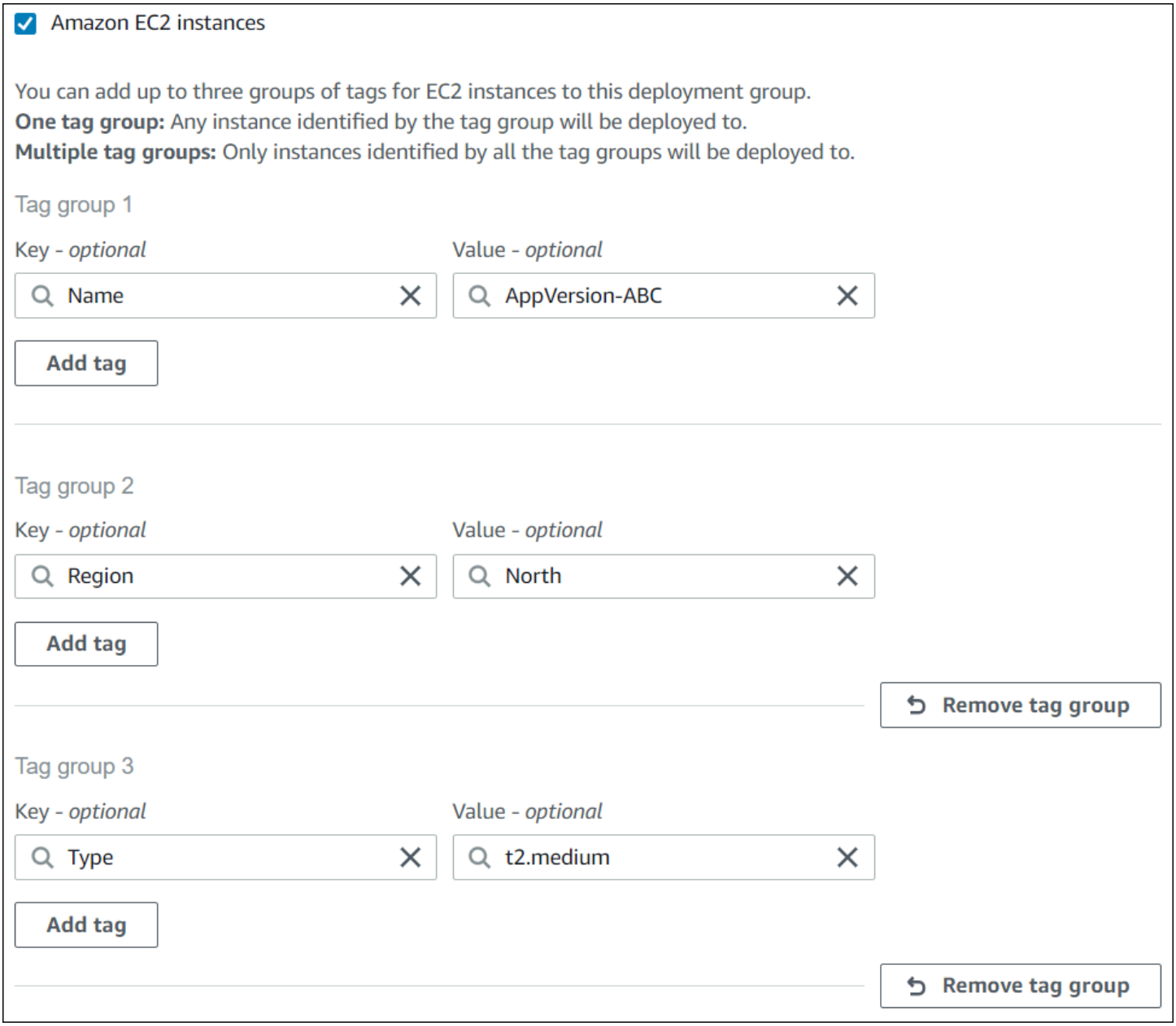
キー	値
リージョン	North

タググループ 3

キー	値
タイプ	t2.medium

多くのリージョンにインスタンスがあり、 でタグ付けされたさまざまなインスタンスタイプのインスタンスがある場合がありますName=AppVersion-ABC。この例では、同じく Region=North および Type=t2.medium のタグが付いたインスタンスのみがデプロイグループの一部になります。

CodeDeploy コンソールのセットアップビュー。



JSON の構造:

```
"ec2TagSet": {
         "ec2TagSetList": [
            [
               {
                  "Type": "KEY_AND_VALUE",
                  "Key": "Name",
                  "Value": "AppVersion-ABC"
               }
            ],
```

```
            [
               {
                  "Type": "KEY_AND_VALUE",
                  "Key": "Region",
                  "Value": "North"
               }
            ],
            [
               {
                  "Type": "KEY_AND_VALUE",
                  "Key": "Type",
                  "Value": "t2.medium"
               }
            ],
         ]
      },
```

例 4: 複数タググループ、複数タグ

複数のタグを持つ複数のタググループを 1 つ以上のグループで使用する場合、インスタンスはそれぞれのグループの少なくとも 1 つのタグに一致している必要があります。

タググループ 1

キー	値
環境	Beta
環境	ステージング

タググループ 2

キー	値
リージョン	North
リージョン	South
リージョン	East

タググループ 3

キー	値
タイプ	t2.medium
タイプ	t2.large

この例では、デプロイグループに含まれるには、インスタンスが次のようにタグ付けされている必要があります。(1) `Environment=Beta` または `Environment=Staging`、(2) `Region=North`、`Region=South` または `Region=East`、(3) `Type=t2.medium` または `Type=t2.large`。

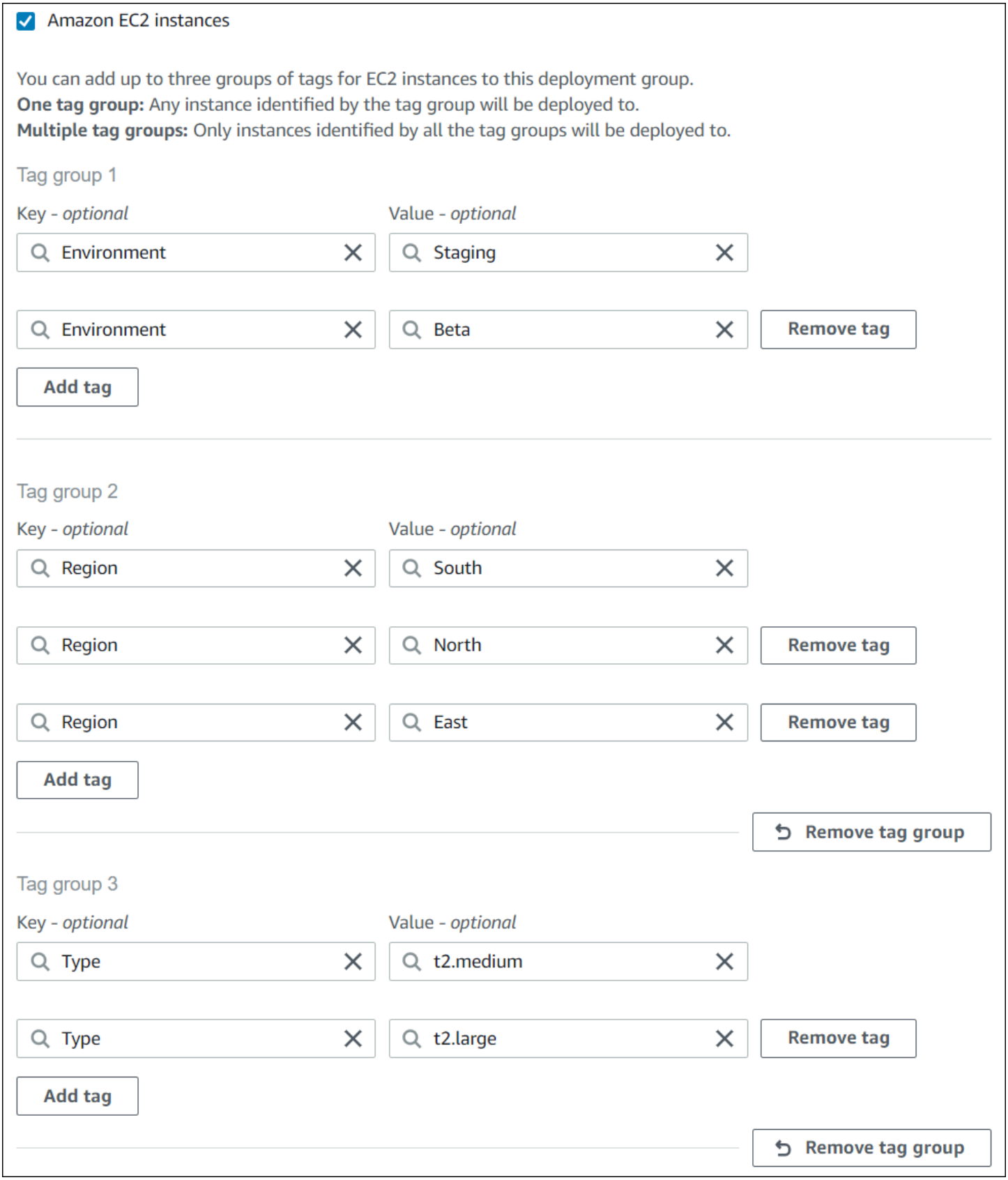
たとえば、次のタググループを持つインスタンスは、デプロイグループに含まれるもののひとつになることがあります。

- `Environment=Beta, Region=North,Type=t2.medium`
- `Environment=Staging,Region=East,Type=t2.large`
- `Environment=Staging,Region=South,Type=t2.large`

次のタググループを持つインスタンスは、デプロイグループに含まれないことがあります。強調表示されたキー値があると、インスタンスは除外されます。

- `Environment=Beta`, Region=West,`Type=t2.medium`
- `Environment=Staging,Region=East`,Type=t2.micro
- Environment=Production,`Region=South,Type=t2.large`

CodeDeploy コンソールのセットアップビュー。


Amazon EC2 instances
You can add up to three groups of tags for EC2 instances to this deployment group.
One tag group: Any instance identified by the tag group will be deployed to.
Multiple tag groups: Only instances identified by all the tag groups will be deployed to.
Tag group 1
Key - optional
Value - optional
Environment
Staging
Environment
Beta
Remove tag
Add tag
Tag group 2
Key - optional
Value - optional
Region
South
Region
North
Remove tag
Region
East
Remove tag
Add tag
Remove tag group
Tag group 3
Key - optional
Value - optional
Type
t2.medium
Type
t2.large
Remove tag
Add tag
Remove tag group

JSON の構造:

```
"ec2TagSet": {
         "ec2TagSetList": [
            [
               {
                  "Type": "KEY_AND_VALUE",
                  "Key": "Environment",
                  "Value": "Beta"
               },
               {
                  "Type": "KEY_AND_VALUE",
                  "Key": "Environment",
                  "Value": "Staging"
               }
            ],
            [
               {
                  "Type": "KEY_AND_VALUE",
                  "Key": "Region",
                  "Value": "North"
               },
               {
                  "Type": "KEY_AND_VALUE",
                  "Key": "Region",
                  "Value": "South"
               },
               {
                  "Type": "KEY_AND_VALUE",
                  "Key": "Region",
                  "Value": "East"
               }
            ],
            [
               {
                  "Type": "KEY_AND_VALUE",
                  "Key": "Type",
                  "Value": "t2.medium"
               },
               {
                  "Type": "KEY_AND_VALUE",
                  "Key": "Type",
                  "Value": "t2.large"
               }
```

```
            ],
        ]
    },
```

CodeDeploy のための Amazon EC2 インスタンスを用いた作業

Amazon EC2 インスタンスは、Amazon Elastic Compute Cloud を使用して作成および設定する仮想コンピューティング環境です Amazon EC2 は、 AWS クラウドでスケーラブルなコンピューティング容量を提供します。Amazon EC2 を使用して、CodeDeploy デプロイに必要な数だけ仮想サーバーを起動できます。

Amazon EC2 の詳細については、Amazon EC2 入門ガイド を参照してください。

このセクションの説明では、CodeDeploy のデプロイで使用する Amazon EC2 インスタンスを作成および設定する方法を示します。

トピック

- CodeDeploy (AWS CLI または Amazon EC2 コンソール) 用の Amazon EC2 インスタンスを作成する
- CodeDeploy 用の Amazon EC2 インスタンスを作成する (CloudFormation テンプレート)
- CodeDeploy と共に動作するように Amazon EC2 インスタンスを構成します

CodeDeploy (AWS CLI または Amazon EC2 コンソール) 用の Amazon EC2 インスタンスを作成する

これらの説明では、CodeDeploy デプロイ用に設定した新しい Amazon EC2 インスタンスを起動する方法を示します。

CloudFormation テンプレートを使用して、CodeDeploy デプロイで使用するように既に設定されている Amazon Linux または Windows Server を実行している Amazon EC2 インスタンスを起動できます。Ubuntu Server または Red Hat Enterprise Linux (RHEL) を実行している Amazon EC2 インスタンスの CloudFormation テンプレートは提供していません。テンプレートを使用する代わりに、「CodeDeploy のためにインスタンスを用いた操作」を参照してください。

Amazon EC2 コンソール AWS CLI、または Amazon EC2 APIsを使用して Amazon EC2 インスタンスを起動できます。

Amazon EC2 インスタンス (コンソール) を起動します。

前提条件

まだ行っていない場合は、CodeDeploy の開始方法「」の手順に従って をセットアップおよび設定 AWS CLI し、IAM インスタンスプロファイルを作成します。

Amazon EC2 インスタンスの起動

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/ec2/ で Amazon EC2 コンソールを開きます。
2. ナビゲーションペインで [インスタンス]、[インスタンスの作成] の順に選択します。
3. [Step 1: Choose an Amazon Machine Image (AMI)] ページで、[Quick Start] タブから、使用するオペレーションシステムおよびバージョンを探して、[Select] を選択します。CodeDeploy によりサポートされている Amazon EC2 AMI オペレーティングシステムを選択する必要があります。詳細については、「CodeDeploy エージェントで対応するオペレーティングシステム」を参照してください。
4. [ステップ 2: インスタンスタイプの選択] ページで、利用可能な Amazon EC2 インタンスタイプを選択し、[次の手順: インスタンスの詳細の設定] を選択します。
5. IAM role 中の Step 3: Configure Instance Details ページ上で、ステップ 4: Amazon EC2 インスタンス用の IAM インスタンスプロファイルを作成する で作成した IAM インスタンスロールを選択します。提案されたロール名を使用している場合は、[**CodeDeployDemo-EC2-Instance-Profile**] を選択します。独自のロール名を作成した場合は、その名前を選択します。

   > **Note**
   >
   > デフォルトの仮想プライベートクラウド (VPC) がネットワークの一覧に表示されていない場合は、Amazon VPC とサブネットを選択するか、作成する必要があります。[Create new VPC (新しい VPC の作成)] または [Create new subnet (新しいサブネットの作成)]、またはその両方を選択します。詳細については、「VPC とサブネット」を参照してください。

6. [次の手順: ストレージの追加] を選択してください。
7. [ステップ 4: ストレージの追加] ページは変更せず、[次の手順: タグの追加] を選択します。
8. [Step 5: Add Tags] (ステップ 5: タグの追加) ページで[Add Tag] (タグの追加) を選択します。
9. [キー] ボックスに「**Name**」と入力します。[値] ボックスに「**CodeDeployDemo**」と入力します。

> **Important**
>
> [キー] ボックスと [値] ボックスの内容は大文字と小文字が区別されます。

10. [Next: Configure Security Group] (次に)：セキュリティグループを設定)を選択します。
11. [ステップ 6: セキュリティグループの設定] ページで、[新規セキュリティグループを作成] オプションを選択したままにします。

    デフォルトの SSH ロールは、Amazon Linux、Ubuntu サーバー、または RHEL を実行する Amazon EC2 インスタンスのために設定されます。デフォルトの RDP ロールは、Windows サーバーを実行する Amazon EC2 インスタンスのために設定されます。
12. HTTP ポートを開く場合は、[ルールの追加] ボタンを選択し、[タイプ] ドロップダウンリストから、**[HTTP]** を選択します。Custom 0.0.0.0/0 のデフォルトの Source を受け入れ、次に Review and Launch を選択します。

    > **Note**
    >
    > 本稼働環境では、Anywhere 0.0.0.0/0 を指定する代わりに、SSH、RDP、および HTTP ポートへのアクセスを制限することを推奨します。CodeDeploy は、無制限のポートアクセスを必要とせず、HTTP アクセスを必要としません。詳細については、「Amazon EC2 インスタンスの保護のヒント」を参照してください。

    [汎用 (SSD) から起動する] ダイアログボックスが表示されたら、指示に従って [次へ] を選択します。
13. [Step 7: Review Instance Launch] ページは変更せず、[Launch] を選択します。
14. [既存のキーペアの選択または新しいキーペアの作成] ダイアログボックスで、[既存のキーペアの選択] または [新しいキーペアの作成] を選択します。すでに Amazon EC2 インスタンスキーペアを設定している場合は、ここで選択できます。

    Amazon EC2 インスタンスのキーペアがまだない場合は、[Create a new key pair (新しいキーペアの作成)] を選択して、わかりやすい名前を付けます。お客様のコンピュータに Amazon EC2 インスタンスキーペアをダウンロードするために、Download Key Pair を選択します。

> **Important**
>
> SSH または RDP を使用して、Amazon EC2 インスタンスへアクセスしたい場合は、キーペアが必要です。

15. [Launch Instances] (インスタンスを起動) をクリックします。
16. Amazon EC2 インスタンスのための ID を選択します。インスタンスが起動され、すべてのチェックが成功するまで先に進まないでください。

CodeDeploy エージェントをインストールします

CodeDeploy エージェントは、CodeDeploy デプロイ中で使用する前に、Amazon EC2 インスタンス上にインストールする必要があります。詳細については、「CodeDeploy エージェントをインストールする」を参照してください。

> **Note**
>
> コンソールでデプロイグループを作成するときに、CodeDeploy エージェントの自動インストールと更新を設定できます。

Amazon EC2 インスタンス (CLI) の起動

前提条件

まだ行っていない場合は、CodeDeploy の開始方法「」の手順に従って をセットアップおよび設定 AWS CLI し、IAM インスタンスプロファイルを作成します。

Amazon EC2 インスタンスの起動

1. For Windows Server only Windows サーバーを実行している Amazon EC2 インスタンスを作成中の場合、create-security-group と authorize-security-group-ingress のコマンドを呼び出し、実行する インスタンスを作成する場合は、RDP アクセス (デフォルトでは許可されない)、または代わりに HTTP アクセスを許可するセキュリティグループを作成します。例えば、CodeDeployDemo-Windows-Security-Group という名前のセキュリティグループを作成するには、次のコマンドを 1 つずつ実行します。

```
aws ec2 create-security-group --group-name CodeDeployDemo-Windows-Security-Group --
description "For launching Windows Server images for use with CodeDeploy"
```

```
aws ec2 authorize-security-group-ingress --group-name CodeDeployDemo-Windows-
Security-Group --to-port 3389 --ip-protocol tcp --cidr-ip 0.0.0.0/0 --from-port
 3389
```

```
aws ec2 authorize-security-group-ingress --group-name CodeDeployDemo-Windows-
Security-Group --to-port 80 --ip-protocol tcp --cidr-ip 0.0.0.0/0 --from-port 80
```

> **Note**
>
> デモンストレーションのため、これらのコマンドは、ポート 3389 を経由して RDP に、またはポート 80 を経由して HTTP に無制限アクセスを許可するセキュリティグループを作成します。ベストプラクティスとして、SSH および HTTP ポートへのアクセスを制限することをお勧めします。CodeDeploy は、無制限のポートアクセスを必要とせず、HTTP アクセスを必要としません。詳細については、「Amazon EC2 インスタンスの保護のヒント」を参照してください。

2. run-instances のコマンドを呼び出して、Amazon EC2 インスタンスを作成して起動します。

   このコマンドを呼び出す前に、以下を収集する必要があります。

   - インスタンスに使用する Amazon マシンイメージ (AMI) (*ami-id*) の ID。ID を取得するには、「適切な AMI の検索」を参照してください。
   - 例えば `t1.micro` のように、作成する Amazon EC2 インスタンス (*instance-type*) のタイプ名。リストについては、Amazon EC2 instance types を参照してください。
   - リージョンの CodeDeploy エージェントのインストールファイルが保存されている Amazon S3 バケットにアクセスするアクセス許可を持つ IAM インスタンスプロファイルの名前。

     IAM インスタンスプロファイルの作成についての情報は、ステップ 4: Amazon EC2 インスタンス用の IAM インスタンスプロファイルを作成する を参照してください。
   - Amazon Linux、Ubuntu サーバー、または RHEL または RDP アクセスを実行している Amazon EC2 インスタンスへのSSH、あるいは Windows サーバーを実行している Amazon EC2 へRDP アクセスを可能とする Amazon EC2インスタンスキーの名前 (*key-name*)。

> **Important**
>
> キーペアのファイル拡張子ではなく、キーペア名のみを入力します。例えば、my-keypair.pem ではなく、my-keypair です。

キーペア名を見つけるには、https://console.aws.amazon.com/ec2 でAmazon EC2 コンソールを開きます。ナビゲーションペインの [ネットワーク & セキュリティ] の下で、[キーペア] を選択し、リストのキーペア名をメモします。

新しいキーペアを生成するには、「Amazon EC2 を使用してキーペアを作成する」を参照してください。のリージョンとエンドポイントにリストされているリージョンのいずれかでキーペアを作成してくださいAWS 全般のリファレンス。そうしないと、CodeDeploy を用いて Amazon EC2 インスタンスのキーペアを使用できなくなります。

Amazon Linux、RHEL、および Ubuntu Server 用

Amazon Linux, Ubuntu Server、あるいは RHEL を実行する Amazon EC2 インスタンスを開始するための run-instances のコマンドを呼び出し、ステップ 4: Amazon EC2 インスタンス用の IAM インスタンスプロファイルを作成する で作成した IAM インスタンスプロファイルをアタッチする方法。例: 。

```
aws ec2 run-instances \
  --image-id ami-id \
  --key-name key-name \
  --count 1 \
  --instance-type instance-type \
  --iam-instance-profile Name=iam-instance-profile
```

> **Note**
>
> このコマンドは、ポート 22 を経由した SSH の無制限のアクセス、または、ポート 80 を経由した HTTP など複数のポートへのアクセスを許可する Amazon EC2 インスタンスのデフォルトのセキュリティグループを作成します。ベストプラクティスとして、SSH および HTTP ポートへのアクセスを制限することをお勧めします。CodeDeploy は、無制限のポートアクセスを必要とせず、HTTP ポートアクセスを

必要としません。詳細については、「Amazon EC2 インスタンスの保護のヒント」を参照してください。

Windows Server の場合

run-instances のコマンドを呼び出して Windows サーバーを実行する Amazon EC2 インスタンスを起動し、ステップ 4: Amazon EC2 インスタンス用の IAM インスタンスプロファイルを作成する で作成した IAM インスタンスプロファイルをアタッチして、ステップ 1 で作成したセキュリティグループの名前を指定するには、例: 。

```
aws ec2 run-instances --image-id ami-id --key-name key-name --count 1 --instance-
type instance-type --iam-instance-profile Name=iam-instance-profile --security-
groups CodeDeploy-Windows-Security-Group
```

これらのコマンドは、指定された AMI、キーペア、およびインスタンスタイプ、指定された IAM インスタンスプロファイルを用いて単一の Amazon EC2 インスタンスプロファイルを起動し、起動時に指定されたスクリプトを実行します。

3. 出力の `InstanceID` の値を記録します。この値を忘れた場合は、Amazon EC2 インスタンスのキーペアに対する describe-instances のコマンドを呼び出すことで、後で取得できます。

```
aws ec2 describe-instances --filters "Name=key-name,Values=keyName" --query
 "Reservations[*].Instances[*].[InstanceId]" --output text
```

インスタンス ID を使用して create-tags のコマンドを呼び出します。このコマンドで Amazon EC2 インスタンスにタグが付けられ、後でデプロイ中に、CodeDeploy が見つけることができるようになります。次の例で、タグ名は **`CodeDeployDemo`** と名付けられていますが、希望する Amazon EC2 インスタンスタグを指定できます。

```
aws ec2 create-tags --resources instance-id --tags Key=Name,Value=CodeDeployDemo
```

複数のタグを同時にインスタンスに適用できます。例: 。

```
aws ec2 create-tags --resources instance-id --tags Key=Name,Value=testInstance
 Key=Region,Value=West Key=Environment,Value=Beta
```

Amazon EC2 インスタンスが起動され、すべてのチェックが成功したことを確認するには、describe-instance-status のコマンドを呼び出すためのインスタンス ID を使用します。

```
aws ec2 describe-instance-status --instance-ids instance-id --query
 "InstanceStatuses[*].InstanceStatus.[Status]" --output text
```

インスタンスが起動され、すべてのチェックが成功すると、ok が出力に表示されます。

CodeDeploy エージェントをインストール

CodeDeploy エージェントは、CodeDeploy デプロイ中で使用する前に、Amazon EC2 インスタンス上にインストールする必要があります。詳細については、「CodeDeploy エージェントをインストールする」を参照してください。

> **Note**
>
> コンソールでデプロイグループを作成するときに、CodeDeploy エージェントの自動インストールと更新を設定できます。

CodeDeploy 用の Amazon EC2 インスタンスを作成する (CloudFormation テンプレート)

CloudFormation テンプレートを使用して、Amazon Linux または Windows Server を実行する Amazon EC2 インスタンスをすばやく起動できます。テンプレートを使用してインスタンスを起動するには AWS CLI、、CodeDeploy コンソール、または AWS APIs を使用できます。インスタンスを起動することに加えて、テンプレートでは以下を実行します。

- CodeDeploy デプロイに参加するアクセス許可をインスタンスに付与 CloudFormation するように に指示します。
- インスタンスにタグ付けして、CodeDeploy がデプロイ中に見つけられるようにします。
- インスタンス上で CodeDeploy エージェントをインストールおよび実行します。

CloudFormation を使用して Amazon EC2 インスタンスをセットアップする必要はありません。代替方法については、「CodeDeploy のためにインスタンスを用いた操作」を参照してください。

Ubuntu Server または Red Hat Enterprise Linux (RHEL) を実行している Amazon EC2 インスタンスの CloudFormation テンプレートは提供していません。

トピック

- [開始する前に]
- CloudFormation テンプレートを使用して Amazon EC2 インスタンスを起動する (コンソール)
- CloudFormation テンプレートを使用して Amazon EC2 インスタンスを起動する (AWS CLI)

[開始する前に]

CloudFormation テンプレートを使用して Amazon EC2 インスタンスを起動する前に、次の手順を完了してください。

1. 「ステップ 1: セットアップ」の説明に従って管理ユーザーを作成したことを確認します。ユーザーに次の最小限の許可があることをもう一度確認し、存在しないものを追加します。

   - cloudformation:*
   - codedeploy:*
   - ec2:*
   - iam:AddRoleToInstanceProfile
   - iam:CreateInstanceProfile
   - iam:CreateRole
   - iam:DeleteInstanceProfile
   - iam:DeleteRole
   - iam:DeleteRolePolicy
   - iam:GetRole
   - iam:DeleteRolePolicy
   - iam:PutRolePolicy
   - iam:RemoveRoleFromInstanceProfile

2. Amazon Linux を実行する Amazon EC2 インスタンスへの SSH アクセス、または Windows Server を実行するインスタンスへの RDP アクセスを有効にするインスタンスのキーペアがあることを確認します。

キーペア名を見つけるには、https://console.aws.amazon.com/ec2 でAmazon EC2 コンソールを開きます。ナビゲーションペインの [ネットワーク & セキュリティ] の下で、[キーペア] を選択し、リストのキーペア名をメモします。

新しいキーペアを生成するには、「Amazon EC2 を使用してキーペアを作成する」を参照してください。キーペアが、 のリージョンとエンドポイントにリストされているリージョンのいずれかで作成されていることを確認しますAWS 全般のリファレンス。それ以外の場合、CodeDeploy を用いてインスタンスのキーペアを使用できなくなります。

CloudFormation テンプレートを使用して Amazon EC2 インスタンスを起動する (コンソール)

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/cloudformation で CloudFormation コンソールを開きます。

   > **Important**
   >
   > で使用したのと同じアカウント AWS マネジメントコンソール で にサインインしますCodeDeploy の開始方法。ナビゲーションバーのリージョンセレクターで、 のリージョンとエンドポイントにリストされているリージョンのいずれかを選択しますAWS 全般のリファレンス。CodeDeploy は、これらのリージョンのみをサポートします。

2. [Create Stack] (スタックの作成) を選択します。
3. [テンプレートの選択] で、[Amazon S3 テンプレート URL の指定] を選択します。ボックスに、リージョンの CloudFormation テンプレートの場所を入力し、次へを選択します。

リージョン	CloudFormation テンプレートの場所
米国東部 (オハイオ) リージョン	`http://s3-us-east-2.amazonaws.com/aws-codedeploy-us-east-2/templates/latest/CodeDeploy_SampleCF_Template.json`
米国東部(バージニア州北部) リージョン	`http://s3.amazonaws.com/aws-codedeploy-us-east-1/templ`

リージョン	CloudFormation テンプレートの場所
	ates/latest/CodeDeploy_Samp leCF_Template.json
US West (N. California) Region	http://s3-us-west-1.amazona ws.com/aws-codedeploy-us-we st-1/templates/latest/CodeD eploy_SampleCF_Template.json
米国西部 (オレゴン) リージョン	http://s3-us-west-2.amazona ws.com/aws-codedeploy-us-we st-2/templates/latest/CodeD eploy_SampleCF_Template.json
カナダ (中部) リージョン	http://s3-ca-central-1.amaz onaws.com/aws-codedeploy-ca -central-1/templates/latest /CodeDeploy_SampleCF_Templa te.json
欧州 (アイルランド) リージョン	http://s3-eu-west-1.amazona ws.com/aws-codedeploy-eu-we st-1/templates/latest/CodeD eploy_SampleCF_Template.json
欧州 (ロンドン) リージョン	http://s3-eu-west-2.amazona ws.com/aws-codedeploy-eu-we st-2/templates/latest/CodeD eploy_SampleCF_Template.json
欧州(パリ)リージョン	http://s3-eu-west-3.amazona ws.com/aws-codedeploy-eu-we st-3/templates/latest/CodeD eploy_SampleCF_Template.json

リージョン	CloudFormation テンプレートの場所
欧州(フランクフルト)リージョン	http://s3-eu-central-1.amazonaws.com/aws-codedeploy-eu-central-1/templates/latest/CodeDeploy_SampleCF_Template.json
イスラエル (テルアビブ) リージョン	http://s3-il-central-1.amazonaws.com/aws-codedeploy-il-central-1/templates/latest/CodeDeploy_SampleCF_Template.json
アジアパシフィック (香港) リージョン	http://s3-ap-east-1.amazonaws.com/aws-codedeploy-ap-east-1/templates/latest/CodeDeploy_SampleCF_Template.json
Asia Pacific (Tokyo) Region	http://s3-ap-northeast-1.amazonaws.com/aws-codedeploy-ap-northeast-1/templates/latest/CodeDeploy_SampleCF_Template.json
Asia Pacific (Seoul) Region	http://s3-ap-northeast-2.amazonaws.com/aws-codedeploy-ap-northeast-2/templates/latest/CodeDeploy_SampleCF_Template.json
アジアパシフィック (シンガポール) リージョン	http://s3-ap-southeast-1.amazonaws.com/aws-codedeploy-ap-southeast-1/templates/latest/CodeDeploy_SampleCF_Template.json

リージョン	CloudFormation テンプレートの場所
アジアパシフィック (シドニー) リージョン	`http://s3-ap-southeast-2.amazonaws.com/aws-codedeploy-ap-southeast-2/templates/latest/CodeDeploy_SampleCF_Template.json`
アジアパシフィック (メルボルン) リージョン	`https://aws-codedeploy-ap-southeast-4.s3.ap-southeast-4.amazonaws.com/templates/latest/CodeDeploy_SampleCF_Template.json`
アジアパシフィック (ムンバイ) リージョン	`http://s3-ap-south-1.amazonaws.com/aws-codedeploy-ap-south-1/templates/latest/CodeDeploy_SampleCF_Template.json`
南米 (サンパウロ) リージョン	`aws-codedeploy-ap-northeast-1.s3.sa-east-1.amazonaws.com/templates/latest/CodeDeploy_SampleCF_Template.json`

4. [スタック名] ボックスに、スタックの名前 (**`CodeDeployDemoStack`** など) を入力します
5. [Parameters] に以下を入力し、[Next] を選択します。

   - [InstanceCount] には、起動するインスタンスの数を入力します (デフォルトの 1 のままにしておくことをお勧めします)。
   - [InstanceType] には、起動するインスタンスタイプを入力します (またはデフォルトの t1.micro のままにします)。
   - [KeyPairName] のために、インスタンスのキーペア名をタイプします。キーペアのファイル拡張子ではなく、キーペア名のみを入力します。
   - OperatingSystem ボックスにのために、**`Windows`** をタイプし、Windows サーバーを実行するインスタンスを起動します (または Linux のデフォルトのままにします)。

- [SSHLocation] には、SSH または RDP でインスタンスに接続するのに使用される IP アドレス範囲を入力します (またはデフォルトの 0.0.0.0/0 のままにします)。

  > **Important**
  >
  > **0.0.0.0/0** のデフォルトは、デモンストレーションのみを目的として提供されています。CodeDeploy では、Amazon EC2 インスタンスのポートへの無制限のアクセス権限を持つ必要はありません。ベストプラクティスとして、SSH (および HTTP) ポートへのアクセスを制限することをお勧めします。詳細については、「Amazon EC2 インスタンスの保護のヒント」を参照してください。

- TagKey のために、デプロイ中に CodeDeploy がインスタンスを特定するために使うインスタンスのタグキーをタイプします (または Name のデフォルトのままに)。
- TagValue のために、CodeDeploy がデプロイ中にインスタンスを特定するために使用するインスタンスのタグ値をタイプします (または CodeDeployDemo のデフォルトのままに)。

6. [Options] ページで、オプションボックスは空白のまま残し、[Next] を選択します。

   > **Important**
   >
   > CloudFormation タグは CodeDeploy タグとは異なります。 はタグ CloudFormation を使用してインフラストラクチャの管理を簡素化します。CodeDeploy は、タグを使用して Amazon EC2 インスタンスを識別します。お客様は、Specify Parameters ページ上に CodeDeploy タクを指定しました。

7. 「レビュー」ページの「機能」で、IAM リソースを作成する CloudFormation 可能性がある I acknowledge を選択し、「作成」を選択します。

   CloudFormation がスタックを作成し、Amazon EC2 インスタンスを起動すると、コンソール CloudFormation でステータス列に CREATE_COMPLETE が表示されます。この処理には数分かかることもあります。

CodeDeploy エージェントが Amazon EC2 インスタンス上で実行されていることを確認するには、CodeDeploy エージェントのオペレーションの管理 を参照して、CodeDeploy でアプリケーションを作成する に進みます。

CloudFormation テンプレートを使用して Amazon EC2 インスタンスを起動する (AWS CLI)

1. create-stack コマンドの呼び出しで CloudFormation テンプレートを使用します。このスタックは、インストールされた CodeDeploy エージェントを用いて新しい Amazon EC2 インスタンスを起動します。

   Amazon Linux を実行する Amazon EC2 インスタンスを起動するには:

   ```
   aws cloudformation create-stack \
     --stack-name CodeDeployDemoStack \
     --template-url templateURL \
     --parameters ParameterKey=InstanceCount,ParameterValue=1
    ParameterKey=InstanceType,ParameterValue=t1.micro \
       ParameterKey=KeyPairName,ParameterValue=keyName
    ParameterKey=OperatingSystem,ParameterValue=Linux \
       ParameterKey=SSHLocation,ParameterValue=0.0.0.0/0
    ParameterKey=TagKey,ParameterValue=Name \
       ParameterKey=TagValue,ParameterValue=CodeDeployDemo \
     --capabilities CAPABILITY_IAM
   ```

   Windows Server を実行中の Amazon EC2 インスタンスを起動するには

   ```
   aws cloudformation create-stack --stack-name CodeDeployDemoStack --template-
   url template-url --parameters ParameterKey=InstanceCount,ParameterValue=1
    ParameterKey=InstanceType,ParameterValue=t1.micro
    ParameterKey=KeyPairName,ParameterValue=keyName
    ParameterKey=OperatingSystem,ParameterValue=Windows
    ParameterKey=SSHLocation,ParameterValue=0.0.0.0/0
    ParameterKey=TagKey,ParameterValue=Name
    ParameterKey=TagValue,ParameterValue=CodeDeployDemo --capabilities CAPABILITY_IAM
   ```

   keyName は、インスタンスのキーペア名です。キーペアのファイル拡張子ではなく、キーペア名のみを入力します。

   template-url は、リージョンの CloudFormation テンプレートの場所です。

リージョン	CloudFormation テンプレートの場所
米国東部 (オハイオ) リージョン	http://s3-us-east-2.amazonaws.com/aws-codedeploy-us-east-2/templates/latest/CodeDeploy_SampleCF_Template.json
米国東部(バージニア州北部) リージョン	http://s3.amazonaws.com/aws-codedeploy-us-east-1/templates/latest/CodeDeploy_SampleCF_Template.json
US West (N. California) Region	http://s3-us-west-1.amazonaws.com/aws-codedeploy-us-west-1/templates/latest/CodeDeploy_SampleCF_Template.json
米国西部 (オレゴン) リージョン	http://s3-us-west-2.amazonaws.com/aws-codedeploy-us-west-2/templates/latest/CodeDeploy_SampleCF_Template.json
カナダ (中部) リージョン	http://s3-ca-central-1.amazonaws.com/aws-codedeploy-ca-central-1/templates/latest/CodeDeploy_SampleCF_Template.json
欧州 (アイルランド) リージョン	http://s3-eu-west-1.amazonaws.com/aws-codedeploy-eu-west-1/templates/latest/CodeDeploy_SampleCF_Template.json
欧州 (ロンドン) リージョン	http://s3-eu-west-2.amazonaws.com/aws-codedeploy-eu-west-2/templates/latest/CodeDeploy_SampleCF_Template.json

リージョン	CloudFormation テンプレートの場所
欧州(パリ)リージョン	http://s3-eu-west-3.amazonaws.com/aws-codedeploy-eu-west-3/templates/latest/CodeDeploy_SampleCF_Template.json
欧州(フランクフルト)リージョン	http://s3-eu-central-1.amazonaws.com/aws-codedeploy-eu-central-1/templates/latest/CodeDeploy_SampleCF_Template.json
イスラエル (テルアビブ) リージョン	http://s3-il-central-1.amazonaws.com/aws-codedeploy-il-central-1/templates/latest/CodeDeploy_SampleCF_Template.json
アジアパシフィック (香港) リージョン	http://s3-ap-east-1.amazonaws.com/aws-codedeploy-ap-east-1/templates/latest/CodeDeploy_SampleCF_Template.json
Asia Pacific (Tokyo) Region	http://s3-ap-northeast-1.amazonaws.com/aws-codedeploy-ap-northeast-1/templates/latest/CodeDeploy_SampleCF_Template.json
Asia Pacific (Seoul) Region	http://s3-ap-northeast-2.amazonaws.com/aws-codedeploy-ap-northeast-2/templates/latest/CodeDeploy_SampleCF_Template.json

リージョン	CloudFormation テンプレートの場所
アジアパシフィック (シンガポール) リージョン	http://s3-ap-southeast-1.amazonaws.com/aws-codedeploy-ap-southeast-1/templates/latest/CodeDeploy_SampleCF_Template.json
アジアパシフィック (シドニー) リージョン	http://s3-ap-southeast-2.amazonaws.com/aws-codedeploy-ap-southeast-2/templates/latest/CodeDeploy_SampleCF_Template.json
アジアパシフィック (メルボルン) リージョン	https://aws-codedeploy-ap-southeast-4.s3.ap-southeast-4.amazonaws.com/templates/latest/CodeDeploy_SampleCF_Template.json
アジアパシフィック (ムンバイ) リージョン	http://s3-ap-south-1.amazonaws.com/aws-codedeploy-ap-south-1/templates/latest/CodeDeploy_SampleCF_Template.json
南米 (サンパウロ) リージョン	aws-codedeploy-ap-northeast-1.s3.sa-east-1.amazonaws.com/templates/latest/CodeDeploy_SampleCF_Template.json

このコマンドは**CodeDeployDemoStack**、指定された Amazon S3 バケットの CloudFormation テンプレートを使用して、 という名前の CloudFormation スタックを作成します。 Amazon S3 Amazon EC2 インスタンスは、[t1.micro]インスタンスタイプに基づいていますが、任意のタイプを使用できます。このインスタンスは、**CodeDeployDemo** の値でタグ付けされていますが、任意の値でタグ付けできます。指定されたインスタンスのキーペアが適用されています。

2. describe-stacks コマンドを呼び出して、 という名前の CloudFormation スタック**CodeDeployDemoStack**が正常に作成されたことを確認します。

```
aws cloudformation describe-stacks --stack-name CodeDeployDemoStack --query
 "Stacks[0].StackStatus" --output text
```

   CREATE_COMPLETE の値が返されるまで進まないでください。

CodeDeploy エージェントが Amazon EC2 インスタンス上で実行されていることを確認するには、CodeDeploy エージェントのオペレーションの管理 を参照して、CodeDeploy でアプリケーションを作成する に進みます。

CodeDeploy と共に動作するように Amazon EC2 インスタンスを構成します

以下の手順では、Amazon Linux、Ubuntu サーバー、Red Hat エンタープライズ Linux (RHEL)、または Windows サーバーを実行して、CodeDeploy デプロイで使用する Amazon EC2 インスタンスを構成する方法について説明します。

> **Note**
>
> Amazon EC2 インスタンスがない場合は、 CloudFormation テンプレートを使用して、実行中の Amazon Linux または Windows Server を起動できます。Ubuntu Server または RHEL 用のテンプレートは提供されていません。

ステップ 1: IAM インスタンスプロファイルが Amazon EC2 インスタンスにアタッチされていることを確認する

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/ec2/ で Amazon EC2 コンソールを開きます。
2. ナビゲーションペインの [Instances] (インスタンス) で、[Instances] (インスタンス) を選択します。
3. 一覧で Amazon EC2 インスタンスを参照して選択します。
4. 詳細ペインの、[説明] タブで [IAM ロール] フィールドの値を書き留め、次のセクションに進みます。

フィールドが空の場合は、IAM インスタンスプロファイルをインスタンスにアタッチすることができます。詳細については、IAM ロールをインスタンスにアタッチする を参照してください。

ステップ 2: 添付されたインスタンスプロファイルが正しいアクセス権限を持っていることを確認します。

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで Roles (ロール) を選択してください。
3. 前のセクションのステップ 4 で書き留めた IAM ロールの名前を参照し、選択します。

> **Note**
>
> 「」の手順に従って作成したサービスロールではなく、 CloudFormation テンプレートによって生成されたサービスロールを使用する場合はステップ 2: CodeDeployのサービスのロールを作成する、次の点に注意してください。
> CloudFormation テンプレートの一部のバージョンでは、生成され、Amazon EC2 インスタンスにアタッチされた IAM インスタンスプロファイルの表示名は、IAM コンソールの表示名と同じではありません。例えば、IAM インスタンスプロファイルは、`CodeDeploySampleStack-expnyi6-InstanceRoleInstanceProfile-IK8J8A9123EX` の表示名を持っているかもしれませんが、一方で IAM コンソール中のインスタンスプロファイルは `CodeDeploySampleStack-expnyi6-InstanceRole-C5P33V1L64EX` の表示名である可能性があります。
> IAM コンソール中のインスタンスプロファイルを特定するには、`CodeDeploySampleStack-expnyi6-InstanceRole` のプレフィックスがどちらでも同じことを確認します。これらの表示名が異なる理由に関する詳細については、「インスタンスプロファイル」を参照してください。

4. [Trust Relationships] タブを選択します。The identity provider(s) ec2.amazonaws.com を読み取る [信頼されたエンティティ] にエンティティがない場合、この Amazon EC2 インスタンスを使用できません。CodeDeploy のためにインスタンスを用いた操作 の情報を使用して、Amazon EC2 インスタンスを停止して作成します。

   The identity provider(s) ec2.amazonaws.com を読み取るエントリがあり、GitHub リポジトリのみにアプリケーションを保存する場合は、「ステップ 3: Amazon EC2 インスタンスへのタグ付け」 に進みます。

The identity provider(s) ec2.amazonaws.com を読み取るエントリがある場合、およびバケットにアプリケーションを Amazon S3 に保存する場合は、Permissions タブを選択します。

5. [Permissions policies (アクセス権限ポリシー)] エリアにポリシーがある場合は、ポリシー名を選択してから [Edit policy (ポリシーの編集)] を選択します。
6. JSON タブを選択します。Amazon S3 バケット中にアプリケーションを保存している場合は、`"s3:Get*"` および `"s3:List*"` が指定したアクションのリストにあることを確認します。

   次のように表示されます。

```
{"Statement":[{"Resource":"*","Action":[
  ... Some actions may already be listed here ...
  "s3:Get*","s3:List*"
  ... Some more actions may already be listed here ...
  ],"Effect":"Allow"}]}
```

   または、次のように表示されます。

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:Get*",
        "s3:List*"
      ],
      "Resource": "*"
    }
  ]
}
```

   If `"s3:Get*"` および `"s3:List*"` が指定されたアクションのリストにない場合、[Edit] を選択して、それらを追加し、[Save] を選択します。(`"s3:Get*"` または `"s3:List*"` のどちらかが

リストの最後のアクションである場合、必ずアクションの後にコンマを追加して、ポリシードキュメントが検証するようにします。)

> **Note**
>
> このポリシーを、Amazon EC2 インスタンスがアクセスする必要のある Amazon S3 バケットにのみ制限することをお勧めします。CodeDeploy エージェントを含む Amazon S3 バケットへのアクセスを必ず許可してください。そうしない場合、CodeDeploy エージェントがインスタンス上にインストールされる、または更新されるときに、エラーが発生する可能性があります。Amazon S3 中の CodeDeploy リソースキットバケットのみへの IAM インスタンスアクセスは許すが、アクセスを防止するバケットのための行を削除するには

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:Get*",
        "s3:List*"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket/*",
        "arn:aws:s3:::aws-codedeploy-us-east-2/*",
        "arn:aws:s3:::aws-codedeploy-us-east-1/*",
        "arn:aws:s3:::aws-codedeploy-us-west-1/*",
        "arn:aws:s3:::aws-codedeploy-us-west-2/*",
        "arn:aws:s3:::aws-codedeploy-ca-central-1/*",
        "arn:aws:s3:::aws-codedeploy-eu-west-1/*",
        "arn:aws:s3:::aws-codedeploy-eu-west-2/*",
        "arn:aws:s3:::aws-codedeploy-eu-west-3/*",
        "arn:aws:s3:::aws-codedeploy-eu-central-1/*",
        "arn:aws:s3:::aws-codedeploy-eu-central-2/*",
        "arn:aws:s3:::aws-codedeploy-eu-north-1/*",
        "arn:aws:s3:::aws-codedeploy-eu-south-1/*",
        "arn:aws:s3:::aws-codedeploy-eu-south-2/*",
        "arn:aws:s3:::aws-codedeploy-il-central-1/*",
        "arn:aws:s3:::aws-codedeploy-ap-east-1/*",
        "arn:aws:s3:::aws-codedeploy-ap-northeast-1/*",
        "arn:aws:s3:::aws-codedeploy-ap-northeast-2/*",
        "arn:aws:s3:::aws-codedeploy-ap-northeast-3/*",
```

```
        "arn:aws:s3:::aws-codedeploy-ap-southeast-1/*",
        "arn:aws:s3:::aws-codedeploy-ap-southeast-2/*",
        "arn:aws:s3:::aws-codedeploy-ap-southeast-3/*",
        "arn:aws:s3:::aws-codedeploy-ap-southeast-4/*",
        "arn:aws:s3:::aws-codedeploy-ap-south-1/*",
        "arn:aws:s3:::aws-codedeploy-ap-south-2/*",
        "arn:aws:s3:::aws-codedeploy-me-central-1/*",
        "arn:aws:s3:::aws-codedeploy-me-south-1/*",
        "arn:aws:s3:::aws-codedeploy-sa-east-1/*"
      ]
    }
  ]
}
```

ステップ 3: Amazon EC2 インスタンスへのタグ付け

Amazon EC2 インスタンスにタグをつけて、CodeDeploy がデプロイ中に見つけられるようにする方法に関する詳細について、「コンソールでタグを使用する」を参照してから、このページに戻ります。

> **Note**
>
> 任意のキーおよび値を使用して Amazon EC2 インスタンスにタグを付けられます。インスタンスにデプロイするとき、必ずこのキーと値を指定してください。

ステップ 4: Amazon EC2 インスタンスに AWS CodeDeploy エージェントをインストールする

Amazon EC2 インスタンス上に CodeDeploy エージェントをインストールし、それが実行していることを検証する方法の説明については、CodeDeploy エージェントのオペレーションの管理 を参照してから CodeDeploy でアプリケーションを作成する に進みます。

CodeDeploy 用のオンプレミスインスタンスを用いての作業

オンプレミスインスタンスは、CodeDeploy エージェントを実行してパブリック AWS サービスエンドポイントに接続できる Amazon EC2 インスタンスではない物理デバイスです。

CodeDeploy アプリケーションリビジョンをオンプレミスインスタンスにデプロイするには、2 つの主なステップがあります。

- ステップ 1 – オンプレミスインスタンスを設定して CodeDeploy を用いて登録し、タグをつけます。
- ステップ 2 - アプリケーションリビジョンをオンプレミスインスタンスにデプロイします。

> **Note**
>
> サンプルアプリケーションリビジョンの作成と、正しく設定および登録されたオンプレミスインスタンスへのデプロイを試す場合は、「チュートリアル: CodeDeploy (Windows サーバー、Ubuntu サーバー、または Red Hat エンタープライズ Linux) を使用してオンプレミスインスタンスにアプリケーションをデプロイします。」を参照してください。オンプレミスインスタンス、および CodeDeploy の使用方法の詳細については、Working with On-Premises Instances を参照してください。

オンプレミスインスタンスをデプロイでそれ以上使用したくない場合は、デプロイグループからオンプレミスインスタンスタグを削除できます。より強力な方法としては、インスタンスからオンプレミスインスタンスタグを削除します。明示的にオンプレミスインスタンスを登録解除し、デプロイでそれ以上使用されないようにすることもできます。詳細については、「CodeDeploy でのオンプレミスインスタンスのオペレーションの管理」を参照してください。

このセクションの手順では、デプロイで使用できるようにオンプレミスインスタンスを設定し、CodeDeploy を用いて登録し、タグ付けする方法を学びます。また、このセクションでは、デプロイする計画がなくなった後に、CodeDeploy を使用して、オンプレミスインスタンスについての情報を取得する情報や、オンプレミスインスタンスの登録を解除する方法を説明します。

トピック

- オンプレミスインスタンスを設定するための前提条件
- CodeDeploy を用いてへのオンプレミスインスタンスを登録
- CodeDeploy でのオンプレミスインスタンスのオペレーションの管理

オンプレミスインスタンスを設定するための前提条件

オンプレミスインスタンスを登録するには、次の前提条件を満たす必要があります。

> **Important**
>
> register-on-premises-instance コマンドを使用していて、 AWS Security Token Service (AWS STS) で生成された一時的な認証情報を定期的に更新する場合は、他の前提条件があります。詳細については、「IAM セッション ARN 登録前提条件」を参照してください。

デバイスの要件

CodeDeploy を用いてオンプレミスインスタンスとして準備し、登録し、タグ付けするデバイスは、サポートされている OS を実行している必要があります。リストについては、「CodeDeploy エージェントで対応するオペレーティングシステム」を参照してください。

OS がサポートされていない場合、ニーズに適合するために CodeDeploy エージェントをオープンソースとして利用できます。さらなる詳細については、GitHub の CodeDeploy agent リポジトリを参照してください。

アウトバウンド通信

オンプレミスインスタンスは、CodeDeploy と通信するためにパブリック AWS サービスエンドポイントに接続できる必要があります。

CodeDeploy エージェントは、ポート 443 経由で HTTPS を使用してアウトバウンドの通信をします。

管理コントロール

オンプレミスインスタンスの設定のためにオンプレミスインスタンス上で使用するローカルまたはネットワークのアカウントは、sudo あるいは `root` (Ubuntu サーバーの場合)、または管理者 (Windows サーバーの場合) として実行できる必要があります。

IAM アクセス許可

オンプレミスインスタンスを登録するために使用する IAM アイデンティティは、登録を完了するため(および必要に応じて登録を削除するため)の許可を付与されている必要があります。

「ステップ 3: CodeDeploy ユーザーのアクセス許可を制限する」で説明されているポリシーに加えて、呼び出し元の IAM アイデンティティに以下の追加のポリシーが添付済みであることを確認します。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iam:CreateAccessKey",
        "iam:CreateUser",
        "iam:DeleteAccessKey",
        "iam:DeleteUser",
        "iam:DeleteUserPolicy",
        "iam:ListAccessKeys",
        "iam:ListUserPolicies",
        "iam:PutUserPolicy",
        "iam:GetUser"
      ],
      "Resource": "*"
    }
  ]
}
```

IAM ポリシーをアタッチする方法については、IAM ポリシーの管理を参照してください。

CodeDeploy を用いてへのオンプレミスインスタンスを登録

オンプレミスインスタンスを登録するには、リクエストを認証するために IAM ID を使用する必要があります。使用する IAM ID と登録方法を以下のオプションから選択できます。

- リクエストを認証するため、IAM ロール ARN を使用します。
  - register-on-premises-instance コマンドを使用し、 AWS Security Token Service (AWS STS) で生成された一時的な認証情報を定期的に更新して、ほとんどの登録オプションを手動で設定します。認証はタイムアウトする一時的なトークンを使用して行われ、定期的に更新する必要があるため、このオプションは最高レベルのセキュリティを提供します。このオプションは、どのような規模の実稼働向けデプロイにも推奨されます。詳細については、「[register-on-premises-instance] コマンド (IAM セッション ARN) を使用してオンプレミスインスタンスを登録」を参照してください。
- (非推奨) リクエストを認証するため、IAM ユーザー ARN を使用します。

- 最も自動化された登録プロセスのために、register コマンドを使用します。このオプションは、セキュリティがそれほど問題にならない実稼働以外のデプロイでのみ使用してください。このオプションは認証に静的 (永続的) 認証情報を使用するため、安全性が低くなります。このオプションは、単一のオンプレミスインスタンスを登録する場合に適しています。詳細については、「オンプレミスインスタンスを登録するために登録コマンド (IAM ユーザー ARN) を使用」を参照してください。
- register-on-premises-instance コマンドを使って、ほとんどの登録オプションを手動で設定します。少数のオンプレミスインスタンスを登録するのに適しています。詳細については、「[register-on-premises-instance] コマンド (IAM ユーザー ARN) を使用してオンプレミスインスタンスを登録します。」を参照してください。

トピック

- [register-on-premises-instance] コマンド (IAM セッション ARN) を使用してオンプレミスインスタンスを登録
- オンプレミスインスタンスを登録するために登録コマンド (IAM ユーザー ARN) を使用
- [register-on-premises-instance] コマンド (IAM ユーザー ARN) を使用してオンプレミスインスタンスを登録します。

[register-on-premises-instance] コマンド (IAM セッション ARN) を使用してオンプレミスインスタンスを登録

オンプレミスインスタンスの認証と登録を最大限制御するには、register-on-premises-instance コマンドと AWS Security Token Service 、 () で生成された一時的な認証情報を定期的に更新します AWS STS。インスタンスの静的 IAM ロールは、CodeDeploy デプロイオペレーションを実行するために、これらの更新された AWS STS 認証情報のロールを引き受けます。

多数のインスタンスを登録する必要がある場合は、このメソッドが最も役に立ちます。これにより、CodeDeploy での登録処理を自動化できます。自分の ID と認証システムを使用して、オンプレミスインスタンスを認証でき、CodeDeploy で使用するためにインスタンス IAM セッション認証情報をサービスからインスタンスへ配布できます。

> **Note**
>
> または、すべてのオンプレミスインスタンスに配布された共有 IAM ユーザーを使用して AWS STS AssumeRole API を呼び出して、オンプレミスインスタンスのセッション認証情

報を取得することもできます。このメソッドは安全性が低いため、本番稼働用またはミッションクリティカルな環境での使用は推奨しません。

次のトピックの情報を使用して、 で生成された一時的なセキュリティ認証情報を使用してオンプレミスインスタンスを設定します AWS STS。

トピック

- IAM セッション ARN 登録前提条件
- ステップ 1: オンプレミスインスタンスが引き受ける IAM ロールを作成
- ステップ 2: を使用して個々のインスタンスの一時的な認証情報を生成する AWS STS
- ステップ 3: オンプレミスインスタンスに設定ファイルを追加
- ステップ 4:CodeDeploy デプロイのためオンプレミスインスタンスを準備します。
- ステップ 5: CodeDeploy でオンプレミスインスタンスを登録します。
- ステップ 6: オンプレミスインスタンスにタグ付け
- ステップ 7: アプリケーションリビジョンをオンプレミスインスタンスにデプロイ
- ステップ 8: オンプレミスインスタンスへのデプロイを追跡

IAM セッション ARN 登録前提条件

オンプレミスインスタンスを設定するための前提条件 にリストされている前提条件に加えて、次の追加の要件を満たす必要があります。

IAM アクセス許可

オンプレミスインスタンスを登録するのに使用する IAM 識別子には、CodeDeploy オペレーションを実行するためのアクセス権限が付与される必要があります。AWSCodeDeployFullAccess 管理ポリシーが IAM ID に添付されていることを確認します。詳細については、IAM ユーザーガイドの AWS マネージドポリシー を参照してください。

一時的な認証情報を更新するシステム

IAM セッション ARN を使用してオンプレミスインスタンスを登録する場合、一時的な認証情報を定期的に更新する適切なシステムが必要です。一時的な認証情報は 1 時間後、または認証情報が生成されたときより短い時間が指定されていればそれより早く期限切れになります。認証情報を更新するためのメソッドは 2 つあります。

- メソッド 1: 企業ネットワーク内で ID および認証システムを適切に使用し、CRON スクリプトを使って ID および認証システムを定期的にポーリングし、最新のセッション認証情報をインスタンスへコピーするようにします。これにより、組織で使用する認証タイプをサポートするために CodeDeploy エージェントまたはサービスを変更 AWS することなく、認証とアイデンティティ構造を と統合できます。
- 方法 2: インスタンスで定期的に CRON ジョブを実行して AWS STS AssumeRole アクションを呼び出し、CodeDeploy エージェントがアクセスできるファイルにセッション認証情報を書き込みます。このメソッドでは、IAM ユーザーの使用、およびオンプレミスインスタンスへの認証情報のコピーをする必要はありますが、多くのオンプレミスインスタンスで同じ IAM ユーザーおよび認証情報を使用できます。

AWS STS 認証情報の作成と操作の詳細については、AWS Security Token Service 「 API リファレンス」および「一時的なセキュリティ認証情報を使用して AWS リソースへのアクセスをリクエストする」を参照してください。

ステップ 1: オンプレミスインスタンスが引き受ける IAM ロールを作成

AWS CLI または IAM コンソールを使用して、CodeDeploy を認証および操作するためにオンプレミスインスタンスで使用される IAM ロールを作成できます。

単一の IAM ロールのみを作成する必要があります。各オンプレミスインスタンスは、このロールに付与されたアクセス権限を提供する一時認証情報を取得するためにこのロールを引き受けることができます。

作成するロールは、CodeDeploy エージェントをインストールするのに必要なファイルにアクセスするために、次の権限が必要です。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:Get*",
                "s3:List*"
            ],
            "Effect": "Allow",
            "Resource": "*"
```

```
        }
    ]
}
```

このポリシーを、オンプレミスインスタンスがアクセスする必要のある Amazon S3 バケットにのみ制限することをお勧めします。このポリシーを制限する場合、CodeDeploy エージェントを含む Amazon S3 バケットへのアクセスを許可することを確認します。そうしない場合、CodeDeploy エージェントがオンプレミスインスタンスにインストールされる、または更新されるたびに、エラーが発生する可能性があります。Amazon S3 リソースへのアクセスコントロールの詳細については、Managing access permissions to your Amazon S3 resources を参照してください。

IAM ロールを作成するには

1. オプションを使用して create-role `--role-name` コマンドを呼び出し、IAM ロールの名前 (例: `CodeDeployInstanceRole`) と `--assume-role-policy-document` オプションを指定してアクセス権限を提供します。

   このインスタンスの IAM ロールを作成するときは、`CodeDeployInstanceRole` という名前を付け、`CodeDeployRolePolicy.json` という名前のファイルに必要なアクセス権限を含めます。

   ```
   aws iam create-role --role-name CodeDeployInstanceRole --assume-role-policy-
   document file://CodeDeployRolePolicy.json
   ```

2. create-role コマンドを呼び出した出力で、ARN フィールドの値をメモします。例えば、次のようになります。

   ```
   arn:aws:iam::123456789012:role/CodeDeployInstanceRole
   ```

   AWS STS AssumeRole API を使用して各インスタンスの短期認証情報を生成する場合は、ロール ARN が必要です。

   IAM ロールの作成の詳細については、IAM ユーザーガイドの「AWS サービスにアクセス許可を委任するロールの作成」を参照してください。

   既存のロールにアクセス権限を割り当てる方法については、AWS CLI Command Reference の put-role-policy を参照してください。

ステップ 2: を使用して個々のインスタンスの一時的な認証情報を生成する AWS STS

オンプレミスインスタンスの登録に使用する一時認証情報を生成する前に、一時認証情報を生成する IAM ID (ユーザーまたはロール) を作成または選択する必要があります。`sts:AssumeRole` アクセス権限は、この IAM ID のポリシーの設定に含める必要があります。

IAM ID にアクセス`sts:AssumeRole`許可を付与する方法については、「AWS サービスにアクセス許可を委任するロールの作成」および`AssumeRole`」を参照してください。

一時認証情報を生成するには、2 とおりの方法があります。

- AWS CLIを用いて assume-role コマンドを使用します。例えば、次のようになります。

```
aws sts assume-role --role-arn arn:aws:iam::12345ACCOUNT:role/role-arn --role-
session-name session-name
```

  コードの説明は以下のとおりです。

  - *12345ACCOUNT* が組織の 12 桁のアカウント番号です。
  - *role-arn* は、ステップ 1: オンプレミスインスタンスが引き受ける IAM ロールを作成 で生成した引き受けるロールの ARN です。
  - *session-name* は、現在作成しているロールセッションへ付ける名前です。

  > **Note**
  >
  > ID と認証システムを定期的にポーリングし、最新のセッション認証情報をインスタンスにコピーする CRON スクリプトを使用する場合 (「」で説明されている一時的な認証情報を更新するための方法 1IAM セッション ARN 登録前提条件)、代わりにサポートされている任意の AWS SDK を使用して `AssumeRole` を呼び出すことができます。

- が提供するツールを使用します AWS。

  aws-codedeploy-session-helper ツールは AWS STS 認証情報を生成し、インスタンスに配置するファイルに書き込みます。このツールは、IAM セッション ARN 登録前提条件 で説明している一時認証情報を更新するメソッド 2 に最適です。この方法では、aws-codedeploy-session-helper ツールは各インスタンスに配置され、IAM ユーザーのアクセス許可を使用してコマンドを実行します。各インスタンスは、このツールと組み合わせて同じ IAM ユーザーの認証情報を使用します。

  詳細については、aws-codedeploy-session-helper GitHub リポジトリを参照してください。

> **Note**
>
> IAM セッション認証情報を作成した後、オンプレミスインスタンスの任意の場所に保存します。次のステップで、CodeDeploy エージェントがこの場所の認証情報にアクセスできるように設定します。

続ける前に、定期的に一時認証情報を更新するために使用するシステムを確認します。一時認証情報が更新されていない場合、オンプレミスインスタンスへのデプロイは失敗します。詳細については、IAM セッション ARN 登録前提条件 にある「一時認証情報を更新するシステム」を参照してください。

ステップ 3: オンプレミスインスタンスに設定ファイルを追加

ルートまたは管理者権限を使用して、オンプレミスインスタンスに設定ファイルを追加します。この設定ファイルは、CodeDeploy に使用される IAM 認証情報とターゲット AWS リージョンを宣言するために使用されます。ファイルは、オンプレミスインスタンスの指定の場所に追加する必要があります。ファイルには、IAM 一時セッション ARN、そのシークレットキー ID とシークレットアクセスキー、およびターゲット AWS リージョンが含まれている必要があります。

設定ファイルを追加するには

1. オンプレミスインスタンスの以下の場所に、`codedeploy.onpremises.yml` (Ubuntu サーバーまたは RHEL オンプレミスインスタンスの場合)、または、`conf.onpremises.yml` (Windows サーバーオンプレミスインスタンスの場合) という名前のファイルを作成します。

   - Ubuntu サーバーの場合:`/etc/codedeploy-agent/conf`
   - Windows サーバーについて：`C:\ProgramData\Amazon\CodeDeploy`

2. テキストエディタを使用して、新しく作成した `codedeploy.onpremises.yml` ファイル (Linux) または `conf.onpremises.yml` ファイル (Windows) に次の情報を追加します。

   ```
   ---
   iam_session_arn: iam-session-arn
   aws_credentials_file: credentials-file
   region: supported-region
   ```

   説明は以下のとおりです。

- *iam-session-arn* は、「ステップ 2: を使用して個々のインスタンスの一時的な認証情報を生成する AWS STS」でメモしておいた IAM セッション ARN です。
- *credentials-file* は、ステップ 2: を使用して個々のインスタンスの一時的な認証情報を生成する AWS STS でメモした一時セッション ARN の認証情報ファイルの場所です。
- *supported-region* は、 のリージョンとエンドポイントに記載されているように、CodeDeploy がサポートするリージョンの 1 つですAWS 全般のリファレンス。 https://docs.aws.amazon.com/general/latest/gr/rande.html#codedeploy_region

ステップ 4:CodeDeploy デプロイのためオンプレミスインスタンスを準備します。

のインストールと設定 AWS CLI

オンプレミスインスタンス AWS CLI に をインストールして設定します。(AWS CLI は、CodeDeploy エージェントをダウンロードしてオンプレミスインスタンスにインストールするために使用されます) 。

1. オンプレミスインスタンス AWS CLI に をインストールするには、AWS Command Line Interface 「 ユーザーガイド」の「 のセットアップ方法 AWS CLI」の手順に従います。

> **Note**
>
> オンプレミスインスタンスを使用するための CodeDeploy コマンドは、 AWS CLIのバージョン 1.7.19 で使用できるようになりました。のバージョンが AWS CLI 既にインストールされている場合は、 を呼び出してそのバージョンを確認できますaws --version。

2. オンプレミスインスタンス AWS CLI で を設定するには、AWS Command Line Interface 「 ユーザーガイド」の「 の設定 AWS CLI」の手順に従います。

> **Important**
>
> (aws configure コマンドを呼び出す AWS CLI などして) を設定するときは、少なくともで説明されているアクセス許可を持つ IAM ユーザーのシークレットキー ID とシークレットアクセスキーを必ず指定してくださいIAM セッション ARN 登録前提条件。

AWS_REGION 環境変数を設定する (Ubuntu Server および RHEL のみ)

Ubuntu サーバーまたは RHEL をオンプレミスインスタンスで実行していない場合は、このステップをスキップして「CodeDeploy エージェントをインストールする」へ進んでください。

Ubuntu サーバーまたは RHEL オンプレミスインスタンスに CodeDeploy エージェントをインストールし、新しいバージョンが使用可能になったらいつでも CodeDeploy エージェントを更新するようにインスタンスを有効にします。これを行うには、インスタンスのAWS_REGION環境変数を CodeDeploy でサポートされているいずれかのリージョンの識別子に設定します。値を CodeDeploy アプリケーション、デプロイグループ、およびアプリケーションリビジョンがあるリージョン (など) に設定することをお勧めしますus-west-2。リージョンのリストについては、の「リージョンとエンドポイント」を参照してくださいAWS 全般のリファレンス。

環境変数を設定するには、端末から以下を呼び出します。

```
export AWS_REGION=supported-region
```

supported-region はリージョン識別子です (例: us-west-2)。

CodeDeploy エージェントをインストール

- Ubuntu サーバーオンプレミスインスタンスの場合は、Ubuntu サーバー用の CodeDeploy エージェントをインストールする の手順に従った後、このページに戻ります。
- RHEL オンプレミスインスタンスについては、Amazon Linux または RHEL 用の CodeDeploy エージェントをインストールする の手順に従った後、このページに戻ります。
- Windows サーバーオンプレミスインスタンスの場合は、Windows サーバー用の CodeDeploy エージェントです。 の手順に従った後、このページに戻ります。

ステップ 5: CodeDeploy でオンプレミスインスタンスを登録します。

このステップの手順では、オンプレミスインスタンス自体からオンプレミスインスタンスを登録していることを想定します。オンプレミスインスタンスは、 AWS CLI がインストールされ、設定された別のデバイスまたはインスタンスから登録できます。

AWS CLI を使用してオンプレミスインスタンスを CodeDeploy に登録し、デプロイで使用できるようにします。

を使用する前に AWS CLI、 で作成した一時セッション認証情報の ARN が必要ですステップ 3: オンプレミスインスタンスに設定ファイルを追加。例えば、AssetTag12010298EX と指定したインスタンスの場合:

```
arn:sts:iam::123456789012:assumed-role/CodeDeployInstanceRole/AssetTag12010298EX
```

register-on-premises-instance コマンドを呼び出し、以下を指定します。

- オンプレミスインスタンスを一意に識別する名前 (--instance-name オプションで指定)。

> **Important**
>
> オンプレミスインスタンスを識別するために、特にデバッグのため、オンプレミスインスタンスの一意な特徴を示す名前 (例えば、もしあれば、STS 認証情報の session-name とシリアルナンバー、または内部アセット識別子など) を指定することを強くお勧めします。名前として MAC アドレスを指定した場合、MAC アドレスにはコロン (:) など CodeDeploy が許可しない文字が含まれることに注意してください。許可された文字の一覧については、「CodeDeploy のクォータ」を参照してください。

- 複数のオンプレミスインスタンスを認証するために ステップ 1: オンプレミスインスタンスが引き受ける IAM ロールを作成 で設定した IAM セッション ARN。

例:

```
aws deploy register-on-premises-instance --instance-name name-of-instance --iam-
session-arn arn:aws:sts::account-id:assumed-role/role-to-assume/session-name
```

コードの説明は以下のとおりです。

- *name-of-instance* は、オンプレミスインスタンスを識別するのに使用する名前です (例:AssetTag12010298EX)。
- *account-id* は、組織の 12 桁のアカウント IDです (例: 111222333444)。
- *role-to-assume* は、インスタンス用に作成した IAM ロールの名前です (例: CodeDeployInstanceRole)。
- *session-name* は、ステップ 2: を使用して個々のインスタンスの一時的な認証情報を生成する AWS STS で指定したセッションロールの名前です。

ステップ 6: オンプレミスインスタンスにタグ付け

AWS CLI または CodeDeploy コンソールを使用して、オンプレミスインスタンスにタグを付けることができます。(CodeDeploy はオンプレミスインスタンスタグを使用してデプロイ中にデプロイターゲットを識別します。)

オンプレミスインスタンスにタグ付けするには (CLI)

- add-tags-to-on-premises-instances コマンドを呼び出し、以下を指定します。
  - オンプレミスインスタンスを一意に識別する名前 (`--instance-names` オプションで指定)。
  - 使用するオンプレミスインスタンスのタグキーの名前とタグ値 (`--tags` オプションで指定)。名前と値はいずれも指定する必要があります。CodeDeploy は値のみがあるオンプレミスインスタンスタグを許可しません。

    例:

```
aws deploy add-tags-to-on-premises-instances --instance-names AssetTag12010298EX
 --tags Key=Name,Value=CodeDeployDemo-OnPrem
```

オンプレミスインスタンスにタグ付けするには (コンソール)

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで、Deploy を展開し、On-premises instances を選択します。
3. オンプレミスインスタンスのリストで、タグ付けするオンプレミスインスタンスの名前を選択します。
4. タグのリストで、目的のタグキーとタグ値を選択または入力します。タグキーとタグ値を入力するたびに、別の行が表示されます。最大 10 個のタグにこれを繰り返すことができます。タグを削除するには[削除] を選択してください。
5. タグを追加したら、[Update Tags] を選択します。

ステップ 7: アプリケーションリビジョンをオンプレミスインスタンスにデプロイ

登録され、タグ付けされたオンプレミスインスタンスにアプリケーションリビジョンをデプロイする準備ができました。

Amazon EC2 インスタンスにアプリケーションリビジョンをデプロイするのと同様の方法でオンプレミスインスタンスにアプリケーションリビジョンをデプロイします。手順については、「CodeDeploy でデプロイを作成する」を参照してください。これらの指示には、アプリケーションの作成、開発グループの作成、およびアプリケーションリビジョンの準備を含む前提条件へのリンクが含まれています。シンプルなサンプルアプリケーションリビジョンをデプロイすることが必要な場合は、チュートリアル: CodeDeploy (Windows サーバー、Ubuntu サーバー、または Red Hat エンタープライズ Linux) を使用してオンプレミスインスタンスにアプリケーションをデプロイします。の ステップ 2: サンプルのアプリケーションリビジョンを作成する で説明してあるものを作成できます。

> **Important**
>
> オンプレミスインスタンスを対象としたデプロイグループの作成の一部として、CodeDeploy サービスロールを再利用する場合は、`Tag:get*` をサービスロールのポリシーステートメントの `Action` の部分に含める必要があります。詳細については、「ステップ 2: CodeDeployのサービスのロールを作成する」を参照してください。

ステップ 8: オンプレミスインスタンスへのデプロイを追跡

登録されタグ付けされたオンプレミスインスタンスへアプリケーションリビジョンをデプロイした後、デプロイの進行状況を追跡できます。

Amazon EC2 インスタンスへのデプロイの追跡と同様の方法でオンプレミスインスタンスへのデプロイの追跡をします。手順については、「CodeDeploy デプロイの詳細を表示する」を参照してください。

オンプレミスインスタンスを登録するために登録コマンド (IAM ユーザー ARN) を使用

> **Important**
>
> IAM ユーザーを使用してインスタンスを登録することは、認証に静的 (永続的) 認証情報を使用するため推奨されません。セキュリティ向上のため、認証には一時的な認証情報を使用し

てインスタンスを登録することをお勧めします。詳細については、「[register-on-premises-instance] コマンド (IAM セッション ARN) を使用してオンプレミスインスタンスを登録」を参照してください。

Important

IAM ユーザーのアクセスキー (永続的認証情報) をローテーションする計画を立ててください。アクセスキーのローテーションの詳細については、「アクセスキーの更新」を参照してください。

このセクションでは、最低限の労力でオンプレミスインスタンスを CodeDeploy で設定してタグ付けする方法を説明します。register コマンドは、単一の、または少数のオンプレミスインスタンスを処理する際に最も有用です。register のコマンドは、インスタンスを認証するために IAM ユーザー ARN を使用している場合のみ使用できます。register のコマンドは、認証のための IAM セッション ARN とは共に使用することはできません。

register のコマンドを使用すると、CodeDeploy で次の操作ができます。

- コマンドで IAM ユーザーを指定しない場合は、オンプレミスインスタンス AWS Identity and Access Management の に IAM ユーザーを作成します。
- オンプレミスインスタンスの設定ファイルに IAM ユーザーの認証情報を保存します。
- CodeDeploy でオンプレミスインスタンスを登録します。
- コマンドの一部にタグを指定した場合、オンプレミスインスタンスにタグを追加します。

Note

register-on-premises-instance コマンドは、register コマンドの代わりです。CodeDeploy を用いてオンプレミスインスタンスの設定、登録、タグ付けのほとんどを自分で行いたい場合は、register-on-premises-instance のコマンドを使用します。また、register-on-premises-instance のコマンドを使うと、IAM ユーザー ARN の代わりに、IAM セッション ARN を使用してインスタンスを登録できます。このアプローチは、大量のオンプレミスインスタンスがある場合、大きな利点となります。具体的には、各オンプレミスインスタンスに 1 つずつ IAM ユーザーを作成するかわりに、単一の IAM セッション ARN を使用して複数のインスタンスを認証できます。詳細については、「[register-on-premises-instance] コマンド (IAM

ユーザー ARN) を使用してオンプレミスインスタンスを登録します。」および「[register-on-premises-instance] コマンド (IAM セッション ARN) を使用してオンプレミスインスタンスを登録」を参照してください。

トピック

- ステップ 1: オンプレミスインスタンス AWS CLI に をインストールして設定する
- ステップ 2: 登録コマンドを呼び出す
- ステップ 3: インストールコマンドを呼び出す
- ステップ 4: デプロイアプリケーションリビジョンをオンプレミスインスタンスにデプロイする
- ステップ 5: オンプレミスインスタンスへのデプロイを追跡

ステップ 1: オンプレミスインスタンス AWS CLI に をインストールして設定する

1. オンプレミスインスタンス AWS CLI に をインストールします。 AWS CLIユーザーガイド の Getting set up with the AWS Command Line Interface の指示に従います。

   > **Note**
   >
   > オンプレミスインスタンスを操作するための CodeDeploy コマンドは、 AWS CLI バージョン 1.7.19 以降で使用できます。 AWS CLI が既にインストールされている場合は、を呼び出しaws --versionてバージョンを確認します。

2. オンプレミスインスタンス AWS CLI で を設定します。 AWS CLIユーザーガイド の Configuring the AWS Command Line Interface の指示に従います。

   > **Important**
   >
   > (aws configure コマンドを呼び出す AWS CLI などして) を設定するときは、 で指定されたアクセス許可に加えて、少なくとも次の AWS アクセス許可を持つ IAM ユーザーのシークレットキー ID とシークレットアクセスキーを必ず指定してくださいオンプレミスインスタンスを設定するための前提条件。これにより、オンプレミスインスタンスで CodeDeploy エージェントをダウンロードしてインストールすることができるようになります。アクセス権限は次のようになります。

JSON

```
{
  "Version":"2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "codedeploy:*",
        "iam:CreateAccessKey",
        "iam:CreateUser",
        "iam:DeleteAccessKey",
        "iam:DeleteUser",
        "iam:DeleteUserPolicy",
        "iam:ListAccessKeys",
        "iam:ListUserPolicies",
        "iam:PutUserPolicy",
        "iam:GetUser",
        "tag:getTagKeys",
        "tag:getTagValues",
        "tag:GetResources"
      ],
      "Resource" : "*"
    },
    {
      "Effect" : "Allow",
      "Action" : [
        "s3:Get*",
        "s3:List*"
      ],
      "Resource" : [
        "arn:aws:s3:::amzn-s3-demo-bucket/*",
        "arn:aws:s3:::amzn-s3-demo-bucket1/*"
      ]
    }
  ]
}
```

> **Note**
>
> 前述の Amazon S3 バケットのいずれかにアクセスしようとしたときにアクセス拒否エラーが表示される場合は、バケットのリソース ARN の `/*` の部分 (例: `arn:aws:s3:::aws-codedeploy-sa-east-1`) を省略してみてください。

JSON

```
{
  "Version":"2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "codedeploy:*",
        "iam:CreateAccessKey",
        "iam:CreateUser",
        "iam:DeleteAccessKey",
        "iam:DeleteUser",
        "iam:DeleteUserPolicy",
        "iam:ListAccessKeys",
        "iam:ListUserPolicies",
        "iam:PutUserPolicy",
        "iam:GetUser",
        "tag:GetResources"
      ],
      "Resource" : "*"
    },
    {
      "Effect" : "Allow",
      "Action" : [
        "s3:Get*",
        "s3:List*"
      ],
      "Resource" : [
        "arn:aws:s3:::amzn-s3-demo-bucket/*",
        "arn:aws:s3:::amzn-s3-demo-bucket2/*"
      ]
    }
```

```
  ]
}
```

ステップ 2: 登録コマンドを呼び出す

このステップでは、オンプレミスインスタンス自体からオンプレミスインスタンスを登録していることを想定します。前のステップで説明したように、 AWS CLI がインストールされ、設定された別のデバイスまたはインスタンスからオンプレミスインスタンスを登録することもできます。

を使用して register コマンドを AWS CLI 呼び出し、以下を指定します。

- CodeDeploy に対してオンプレミスインスタンスを一意に識別する名前 (`--instance-name` のオプションを用いて)。

  > **Important**
  >
  > 後でオンプレミスインスタンスを識別するために、特にデバッグのため、オンプレミスインスタンスの一意な特徴を示す名前 (例えば、もしあれば、シリアルナンバーや一意の内部アセット識別子など) を使用することを強くお勧めします。名前に MAC アドレスを指定した場合、MAC アドレスにはコロン (:) など、CodeDeploy が許可しない文字が含まれることに注意してください。許可された文字の一覧については、「CodeDeploy のクォータ」を参照してください。

- 必要に応じて、このオンプレミスインスタンスと関連付ける既存の IAM ユーザーの ARN (`--iam-user-arn` のオプションを用いて) IAM ユーザーの ARN を取得するには、get-user コマンドを呼び出すか、IAM コンソールの [ユーザー] セクションで IAM ユーザー名を選択した後、[サマリー] セクションで [ユーザー ARN] の値を見つけます。このオプションを指定しない場合、CodeDeploy はユーザーに代わって AWS アカウント内に IAM ユーザーを作成し、オンプレミスインスタンスに関連付けます。

  > **Important**
  >
  > `--iam-user-arn` オプションを指定する場合は、「ステップ 4: オンプレミスインスタンスに設定ファイルを追加」の説明にあるとおり、オンプレミスインスタンスの設定ファイルを手動で作成することも必要です。
  > 1 つのオンプレミスインスタンスのみに対し、1 人の IAM ユーザーのみを関連付けることができます。複数のオンプレミスインスタンスに 1 人の IAM ユーザーを関連付けようとす

ると、エラー、オンプレミスインスタンスへのデプロイの失敗、またはオンプレミスインスタンスへのデプロイが無期限の保留状態のままとなります。

- 必要に応じて、デプロイ先の Amazon EC2 インスタンスのセットを識別するために CodeDeploy が使用するオンプレミスインスタンスタグのセット (`--tags` の オプションを用いて)。各タグを `Key=tag-key,Value=tag-value` で指定します (例: `Key=Name,Value=Beta Key=Name,Value=WestRegion`)。このオプションを指定しない場合、タグは登録されません。後でタグを登録するには、add-tags-to-on-premises-instances コマンドを呼び出します。
- オプションで、オンプレミスインスタンスが CodeDeploy に登録される AWS リージョン (`--region`オプションを使用) 。これは、 のリージョンとエンドポイントにリストされているサポートされているリージョンの 1 つである必要があります AWS 全般のリファレンス (例: `us-west-2`)。このオプションを指定しない場合、呼び出し元の IAM ユーザーに関連付けられたデフォルトの AWS リージョンが使用されます。

例えば、次のようになります。

```
aws deploy register --instance-name AssetTag12010298EX --iam-user-
arn arn:aws:iam::444455556666:user/CodeDeployUser-OnPrem --tags
 Key=Name,Value=CodeDeployDemo-OnPrem --region us-west-2
```

register コマンドは次のことを行います。

1. 既存の IAM ユーザーを指定しない場合、IAM ユーザーを作成して必要なアクセス権限を付与し、対応するシークレットキーおよびシークレットアクセスキーを生成します。オンプレミスインスタンスは、この IAM ユーザーとアクセス権限および認証情報を使用して CodeDeploy との認証および操作を行います。
2. CodeDeploy でオンプレミスインスタンスを登録します。
3. 指定されている場合には、CodeDeploy 中で、`--tags` のオプションで指定したタグと、登録済みのオンプレミスインスタンスの名前を関連付けます。
4. IAM ユーザーが作成されている場合、register のコマンドの呼び出し元と同じディレクトリに必要な設定ファイルも作成します。

このコマンドでエラーが発生した場合、エラーメッセージが表示され、手動で残りのステップを完了する方法について説明します。そうでない場合は、成功メッセージが表示され、次のステップに示すとおり、install コマンドを呼び出す方法について説明します。

ステップ 3: インストールコマンドを呼び出す

オンプレミスインスタンスから、 AWS CLI を使用して install コマンドを呼び出し、以下を指定します。

- 設定ファイルへのパス (`--config-file` オプションで指定)。
- 必要に応じて、オンプレミスインスタンスにある既存の設定ファイルを置き換えるかどうか (`--override-config` オプションで指定)。指定しない場合、既存の設定ファイルは置き換えられません。
- オプションで、オンプレミスインスタンスが CodeDeploy に登録される AWS リージョン (`--region`オプションを使用) 。これは、 のリージョンとエンドポイントにリストされているサポートされているリージョンの 1 つである必要があります AWS 全般のリファレンス (例: `us-west-2`)。このオプションを指定しない場合、呼び出し元の IAM ユーザーに関連付けられたデフォルトの AWS リージョンが使用されます。
- 必要に応じて、CodeDeploy エージェントのインストール元のカスタムロケーション (`--agent-installer` のオプションを用いて)。このオプションは、CodeDeploy が公式にはサポートしていない CodeDeploy エージェント のカスタムバージョンのインストールに使用できます (GitHub の CodeDeploy エージェント リポジトリに基づくカスタムバージョンなど)。値は、次のいずれかを含む Amazon S3 バケットへのパスである必要があります。
  - CodeDeploy エージェントのインストールスクリプト (GitHub の CodeDeploy agent エージェントリポジトリにあるインストールファイルと同様、Linux または Unix ベースの OS 用)。
  - CodeDeploy エージェントのインストーラパッケージ (.msi) ファイル (Windows ベースの OS 用)。

  このオプションを指定しない場合、CodeDeploy は独自の場所から、オンプレミスインスタンス上の OS と互換性のある正式にサポートされているバージョンの CodeDeploy エージェントをインストールしようとします。

例:

```
aws deploy install --override-config --config-file /tmp/codedeploy.onpremises.yml --
region us-west-2 --agent-installer s3://aws-codedeploy-us-west-2/latest/codedeploy-
agent.msi
```

install コマンドは次のことを行います。

1. オンプレミスインスタンスが Amazon EC2 インスタンスかどうかを確認します。そうである場合は、エラーメッセージが表示されます。
2. オンプレミスインスタンスの設定ファイルを、インスタンスの指定された場所から CodeDeploy エージェントが見つけやすい場所へ (まだその場所にない場合) コピーします。

   Ubuntu サーバーおよび Red Hat Enterprise Linux (RHEL)の場合、これは `/etc/codedeploy-agent/conf/codedeploy.onpremises.yml` になります。

   Windows サーバーの場合、これは `C:\ProgramData\Amazon\CodeDeploy\conf.onpremises.yml` になります。

   `--override-config` オプションを指定した場合は、ファイルを作成または上書きします。
3. オンプレミスインスタンスに CodeDeploy エージェントをインストールし、起動します。

ステップ 4: デプロイアプリケーションリビジョンをオンプレミスインスタンスにデプロイする

登録され、タグ付けされたオンプレミスインスタンスにアプリケーションリビジョンをデプロイする準備ができました。

Amazon EC2 インスタンスにアプリケーションリビジョンをデプロイするのと同様の方法でオンプレミスインスタンスにアプリケーションリビジョンをデプロイします。手順については、「CodeDeploy でデプロイを作成する」を参照してください。これらの指示は、アプリケーションの作成、開発グループの作成、およびアプリケーションリビジョンの準備を含む前提条件と関連しています。シンプルなサンプルアプリケーションリビジョンをデプロイすることが必要な場合は、チュートリアル: CodeDeploy (Windows サーバー、Ubuntu サーバー、または Red Hat エンタープライズ Linux) を使用してオンプレミスインスタンスにアプリケーションをデプロイします。 の ステップ 2: サンプルのアプリケーションリビジョンを作成する で説明してあるものを作成できます。

> **Important**
>
> オンプレミスインスタンスを対象としたデプロイグループの作成の一部として、既存の CodeDeploy サービスロールを再利用する場合は、`Tag:get*` をサービスロールのポリシーステートメントの `Action` の部分に含める必要があります。詳細については、「ステップ 2: CodeDeployのサービスのロールを作成する」を参照してください。

ステップ 5: オンプレミスインスタンスへのデプロイを追跡

登録されタグ付けされたオンプレミスインスタンスへアプリケーションリビジョンをデプロイした後、デプロイの進行状況を追跡できます。

Amazon EC2 インスタンスへのデプロイの追跡と同様の方法でオンプレミスインスタンスへのデプロイの追跡をします。手順については、「CodeDeploy デプロイの詳細を表示する」を参照してください。

他のオプションについては、「CodeDeploy でのオンプレミスインスタンスのオペレーションの管理」を参照してください。

[register-on-premises-instance] コマンド (IAM ユーザー ARN) を使用してオンプレミスインスタンスを登録します。

> **Important**
>
> IAM ユーザーを使用してインスタンスを登録することは、認証に静的 (永続的) 認証情報を使用するため推奨されません。セキュリティ向上のため、認証には一時的な認証情報を使用してインスタンスを登録することをお勧めします。詳細については、「[register-on-premises-instance] コマンド (IAM セッション ARN) を使用してオンプレミスインスタンスを登録」を参照してください。

> **Important**
>
> IAM ユーザーのアクセスキー (永続的認証情報) をローテーションする計画を立ててください。アクセスキーのローテーションの詳細については、「アクセスキーの更新」を参照してください。

認証に固定 IAM ユーザー認証情報を使って、CodeDeploy でオンプレミスインスタンスの設定、登録、タグ付けをほとんど自分で行う場合は、これらの手順に従います。

トピック

- ステップ 1: オンプレミスインスタンス用に IAM ユーザーを作成する
- ステップ 2: ユーザーにアクセス権限を割り当てる
- ステップ 3: ユーザーの認証情報を取得する

- ステップ 4: オンプレミスインスタンスに設定ファイルを追加
- ステップ 5: をインストールして設定する AWS CLI
- ステップ 6: AWS_REGION 環境変数を設定する (Ubuntu Server および RHEL のみ)
- ステップ 7: CodeDeploy エージェントのインストール
- ステップ 8: CodeDeploy でオンプレミスインスタンスを登録します。
- ステップ 9: オンプレミスインスタンスにタグ付けする
- ステップ 10: アプリケーションリビジョンをオンプレミスインスタンスにデプロイする
- ステップ 11: オンプレミスインスタンスへのデプロイを追跡する

ステップ 1: オンプレミスインスタンス用に IAM ユーザーを作成する

オンプレミスインスタンスが認証や CodeDeploy とのやり取りに使用する IAM ユーザーを作成します。

> **Important**
>
> それぞれの参加しているオンプレミスインスタンスに対して個別の IAM ユーザーを作成する必要があります。複数のオンプレミスインスタンスに対して単独の IAM ユーザーを再利用しようとすると、CodeDeploy を用いての、オンプレミスインスタンスの登録やタグ付けができない場合があります。それらのオンプレミスインスタンスへのデプロイは、無期限の保留状態のまま、または完全にエラーとなる場合があります。

IAM ユーザーに、目的を識別するために、[CodeDeployUser-OnPrem] のような名前を割り当てることをお勧めします。

AWS CLI または IAM コンソールを使用して IAM ユーザーを作成できます。詳細については、「Creating an IAM user in your AWS account を参照してください。

> **Important**
>
> AWS CLI または IAM コンソールを使用して新しい IAM ユーザーを作成するかどうかにかかわらず、ユーザーに提供されたユーザー ARN を書き留めます。この情報は後に ステップ 4: オンプレミスインスタンスに設定ファイルを追加 と ステップ 8: CodeDeploy でオンプレミスインスタンスを登録します。 で必要になります。

ステップ 2: ユーザーにアクセス権限を割り当てる

オンプレミスインスタンスが Amazon S3 バケットからアプリケーションリビジョンをデプロイする場合、IAM ユーザーがバケットとやり取りするための権限を割り当てる必要があります。 AWS CLI または IAM コンソールを使用してアクセス許可を割り当てることができます。

> **Note**
>
> GitHub リポジトリからのみアプリケーションリビジョンをデプロイする場合は、このステップを飛ばして ステップ 3: ユーザーの認証情報を取得する へ進んでください。(ステップ 1: オンプレミスインスタンス用に IAM ユーザーを作成する で作成した IAM ユーザーに関する情報はまだ必要です。後のステップで使用します。)

アクセス権限 (CLI) を割り当てるには

1. AWS CLIを呼び出すのに使用している Amazon EC2インスタンスまたはデバイス上に、次のポリシーの内容を用いてファイルを作成します。ファイルに **CodeDeploy-OnPrem-Permissions.json** のような名前を付けて、ファイルを保存します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:Get*",
                "s3:List*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> このポリシーを、オンプレミスインスタンスがアクセスする必要のある Amazon S3 バケットにのみ制限することをお勧めします。このポリシーを制限する場合は、 AWS CodeDeploy エージェントを含む Amazon S3 バケットへのアクセスも許可してください。そうしない場合、CodeDeploy エージェントが関連するオンプレミスインスタンス

にインストールされる、または更新されるたびに、エラーが発生する可能性があります。
例:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:Get*",
        "s3:List*"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket/*",
        "arn:aws:s3:::aws-codedeploy-us-east-2/*",
        "arn:aws:s3:::aws-codedeploy-us-east-1/*",
        "arn:aws:s3:::aws-codedeploy-us-west-1/*",
        "arn:aws:s3:::aws-codedeploy-us-west-2/*",
        "arn:aws:s3:::aws-codedeploy-ca-central-1/*",
        "arn:aws:s3:::aws-codedeploy-eu-west-1/*",
        "arn:aws:s3:::aws-codedeploy-eu-west-2/*",
        "arn:aws:s3:::aws-codedeploy-eu-west-3/*",
        "arn:aws:s3:::aws-codedeploy-eu-central-1/*",
        "arn:aws:s3:::aws-codedeploy-eu-central-2/*",
        "arn:aws:s3:::aws-codedeploy-eu-north-1/*",
        "arn:aws:s3:::aws-codedeploy-eu-south-1/*",
        "arn:aws:s3:::aws-codedeploy-eu-south-2/*",
        "arn:aws:s3:::aws-codedeploy-il-central-1/*",
        "arn:aws:s3:::aws-codedeploy-ap-east-1/*",
        "arn:aws:s3:::aws-codedeploy-ap-northeast-1/*",
        "arn:aws:s3:::aws-codedeploy-ap-northeast-2/*",
        "arn:aws:s3:::aws-codedeploy-ap-northeast-3/*",
        "arn:aws:s3:::aws-codedeploy-ap-southeast-1/*",
        "arn:aws:s3:::aws-codedeploy-ap-southeast-2/*",
        "arn:aws:s3:::aws-codedeploy-ap-southeast-3/*",
        "arn:aws:s3:::aws-codedeploy-ap-southeast-4/*",
        "arn:aws:s3:::aws-codedeploy-ap-south-1/*",
        "arn:aws:s3:::aws-codedeploy-ap-south-2/*",
        "arn:aws:s3:::aws-codedeploy-me-central-1/*",
        "arn:aws:s3:::aws-codedeploy-me-south-1/*",
        "arn:aws:s3:::aws-codedeploy-sa-east-1/*"
      ]
```

```
    }
  ]
}
```

2. put-user-policy コマンドを呼び出し、IAM ユーザーの名前 (`--user-name` のオプションを用いて)、ポリシーの名前 (`--policy-name` のオプションを用いて)、および新しく作成したポリシードキュメントへのパス (`--policy-document` のオプションを用いて) を指定します。例えば、**`CodeDeploy-OnPrem-Permissions.json`** というファイルが、コマンドを呼び出しているのと同じディレクトリ (フォルダ) にあるとします。

> **Important**
>
> ファイル名の前に必ず `file://` を含めてください。このコマンドでは必須です。

```
aws iam put-user-policy --user-name CodeDeployUser-OnPrem --policy-name CodeDeploy-OnPrem-Permissions --policy-document file://CodeDeploy-OnPrem-Permissions.json
```

アクセス権限を割り当てるには (コンソール)

1. https://console.aws.amazon.com/iam/ で IAM コンソール を開きます。
2. ナビゲーションペインで、[Policies] を選択し、[Create Policy] を選択します。([Get Started] ボタンが表示された場合は、そのボタンを選択してから、[Create Policy] を選択します)。
3. [独自のポリシーを作成] の横で、[選択] を選択します。
4. [ポリシー名] ボックスに、このポリシーの名前を入力します (**`CodeDeploy-OnPrem-Permissions`** など)。
5. ポリシードキュメントボックスで、次のアクセス許可式を入力または貼り付けます。これにより、 AWS CodeDeploy は、ポリシーで指定された Amazon S3 バケットから IAM ユーザーに代わってオンプレミスインスタンスにアプリケーションリビジョンをデプロイできます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:Get*",
                "s3:List*"
```

```
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

6. [ポリシーを作成] を選択します。
7. ナビゲーションペインで [ユーザー] を選択します。
8. ユーザーのリストで、ステップ 1: オンプレミスインスタンス用に IAM ユーザーを作成する で作成した IAM ユーザーを探して選択します。
9. [Permissions] タブを開き、[Managed Policies] で [Attach Policy] を選択します。
10. **CodeDeploy-OnPrem-Permissions** という名前のポリシーを選択した後、[ポリシーのアタッチ] を選択します。

ステップ 3: ユーザーの認証情報を取得する

IAM ユーザーのシークレットキー ID およびシークレットアクセスキーを取得します。これは、ステップ 4: オンプレミスインスタンスに設定ファイルを追加 で必要になります。 AWS CLI または IAM コンソールを使用して、シークレットキー ID とシークレットアクセスキーを取得できます。

> **Note**
>
> すでにシークレットキー ID およびシークレットアクセスキーがある場合、このステップを飛ばして ステップ 4: オンプレミスインスタンスに設定ファイルを追加 へ進んでください。
> ユーザーが の AWS 外部とやり取りする場合は、プログラムによるアクセスが必要です AWS マネジメントコンソール。プログラムによるアクセスを許可する方法は、 がアクセスするユーザーのタイプによって異なります AWS。
> ユーザーにプログラムによるアクセス権を付与するには、以下のいずれかのオプションを選択します。

プログラムによるアクセス権を必要とするユーザー	目的	方法
IAM	(推奨) コンソール認証情報を一時的な認証情報として使用して AWS CLI、、	使用するインターフェイスの指示に従ってください。

プログラムによるアクセス権を必要とするユーザー	目的	方法
	AWS SDKs、または AWS APIs。	• については AWS CLI、AWS Command Line Interface 「 ユーザーガイド」のAWS 「ローカル開発のためのログイン」を参照してください。 • AWS SDKs「 SDK およびツールリファレンスガイド」のAWS 「ローカル開発のためのログイン」を参照してください。 AWS SDKs
ワークフォースアイデンティティ (IAM アイデンティティセンターで管理されているユーザー)	一時的な認証情報を使用して AWS CLI、、 AWS SDKs、または AWS APIs。	使用するインターフェイスの指示に従ってください。 • については AWS CLI、AWS Command Line Interface 「 ユーザーガイド」の「 を使用する AWS CLI ように AWS IAM アイデンティティセンターを設定する」を参照してください。 • AWS SDKs、ツール、API については、 AWS APIs 「 SDK およびツールリファレンスガイド」の「IAM アイデンティティセンター認証」を参照してください。 AWS SDKs

プログラムによるアクセス権を必要とするユーザー	目的	方法
IAM	一時的な認証情報を使用して AWS CLI、、 AWS SDKs、または AWS APIs。	「IAM ユーザーガイド」の「 AWS リソースでの一時的な認証情報の使用」の手順に従います。
IAM	(非推奨) 長期認証情報を使用して、AWS CLI、 AWS SDKs、または AWS APIs。	使用するインターフェイスの指示に従ってください。 • については AWS CLI、「 AWS Command Line Interface ユーザーガイド」の「IAM ユーザー認証情報を使用した認証」を参照してください。 • AWS SDKs「 SDK とツールリファレンスガイド」の「長期認証情報を使用した認証」を参照してください。 AWS SDKs • API AWS APIs「IAM ユーザーのアクセスキーの管理」を参照してください。

認証情報 (CLI) を取得するには

1. list-access-keys コマンドを呼び出し、IAM ユーザーの名前を指定し (`--user-name` のオプションを用いて)、アクセスキー ID のみのクエリを実行します (`--query` および `--output` のオプションを用いて)。例:

```
aws iam list-access-keys --user-name CodeDeployUser-OnPrem --query
 "AccessKeyMetadata[*].AccessKeyId" --output text
```

2. キーが出力に表示されないか、1 つのキーについての情報だけが出力に表示されたら、create-access-key コマンドを呼び出し、IAM ユーザーの名前を指定します (--user-name のオプションを用いて)。

```
aws iam create-access-key --user-name CodeDeployUser-OnPrem
```

create-access-key コマンドを呼び出した出力で、AccessKeyId フィールドおよび SecretAccessKey フィールドの値をメモします。この情報は ステップ 4: オンプレミスインスタンスに設定ファイルを追加 で必要になります。

> **Important**
>
> このシークレットアクセスキーにアクセスできるのは、この時だけです。このシークレットアクセスキーを忘れた、またはアクセスできなくなった場合、ステップ 3: ユーザーの認証情報を取得する のステップに従い、新しいキーを生成する必要があります。

3. 2 つのアクセスキーが既にリストされている場合は、そのうちの 1 つを削除する必要があります。削除するには、delete-access-key コマンドを呼び出して、IAM ユーザーの名前 (--user-name のオプションを用いて) と削除するアクセスキーの ID (--access-key-id のオプションを用いて) を指定します。それから、このステップの前のほうにある説明のとおり、create-access-key コマンドを呼び出します。delete-access-key コマンドを呼び出す例は以下のとおりです。

```
aws iam delete-access-key --user-name CodeDeployUser-OnPrem --access-key-id access-key-ID
```

> **Important**
>
> このアクセスキーの 1 つを削除するために delete-access-key のコマンドを呼び出すとき、および ステップ 4: オンプレミスインスタンスに設定ファイルを追加 で説明するようにオンプレミスインスタンスがすでにこのアクセスキーを使用しているときは、ステップ 4: オンプレミスインスタンスに設定ファイルを追加 の説明に再び従って、この IAM ユーザーに関連付けられる別のアクセスキー ID とシークレットアクセスキーを指定する必要があります。そうしない場合、そのオンプレミスインスタンスへのデプロイは、無期限の保留状態のまま、または完全にエラーとなる可能性があります。

認証情報を取得するには (コンソール)

1. a. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
   b. ユーザーのリストがナビゲーションペインに表示されない場合は、[Users] を選択します。
   c. ユーザーのリストで、ステップ 1: オンプレミスインスタンス用に IAM ユーザーを作成する で作成した IAM ユーザーを探して選択します。
2. [Security credentials] タブにキーが表示されていない場合、または 1 つしか表示されていない場合は、[Create access key] を選択します。

   2 つのアクセスキーが表示されている場合は、片方を削除する必要があります。片方のアクセスキーの横にある [Delete] を選択した後、[Create access key] を選択します。

   > **Important**
   >
   > このいずれかのアクセスキーの横にある Delete を選択する場合、および、オンプレミスインスタンスが ステップ 4: オンプレミスインスタンスに設定ファイルを追加 にある説明のとおりにすでにそのアクセスキーを使用している場合は、ステップ 4: オンプレミスインスタンスに設定ファイルを追加 の手順に再び従って、この IAM ユーザーと関連付けられた異なるアクセスキー ID およびシークレットアクセスキーを指定する必要があります。そうしない場合、そのオンプレミスインスタンスへのデプロイは、無期限の保留状態のまま、または完全にエラーとなる可能性があります。

3. [Show] を選択し、アクセスキー ID とシークレットアクセスキーをメモします。この情報は、次のステップで必要になります。または、[Download .csv file] を選択して、アクセスキー ID とシークレットアクセスキーのコピーを保存することができます。

   > **Important**
   >
   > 認証情報をメモする、または、ダウンロードするのでない限り、このシークレットアクセスキーにアクセスできるのは、この時だけです。このシークレットアクセスキーを忘れた、またはアクセスできなくなった場合、ステップ 3: ユーザーの認証情報を取得する のステップに従い、新しいキーを生成する必要があります。

4. [Close] を選択して [Users > ***IAM User Name***] ページに戻ります。

ステップ 4: オンプレミスインスタンスに設定ファイルを追加

ルートまたは管理者権限を使用して、オンプレミスインスタンスに設定ファイルを追加します。この設定ファイルは、CodeDeploy に使用する IAM ユーザー認証情報とターゲット AWS リージョンを宣言するために使用されます。ファイルは、オンプレミスインスタンスの指定の場所に追加する必要があります。ファイルには、IAM ユーザーの ARN、シークレットキー ID、シークレットアクセスキー、およびターゲット AWS リージョンが含まれている必要があります。ファイルは特定の形式に従っている必要があります。

1. オンプレミスインスタンス上の以下の場所に、`codedeploy.onpremises.yml` (Ubuntu サーバーまたは RHEL オンプレミスインスタンスの場合)、または `conf.onpremises.yml` (Windows サーバーのオンプレミスインスタンスの場合) という名前のファイルを作成します。

   - Ubuntu サーバーの場合:`/etc/codedeploy-agent/conf`
   - Windows サーバーについて : `C:\ProgramData\Amazon\CodeDeploy`

2. テキストエディタを使用して、新しく作成した `codedeploy.onpremises.yml` または `conf.onpremises.yml` ファイルに次の情報を追加します。

   ```
   ---
   aws_access_key_id: secret-key-id
   aws_secret_access_key: secret-access-key
   iam_user_arn: iam-user-arn
   region: supported-region
   ```

   説明は以下のとおりです。

   - *`secret-key-id`* は、ステップ 1: オンプレミスインスタンス用に IAM ユーザーを作成する または ステップ 3: ユーザーの認証情報を取得する でメモした、対応するIAM ユーザーのシークレットキー ID です。
   - *`secret-access-key`* は、ステップ 1: オンプレミスインスタンス用に IAM ユーザーを作成する または ステップ 3: ユーザーの認証情報を取得する でメモした、対応する IAM ユーザーのシークレットアクセスキーです。
   - *`iam-user-arn`* は、「ステップ 1: オンプレミスインスタンス用に IAM ユーザーを作成する」で先にメモした IAM ユーザーの ARN です。
   - *`supported-region`* は、CodeDeploy アプリケーション、デプロイグループ、およびアプリケーションリビジョンが配置されている CodeDeploy でサポートされているリージョンの識別子です (例: `us-west-2`)。リージョンのリストについては、『』の「リージョンとエンドポイント」を参照してくださいAWS 全般のリファレンス。

> **Important**
>
> ステップ 3: ユーザーの認証情報を取得する の片方のアクセスキーの横にある Delete を選択し、またオンプレミスインスタンスがすでに関連するアクセスキー ID およびシークレットアクセスキーを使用している場合は、ステップ 4: オンプレミスインスタンスに設定ファイルを追加 の手順に再び従って、この IAM ユーザーと関連付けられた異なるアクセスキー ID およびシークレットアクセスキーを指定する必要があります。そうしない場合、オンプレミスインスタンスへのデプロイは、無期限の保留状態のまま、または完全にエラーとなる可能性があります。

ステップ 5: をインストールして設定する AWS CLI

オンプレミスインスタンス AWS CLI に をインストールして設定します。(AWS CLI は で使用されステップ 7: CodeDeploy エージェントのインストール、オンプレミスインスタンスに CodeDeploy エージェントをダウンロードしてインストールします。)

1. オンプレミスインスタンス AWS CLI に をインストールするには、AWS Command Line Interface 「 ユーザーガイド」の「 のセットアップ AWS CLI」の手順に従います。

> **Note**
>
> オンプレミスインスタンスを使用するための CodeDeploy コマンドは、 AWS CLIのバージョン 1.7.19 で使用できるようになりました。のバージョンが AWS CLI 既にインストールされている場合は、 を呼び出してそのバージョンを確認できますaws --version。

2. オンプレミスインスタンス AWS CLI で を設定するには、AWS Command Line Interface 「 ユーザーガイド」の「 の設定 AWS CLI」の手順に従います。

> **Important**
>
> (aws configure コマンドを呼び出す AWS CLI などして) を設定するときは、 で指定されたアクセス許可に加えて、少なくとも以下の AWS アクセス許可を持つ IAM ユーザーのシークレットキー ID とシークレットアクセスキーを必ず指定してくださいオンプレミスインスタンスを設定するための前提条件。これにより、オンプレミスインスタンス上で CodeDeploy エージェントをダウンロードしてインストールすることができます。

JSON

```
{
  "Version":"2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "codedeploy:*"
      ],
      "Resource" : "*"
    },
    {
      "Effect" : "Allow",
      "Action" : [
        "s3:Get*",
        "s3:List*"
      ],
      "Resource" : [
        "arn:aws:s3:::amzn-s3-demo-bucket/*",
        "arn:aws:s3:::amzn-s3-demo-bucket1/*"
      ]
    }
  ]
}
```

JSON

```
{
  "Version":"2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "codedeploy:*"
      ],
      "Resource" : "*"
    },
    {
```

```
      "Effect" : "Allow",
      "Action" : [
        "s3:Get*",
        "s3:List*"
      ],
      "Resource" : [
        "*"
      ]
    }
  ]
}
```

これらのアクセス権限は、ステップ 1: オンプレミスインスタンス用に IAM ユーザーを作成する で作成した IAM ユーザー、または別のユーザーのいずれかに割り当てられることができます。これらのアクセス権限を ユーザーに割り当てるには、ステップ 1: オンプレミスインスタンス用に IAM ユーザーを作成する にある手順に従いますが、手順のアクセス権限の代わりにこれらのアクセス権限を使用します。

ステップ 6: AWS_REGION 環境変数を設定する (Ubuntu Server および RHEL のみ)

オンプレミスインスタンス上で Ubuntu サーバーまたは RHEL を実行中でなければ、このステップをスキップして直接 ステップ 7: CodeDeploy エージェントのインストール へ進んでください。

Ubuntu サーバーまたは RHEL オンプレミスインスタンス上に CodeDeploy エージェントをインストールし、新しいバージョンが使用可能になったらいつでも CodeDeploy エージェントを更新するようにインスタンスを有効にします。これを行うには、インスタンスのAWS_REGION環境変数を CodeDeploy でサポートされているリージョンの 1 つの識別子に設定します。値を CodeDeploy アプリケーション、デプロイグループ、およびアプリケーションリビジョンがあるリージョン (など) に設定することをお勧めしますus-west-2。リージョンのリストについては、『』の「リージョンとエンドポイント」を参照してくださいAWS 全般のリファレンス。

環境変数を設定するには、端末から以下を呼び出します。

```
export AWS_REGION=supported-region
```

supported-region はリージョン識別子です (例: us-west-2)。

ステップ 7: CodeDeploy エージェントのインストール

オンプレミスインスタンスに CodeDeploy エージェントをインストールします。

- Ubuntu サーバーオンプレミスインスタンスの場合は、Ubuntu サーバー用の CodeDeploy エージェントをインストールする の手順に従った後、このページに戻ります。
- RHEL オンプレミスインスタンスについては、Amazon Linux または RHEL 用の CodeDeploy エージェントをインストールする の手順に従った後、このページに戻ります。
- Windows Server オンプレミスインスタンスの場合は、Windows サーバー用の CodeDeploy エージェントです。 の手順に従った後、このページに戻ります。

ステップ 8: CodeDeploy でオンプレミスインスタンスを登録します。

このステップの手順では、オンプレミスインスタンス自体からオンプレミスインスタンスを登録していることを想定します。「」で説明されているように、 AWS CLI がインストールされ、設定された別のデバイスまたはインスタンスからオンプレミスインスタンスを登録できますステップ 5: をインストールして設定する AWS CLI。

AWS CLI を使用してオンプレミスインスタンスを CodeDeploy に登録し、デプロイで使用できるようにします。

1. を使用する前に AWS CLI、 で作成した IAM ユーザーのユーザー ARN が必要ですステップ 1: オンプレミスインスタンス用に IAM ユーザーを作成する。ユーザー ARN がまだない場合は、get-user コマンドを呼び出し、IAM ユーザーの名前を指定し (`--user-name` のオプションを用いて)、ユーザー ARN のみをクエリします (`--query` のオプションと `--output` のオプションを用いて)。

   ```
   aws iam get-user --user-name CodeDeployUser-OnPrem --query "User.Arn" --output text
   ```

2. register-on-premises-instance コマンドを呼び出し、以下を指定します。

   - オンプレミスインスタンスを一意に識別する名前 (`--instance-name` オプションで指定)。

     > **Important**
     >
     > オンプレミスインスタンスを識別するために、特にデバッグのため、オンプレミスインスタンスの一意な特徴を示す名前 (例えば、もしあれば、シリアルナンバーや内部アセット識別子など) を指定することを強くお勧めします。名前として MAC アドレスを指定した場合、MAC アドレスには、コロン (:) など CodeDeploy が許可し

ない文字が含まれることに注意してください。許可された文字の一覧については、「CodeDeploy のクォータ」を参照してください。

- ステップ 1: オンプレミスインスタンス用に IAM ユーザーを作成する で作成した IAM ユーザーの ARN (`--iam-user-arn` のオプションを用いて)

  例:

```
aws deploy register-on-premises-instance --instance-name AssetTag12010298EX --
iam-user-arn arn:aws:iam::444455556666:user/CodeDeployUser-OnPrem
```

ステップ 9: オンプレミスインスタンスにタグ付けする

AWS CLI または CodeDeploy コンソールを使用して、オンプレミスインスタンスにタグを付けることができます。(CodeDeploy はオンプレミスインスタンスタグを使用してデプロイ中にデプロイターゲットを識別します。)

オンプレミスインスタンスにタグ付けするには (CLI)

- add-tags-to-on-premises-instances コマンドを呼び出し、以下を指定します。

  - オンプレミスインスタンスを一意に識別する名前 (`--instance-names` オプションで指定)。
  - 使用するオンプレミスインスタンスのタグキーの名前とタグ値 (`--tags` オプションで指定)。名前と値はいずれも指定する必要があります。CodeDeploy は値のみがあるオンプレミスインスタンスタグを許可しません。

  例:

```
aws deploy add-tags-to-on-premises-instances --instance-names AssetTag12010298EX
 --tags Key=Name,Value=CodeDeployDemo-OnPrem
```

オンプレミスインスタンスにタグ付けするには (コンソール)

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

> **Note**
>
> 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. CodeDeploy メニューから On-premises instances を選択します。
3. オンプレミスインスタンスのリストで、タグ付けするオンプレミスインスタンスの横にある矢印を選択します。
4. タグのリストで、目的のタグキーとタグ値を選択または入力します。タグキーとタグ値を入力するたびに、別の行が表示されます。最大 10 個のタグにこれを繰り返すことができます。タグを削除するには、削除アイコンを選択します ()。
5. タグを追加したら、[Update Tags] を選択します。

ステップ 10: アプリケーションリビジョンをオンプレミスインスタンスにデプロイする

登録され、タグ付けされたオンプレミスインスタンスにアプリケーションリビジョンをデプロイする準備ができました。

Amazon EC2 インスタンスにアプリケーションリビジョンをデプロイするのと同様の方法でオンプレミスインスタンスにアプリケーションリビジョンをデプロイします。手順については、「CodeDeploy でデプロイを作成する」を参照してください。これらの指示には、アプリケーションの作成、開発グループの作成、およびアプリケーションリビジョンの準備を含む前提条件へのリンクが含まれています。シンプルなサンプルアプリケーションリビジョンをデプロイすることが必要な場合は、チュートリアル: CodeDeploy (Windows サーバー、Ubuntu サーバー、または Red Hat エンタープライズ Linux) を使用してオンプレミスインスタンスにアプリケーションをデプロイします。の ステップ 2: サンプルのアプリケーションリビジョンを作成する で説明してあるものを作成できます。

> **Important**
>
> オンプレミスインスタンスを対象としたデプロイグループの作成の一部として、CodeDeploy サービスロールを再利用する場合は、`Tag:get*` をサービスロールのポリシーステートメントの `Action` の部分に含める必要があります。詳細については、「ステップ 2: CodeDeployのサービスのロールを作成する」を参照してください。

ステップ 11: オンプレミスインスタンスへのデプロイを追跡する

登録されタグ付けされたオンプレミスインスタンスへアプリケーションリビジョンをデプロイした後、デプロイの進行状況を追跡できます。

Amazon EC2 インスタンスへのデプロイの追跡と同様の方法でオンプレミスインスタンスへのデプロイの追跡をします。手順については、「CodeDeploy デプロイの詳細を表示する」を参照してください。

CodeDeploy でのオンプレミスインスタンスのオペレーションの管理

このセクションの手順に従って、オンプレミスインスタンスに関する詳細情報の取得、タグの削除、オンプレミスインスタンスのアンインストールと登録解除など、オンプレミスインスタンスをCodeDeployに登録した後の操作を管理します。

トピック

- 単一のオンプレミスインスタンスに関する情報を取得します
- 複数のオンプレミスインスタンスに関する情報を取得します
- オンプレミスインスタンスから手動でオンプレミスインスタンスタグを削除します
- CodeDeploy エージェントを自動でアンインストールし、設定ファイルをオンプレミスインスタンスから削除します。
- オンプレミスインスタンスを自動的に登録解除します
- 手動でオンプレミスのインスタンスの登録を解除します

単一のオンプレミスインスタンスに関する情報を取得します

CodeDeploy デプロイの詳細を表示する にある手順に従って、単一のオンプレミスインスタンスに関する情報を取得できます。 AWS CLI または CodeDeploy コンソールを使用して、単一のオンプレミスインスタンスに関する詳細情報を取得できます。

単一のオンプレミスインスタンスについての情報を取得するには (CLI)

- get-on-premises-instance コマンドを呼び出し、オンプレミスインスタンスを一意に識別する名前を指定します (`--instance-name` のオプションを用いて)。

```
aws deploy get-on-premises-instance --instance-name AssetTag12010298EX
```

単一のオンプレミスインスタンスに関する情報を取得するには (コンソール)

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで、Deploy を展開し、On-premises instances を選択します。
3. オンプレミスインスタンスのリストで、詳細を表示するオンプレミスインスタンスの名前を選択します。

複数のオンプレミスインスタンスに関する情報を取得します

CodeDeploy デプロイの詳細を表示する にある手順に従って、オンプレミスインスタンスに関する情報を取得できます。 AWS CLI または CodeDeploy コンソールを使用して、オンプレミスインスタンスに関する詳細情報を取得できます。

複数のオンプレミスインスタンスに関する情報を取得するには (CLI)

1. オンプレミスインスタンスの名前のリストでは、list-on-premises-instances コマンドを呼び出し、以下を指定します。

   - すべての登録または登録解除されたオンプレミスインスタンスに関する情報を取得するかどうか (`--registration-status` オプションの `Registered` または `Deregistered` でそれぞれ指定)。これを省略すると、登録されている、および登録解除されたオンプレミスインスタンスの名前が両方とも返されます。
   - 特定のオンプレミスインスタンスのタグが付けられたオンプレミスインスタンスに関する情報だけを取得するかどうか (`--tag-filters` オプションで指定)。各オンプレミスインスタンスタグで、`Key`、`Value`、および `Type` を指定します (常に `KEY_AND_VALUE` である必要があります)。複数のオンプレミスインスタンスタグは、`Key`、`Value`、`Type` の間にスペースを入れて区切ります。

   例:

```
aws deploy list-on-premises-instances --registration-status Registered
 --tag-filters Key=Name,Value=CodeDeployDemo-OnPrem,Type=KEY_AND_VALUE
 Key=Name,Value=CodeDeployDemo-OnPrem-Beta,Type=KEY_AND_VALUE
```

2. 詳細については、オンプレミスインスタンスの名前で (`--instance-names` のオプションを用いて)、batch-get-on-premises-instances コマンドを呼び出します 。

```
aws deploy batch-get-on-premises-instances --instance-names AssetTag12010298EX
 AssetTag09920444EX
```

複数のオンプレミスインスタンスに関する情報を取得するには (コンソール)

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

> **Note**
>
> 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで、Deploy を展開し、デプロイを選択し、On-premises instances を選択します。

   オンプレミスインスタンスに関する情報が表示されます。

オンプレミスインスタンスから手動でオンプレミスインスタンスタグを削除します

通常、タグが使われなくなった場合や、タグに依存するデプロイグループからオンプレミスインスタンスを削除する場合は、オンプレミスインスタンスからオンプレミスインスタンスタグを削除します。 AWS CLI または AWS CodeDeploy コンソールを使用して、オンプレミスインスタンスからオンプレミスインスタンスタグを削除できます。

登録を解除する前にオンプレミスインスタンスからオンプレミスインスタンスタグを削除する必要はありません。

オンプレミスインスタンスからオンプレミスインスタンスタグを手動で削除しても、インスタンスの登録は解除されません。CodeDeploy エージェントはインスタンスからアンインストールされません。インスタンスから設定ファイルは削除されません。インスタンスに関連付けられた IAM ユーザーは削除されません。

自動的にオンプレミスインスタンスの登録を解除するには、オンプレミスインスタンスを自動的に登録解除します を参照してください。

オンプレミスインスタンスの登録を手動で解除するには、手動でオンプレミスのインスタンスの登録を解除します を参照してください。

CodeDeploy エージェントを自動でアンインストールし、設定ファイルをオンプレミスインスタンスから削除するには、CodeDeploy エージェントを自動でアンインストールし、設定ファイルをオンプレミスインスタンスから削除します。 を参照してください。

CodeDeploy エージェントだけをオンプレミスインスタンスから手動でアンインストールするには、CodeDeploy エージェントのオペレーションの管理 を参照してください。

関連する IAM ユーザーを手動で削除するには、Deleting an IAM user from your AWS account を参照してください。

オンプレミスインスタンスから手動でオンプレミスインスタンスタグを削除するには (CLI)

- remove-tags-from-on-premises-instances を呼び出し、以下を指定します。

  - オンプレミスインスタンスを一意に識別する名前 (`--instance-names` オプションで指定)。
  - 削除するタグの名前と値 (`--tags` オプションで指定)。

    例:

```
aws deploy remove-tags-from-on-premises-instances --instance-names
 AssetTag12010298EX --tags Key=Name,Value=CodeDeployDemo-OnPrem
```

オンプレミスインスタンスから手動でオンプレミスインスタンスタグを削除するには (コンソール)

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで、Deploy を展開し、On-premises instances を選択します。

3. オンプレミスインスタンスのリストで、タグを削除するオンプレミスインスタンスの名前を選択します。
4. [タグ] で、削除する各タグの横にある [削除] を選択します。
5. タグを削除したら、[Update tags] を選択します。

CodeDeploy エージェントを自動でアンインストールし、設定ファイルをオンプレミスインスタンスから削除します。

通常、今後デプロイする予定がなければ、CodeDeploy エージェントをアンインストールし、オンプレミスインスタンスから設定ファイルを削除します。

> **Note**
>
> CodeDeploy エージェントを自動でアンインストールし、設定ファイルをオンプレミスインスタンスから削除しても、オンプレミスインスタンスの登録は解除されません。オンプレミスインスタンスに関連付けられたオンプレミスインスタンスタグの関連付けは解除されません。オンプレミスインスタンスに関連付けられた IAM ユーザーは削除されません。
> 自動的にオンプレミスインスタンスの登録を解除するには、オンプレミスインスタンスを自動的に登録解除します を参照してください。
> オンプレミスインスタンスの登録を手動で解除するには、手動でオンプレミスのインスタンスの登録を解除します を参照してください。
> オンプレミスインスタンスタグの関連付けを手動で解除するには、オンプレミスインスタンスから手動でオンプレミスインスタンスタグを削除します を参照してください。
> オンプレミスインスタンスから CodeDeploy エージェントを手動でアンインストールするには、CodeDeploy エージェントのオペレーションの管理 を参照してください。
> 関連する IAM ユーザーを手動で削除するには、Deleting an IAM user from your AWS account を参照してください。

オンプレミスインスタンスから、 AWS CLI を使用してアンインストールコマンドを呼び出します。

例えば、次のようになります。

```
aws deploy uninstall
```

uninstall コマンドは次のことを行います。

1. オンプレミスインスタンスで実行の CodeDeploy エージェントを停止します。

2. オンプレミスインスタンスから CodeDeploy エージェントをアンインストールします。
3. オンプレミスインスタンスから設定ファイルを削除します。(Ubuntu サーバーとRHELの場合、これは`/etc/codedeploy-agent/conf/codedeploy.onpremises.yml` です。Windows サーバーの場合、これは`C:\ProgramData\Amazon\CodeDeploy\conf.onpremises.yml` です。)

オンプレミスインスタンスを自動的に登録解除します

通常は、デプロイする計画がなくなった後、オンプレミスインスタンスの登録を解除します。オンプレミスインスタンスの登録を解除すると、オンプレミスインスタンスがデプロイグループのオンプレミスインスタンスタグの一部であっても、オンプレミスインスタンスはデプロイには含まれなくなります。を使用して、オンプレミスインスタンスの登録 AWS CLI を解除できます。

> **Note**
>
> CodeDeploy コンソールを使用してオンプレミスインスタンスの登録を解除することはできません。また、オンプレミスインスタンスの登録を解除しても、オンプレミスインスタンスに関連付けられたオンプレミスインスタンスタグは解除されません。オンプレミスインスタンスから CodeDeploy エージェントをアンインストールしません。オンプレミスインスタンスの設定ファイルはオンプレミスインスタンスから削除されません。
> このセクションのアクティビティの一部 (ただしすべてではない) を実行するために CodeDeploy コンソールを使うには、手動でオンプレミスのインスタンスの登録を解除します の CodeDeploy コンソールセクションを参照してください。
> オンプレミスインスタンスタグの関連付けを手動で解除するには、オンプレミスインスタンスから手動でオンプレミスインスタンスタグを削除します を参照してください。
> CodeDeploy エージェントを自動でアンインストールし、設定ファイルをオンプレミスインスタンスから削除するには、CodeDeploy エージェントを自動でアンインストールし、設定ファイルをオンプレミスインスタンスから削除します。 を参照してください。
> CodeDeploy エージェントのみをオンプレミスインスタンスから手動でアンインストールするには、CodeDeploy エージェントのオペレーションの管理 を参照してください。

を使用して AWS CLI 登録解除コマンドを呼び出し、以下を指定します。

- CodeDeploy に対してオンプレミスインスタンスを一意に識別する名前 (`--instance-name` のオプションを用いて)
- 必要に応じて、オンプレミスインスタンスに関連付けられた IIAM ユーザーを削除するかどうか デフォルトの動作では、IAM ユーザーを削除します。オンプレミスインスタンスに関連付けられた

IAM ユーザーを削除しない場合は、コマンドで `--no-delete-iam-user` オプションを指定します。

- オプションで、オンプレミスインスタンスが CodeDeploy に登録された AWS リージョン (`--region`オプションを使用) 。これは、 のリージョンとエンドポイントにリストされているサポートされているリージョンのいずれかである必要があります AWS 全般のリファレンス (例: `us-west-2`)。このオプションを指定しない場合、呼び出し元の IAM ユーザーに関連付けられたデフォルトの AWS リージョンが使用されます。

インスタンスを登録解除し、ユーザーを削除する例:

```
aws deploy deregister --instance-name AssetTag12010298EX --region us-west-2
```

インスタンスを登録解除し、ユーザーを削除しない例:

```
aws deploy deregister --instance-name AssetTag12010298EX --no-delete-iam-user --region
 us-west-2
```

deregister コマンドは次のことを行います。

1. CodeDeploy でオンプレミスインスタンスの登録を解除します。
2. 指定した場合、オンプレミスインスタンスに関連付けられている IAM ユーザーを削除します。

オンプレミスインスタンスの登録解除後:

- コンソールにはすぐに表示されなくなります。
- 直後に同じ名前で別のインスタンスを作成できます。

このコマンドでエラーが発生した場合、エラーメッセージが表示され、手動で残りのステップを完了する方法について説明します。それ以外の場合は、成功メッセージが表示され、uninstall コマンドを呼び出す方法が説明されます。

手動でオンプレミスのインスタンスの登録を解除します

通常は、デプロイする計画がなくなった後、オンプレミスインスタンスの登録を解除します。を使用して AWS CLI 、オンプレミスインスタンスを手動で登録解除します。

手動でオンプレミスインスタンスの登録を解除しても、CodeDeploy エージェントはアンインストールされません。インスタンスから設定ファイルは削除されません。インスタンスに関連付けられた IAM ユーザーは削除されません。インスタンスに関連付けられるタグは削除されません。

CodeDeploy エージェントを自動でアンインストールし、設定ファイルをオンプレミスインスタンスから削除するには、CodeDeploy エージェントを自動でアンインストールし、設定ファイルをオンプレミスインスタンスから削除します。 を参照してください。

手動で CodeDeploy エージェントのみをアンインストールするには、CodeDeploy エージェントのオペレーションの管理 を参照してください。

関連する IAM ユーザーを手動で削除するには、Deleting an IAM user from your AWS account を参照してください。

関連付けられたオンプレミスインスタンスタグのみを手動で削除するには、オンプレミスインスタンスから手動でオンプレミスインスタンスタグを削除します を参照してください。

- deregister-on-premises-instance コマンドを呼び出し、オンプレミスインスタンスを一意に識別する名前を指定します (`--instance-name` のオプションを用いて)。

  ```
  aws deploy deregister-on-premises-instance --instance-name AssetTag12010298EX
  ```

  オンプレミスインスタンスの登録解除後:

  - コンソールにはすぐに表示されなくなります。
  - 直後に同じ名前で別のインスタンスを作成できます。

CodeDeploy を用いてインスタンスの詳細の表示

CodeDeploy コンソール、 AWS CLI、または CodeDeploy APIs を使用して、デプロイで使用されるインスタンスの詳細を表示できます。

インスタンスを見るための CodeDeploy API アクションについての情報については、GetDeploymentInstance、ListDeploymentInstances、および ListOnPremisesInstances を参照してください。

トピック

- インスタンスの詳細の表示 (コンソール)

- インスタンスの詳細の表示 (CLI)

インスタンスの詳細の表示 (コンソール)

インスタンスの詳細を表示するには:

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

> **Note**
>
> 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで [デプロイ] を展開し、[アプリケーション] を選択します。

> **Note**
>
> エントリが表示されない場合は、正しいリージョンが選択されていることを確認しましょう。ナビゲーションバーのリージョンセレクターで、「AWS 全般のリファレンス」の「リージョンとエンドポイント」に一覧表示されているいずれかのリージョンを選択します。CodeDeploy は、これらのリージョンでのみサポートされています。

3. デプロイの詳細を表示するには、そのインスタンスのデプロイ ID を選択します。
4. デプロイのページの [インスタンスアクティビティ] セクションですべてのインスタンスを表示できます。
5. インスタンスの個別のデプロイライフサイクルイベントに関する詳細を表示するには、デプロイの詳細ページの [Events] 列で、[View events] を選択します。

> **Note**
>
> いずれかのライフサイクルイベントに対して [Failed] が表示された場合、インスタンスの詳細ページで、[View logs]、[View in EC2]、またはその両方を選択します。トラブルシューティングのヒントについては、「インスタンスの問題のトラブルシューティング」を参照してください。

6. Amazon EC2 インスタンスに関するさらなる詳細情報を確認したい場合は、Instance ID 列でインスタンスの ID を選択します。

インスタンスの詳細の表示 (CLI)

を使用してインスタンスの詳細 AWS CLI を表示するには、 `get-deployment-instance` コマンドまたは `list-deployment-instances` コマンドを呼び出します。

1 つのインスタンスに関する詳細を表示するには、get-deployment-instance コマンドを呼び出し、以下を指定します。

- 一意のデプロイ ID。デプロイ ID を取得するには、list-deployments コマンドを呼び出します。
- 一意のインスタンス ID。インスタンス ID を取得するには、list-deployment-instances コマンドを呼び出します。

デプロイ中で使用されるインスタンスの ID のリストを表示するには、list-deployment-instances コマンドを呼び出し、以下を指定します：

- 一意のデプロイ ID。デプロイ ID を取得するには、list-deployments コマンドを呼び出します。
- オプションで、デプロイの状態別に特定のインスタンス ID のみを含めるかどうか。(指定しない場合、一致するすべてのインスタンス ID が、デプロイの状態にかかわらず表示されます。)

CodeDeploy インスタンスのヘルス

CodeDeploy は、デプロイグループのインスタンスのヘルスステータスをモニタリングします。正常なインスタンスの数が、デプロイ中にデプロイグループに指定された正常なインスタンスの最小数を下回ると、デプロイが失敗します。例えば、インスタンスの 85% がデプロイ中に正常な状態を維持する必要があり、デプロイグループに 10 個のインスタンスが含まれている場合、1 つのインスタンスへのデプロイが失敗したときであっても、デプロイ全体が失敗します。これは、最新のアプリケーションリビジョンをインストールできるようにインスタンスをオフラインにすると、使用可能な正常なインスタンス数が既に 90% に減少するためです。失敗したインスタンスと別のオフラインインスタンスを合わせると、正常に使用できるインスタンスの 80% が正常であり、利用できることを意味します。CodeDeploy は全体的なデプロイに失敗します。

デプロイ全体が成功するには、以下が満たされている必要があります。

- CodeDeploy は、デプロイ中の各インスタンスにデプロイできます。
- 少なくとも 1 つのインスタンスへのデプロイに成功する。つまり、デプロイ全体が成功するには、最小限の正常なホストの値が 0 の場合であっても、少なくとも 1 つ以上のインスタンスへのデプロイに成功する必要があります (1 つ以上のインスタンスが正常)。

トピック

- ヘルスステータス
- 正常なインスタンスの最小数について
- アベイラビリティーゾーンあたりの正常なインスタンスの最小数について

ヘルスステータス

CodeDeploy は、各インスタンスに 2 つのヘルスステータス値 (revision health と instance health) を割り当てます。

リビジョンの状態

リビジョンの状態は、現在インスタンスにインストールされているアプリケーションリビジョンに基づきます。以下のステータス値があります。

- Current: インスタンスにインストールされているリビジョンは、デプロイグループの前回成功したデプロイのリビジョンに一致します。
- Old: インスタンスにインストールされているリビジョンは、アプリケーションの以前のバージョンと一致します。
- Unknown: アプリケーションリビジョンはインスタンスに正常にインストールされていません。

インスタンスの状態

インスタンスの状態は、インスタンスへのデプロイが成功したかどうかに基づきます。以下の値があります。

- Healthy: インスタンスへの前回のデプロイは成功しました。
- Unhealthy: インスタンスへのリビジョンのデプロイの試みは失敗したか、リビジョンがインスタンスにデプロイされていません。

CodeDeploy は、リビジョンのヘルスとインスタンスのヘルスを使用して、次の順序でデプロイグループのインスタンスにデプロイをスケジュールします。

1. インスタンスの状態が Unhealthy。
2. リビジョンの状態が Unknown。
3. リビジョンの状態が Old。
4. リビジョンの状態が Current。

デプロイ全体が成功すると、リビジョンは更新され、デプロイグループのヘルスステータスの値が更新されて、最新のデプロイを反映します。

- デプロイに成功したすべての最新のインスタンスは current のままになります。それ以外の場合は、unknown になります。
- デプロイに成功したすべての古いインスタンスまたは不明なインスタンスは current になります。それ以外の場合は、old または unknown のままになります。
- デプロイに成功したすべての正常なインスタンスは healthy のままになります。それ以外の場合は、unhealthy になります。
- デプロイに成功したすべての異常なインスタンスは healthy になります。それ以外の場合は、unhealthy のままになります。

全体的なデプロイが失敗するか、停止している場合:

- CodeDeploy がアプリケーションリビジョンのデプロイを試みた各インスタンスでは、そのインスタンスのデプロイの試みが成功したか失敗したかによって、そのインスタンスの状態が healthy または unhealthy に設定されています。
- CodeDeploy がアプリケーションリビジョンのデプロイを試みなかった各インスタンスは、現在のインスタンスのヘルスの値を保持します。
- デプロイグループのリビジョンに変更はありません。

正常なインスタンスの最小数について

正常なインスタンスに必要な最小数は、デプロイ設定の一部として定義されます。

> **Important**
>
> Blue/Green デプロイ中に、デプロイ設定と最小限の正常なホストの値は、元の環境ではなく置き換え先環境のインスタンスに適用されます。ただし、元の環境のインスタンスがロードバランサーから登録解除されている場合、1 つの元のインスタンスが正常な登録解除に失敗したときであっても、全体的なデプロイは失敗とマークされます。

CodeDeploy は、最小限の正常なホストの値を一般的に使用した 3 つのデフォルトのデプロイ構成を提供します。

デフォルトのデプロイ設定名	事前定義された最小限の正常なホストの値
CodeDeployDefault.OneAtATime	1
CodeDeployDefault.HalfAtATime	50%
CodeDeployDefault.AllAtOnce	0

デフォルトのデプロイ設定の詳細は、「CodeDeploy でデプロイ設定を使用する」で参照できます。

CodeDeploy 中でカスタムデプロイ設定を作成して、独自の最小限の正常なホストの値を定義できます。これらの値は、次のオペレーションを使用して整数または割合 (%) として定義できます。

- `minimum-healthy-hosts` で create-deployment-config コマンドを使う時の AWS CLIとして。
- CodeDeploy API の MinimumHealthyHosts データタイプでの `Value` として 。
- CloudFormation テンプレートで AWS::CodeDeploy::DeploymentConfig `MinimumHealthyHosts`を使用する場合と同じです。

CodeDeploy では、2 つの主な目的のために、デプロイに対して正常なインスタンスの最小数を指定できます。

- 全体的なデプロイの成功または失敗を判断する。アプリケーションリビジョンが少なくとも最小数の正常なインスタンスに正しくデプロイされた場合、デプロイは成功します。
- デプロイが続行するためにデプロイ中に正常である必要があるインスタンス数を決定する。

デプロイグループの正常なインスタンスの最小数は、インスタンス数、または合計インスタンス数の割合 (%) として指定できます。パーセンテージを指定する場合、CodeDeploy はデプロイの開始時にパーセンテージを同等のインスタンス数に変換し、端数がある場合は切り上げます。

CodeDeploy は、デプロイプロセス中にデプロイグループのインスタンスのヘルスステータスを追跡し、デプロイの指定された正常なインスタンスの最小数を使用して、デプロイを続行するかどうか判断します。基本的な原則は、デプロイによって、正常なインスタンスの数が、指定した最小数を下回ってはならないということです。このルールの 1 つの例外は、デプロイグループの正常なインスタンスの数が、指定した最小数より最初から少ない場合です。その場合、デプロイプロセスによってそれ以上正常なインスタンスの数が減ることはありません。

> **Note**
>
> CodeDeploy は現在停止状態にあるものも含めて、デプロイグループ中のすべてのインスタンスへのデプロイを試みます。最小限の正常なホストの計算では、停止中のインスタンスは失敗したインスタンスと同じ影響を与えます。停止中のインスタンスが多すぎるために失敗したデプロイを解決するには、インスタンスを再起動するか、タグを変更してデプロイグループから除外します。

CodeDeploy は、アプリケーションリビジョンをデプロイグループの異常なインスタンスにデプロイするよう試みて、デプロイプロセスを開始します。デプロイに成功するたびに、CodeDeploy はインスタンスのヘルスステータスを「正常」に変更し、それをデプロイグループの正常なインスタンスに追加します。それで、CodeDeploy は正常なインスタンスの現在の数を、正常なインスタンスの指定された最小数と比較します。

- 正常なインスタンスの数が、正常なインスタンスの指定された最小数以下である場合、CodeDeploy はデプロイをキャンセルし、より多くのデプロイによって正常なインスタンスの数が減らないようにします。
- 正常なインスタンスの数が指定された正常なインスタンスの最小数よりも少なくとも 1 つ大きい場合、CodeDeploy はアプリケーションのリビジョンを正常なインスタンスの元のセットにデプロイします。

正常なインスタンスへのデプロイが失敗した場合、CodeDeploy はそのインスタンスのヘルスステータスを unhealthy に変更します。デプロイが進行するにつれて、CodeDeploy は正常なインスタンスの現在の数を更新し、正常なインスタンスの指定された最小数と比較します。デプロイプロセスのある時点で、正常なインスタンスの数が指定された最小数にまで減った場合、CodeDeploy はデプロイを停止します。この方法により、次のデプロイが失敗し、正常なインスタンスの数が指定された最小数を下回ることを回避できます。

> **Note**
>
> 指定する正常なインスタンスの最小数が、デプロイグループの合計インスタンス数より少ないことを確認します。割合 (%) の値を指定する場合、切り上げられることを覚えておいてください。それ以外の場合、デプロイが開始すると、正常なインスタンスの数は正常なインスタンスの指定された最小数以下となり、CodeDeploy によりデプロイ全体が直ちに失敗します。

また、CodeDeploy は指定された正常なインスタンスの最小数および正常なインスタンスの実際の数を使用して、複数のインスタンスにアプリケーションリビジョンをデプロイするかどうかと、その方法を決定します。デフォルトでは、CodeDeploy は正常なインスタンスの数が、指定された正常なインスタンスの最小数を下回るリスクなく、可能な限り多くのインスタンスにアプリケーションリビジョンをデプロイします。

一度にデプロイする必要があるデプロイ先のインスタンス数を決定する場合、CodeDeploy は次の計算を使用します。

```
[total-hosts] - [minimum-healthy-hosts] =
        [number-of-hosts-to-deploy-to-at-once]
```

例えば、次のようになります。

- デプロイグループに 10 個のインスタンスがあり、正常なインスタンスの最小数を 9 に設定した場合、CodeDeploy は一度に 1 つのインスタンスにデプロイします。
- デプロイグループに 10 個のインスタンスがあり、正常なインスタンスの最小数を 3 に設定した場合、CodeDeploy は最初のバッチで同時に 7 つのインスタンスにデプロイし、2 番目のバッチで残りの 3 つのインスタンスにデプロイします。
- デプロイグループに 10 個のインスタンスがあり、正常なインスタンスの最小数を 0 に設定した場合、CodeDeploy は同時に 10 個のインスタンスにデプロイします。

例

次の例では、10 個のインスタンスがあるデプロイグループを前提としています。

正常なインスタンスの最小数: 95%

CodeDeploy は正常なインスタンスの最小数を 10 個のインスタンスに切り上げます。これは正常なインスタンスの数と等しくなります。全体的なデプロイは、いずれのインスタンスにもリビジョンをデプロイすることなく、直ちに失敗します。

正常なインスタンスの最小数: 9

CodeDeploy は一度に 1 個のインスタンスへリビジョンをデプロイします。いずれかのインスタンスへのデプロイが失敗した場合、正常なインスタンスの数は正常なインスタンスの最小数に等しくなるため、CodeDeploy のデプロイ全体は直ちに失敗します。このルールの例外は、最後のインスタンスが失敗した場合でも、デプロイは引き続き成功することです。

CodeDeploy は、一度に 1 個のインスタンスずつ、デプロイが失敗するか、デプロイ全体が完了するまで、デプロイを続行します。10 のデプロイすべてに成功した場合、デプロイグループには 10 個の正常なインスタンスが含まれます。

正常なインスタンスの最小数: 8

CodeDeploy は一度に 2 個のインスタンスへリビジョンをデプロイします。これらのうち 2 つのデプロイに失敗した場合、CodeDeploy は直ちにデプロイ全体を失敗します。このルールの例外は、最後のインスタンスが 2 番目に失敗した場合でも、デプロイは成功することです。

正常なインスタンスの最小数: 0

CodeDeploy は、一度にデプロイグループ全体へリビジョンをデプロイします。デプロイ全体が成功するには、インスタンスに対して、少なくとも 1 つ以上のデプロイが成功する必要があります。正常なインスタンスが 0 の場合、デプロイは失敗します。これは、デプロイ全体を成功としてマークするには、正常なインスタンスの最小値が 0 であっても、デプロイ全体の完了時に 1 つ以上のインスタンスが正常であることが要件であるためです。

アベイラビリティーゾーンあたりの正常なインスタンスの最小数について

> **Note**
>
> このセクションでは、Amazon EC2 インスタンスを表す用語としてインスタンスとホストを同じ意味で使用しています。

複数のアベイラビリティーゾーンのインスタンスにデプロイする場合は、オプションでzonal configuration機能を有効にすると、CodeDeploy は一度に 1 つのアベイラビリティーゾーンにデプロイできます。

この機能を有効にすると、CodeDeploy は正常なホストの数が「ゾーンあたりの正常なホストの最小数」および「正常なホストの最小数」の値を常に超えていることを確認します。正常なホストの数がいずれかの値を下回ると、CodeDeploy はすべてのアベイラビリティーゾーンでデプロイに失敗します。

一度にデプロイする先のホストの数を計算するために、CodeDeploy は「ゾーンあたりの正常なホストの最小数」と「正常なホストの最小数」の値の両方を使用します。CodeDeploy では、[A] と [B] (以下の [A] と [B]) の計算結果の小さい方を使用します。

```
[A] = [total-hosts] - [min-healthy-hosts] =
```

```
        [number-of-hosts-to-deploy-to-at-once]
```

```
[B] = [total-hosts-per-AZ] - [min-healthy-hosts-per-AZ] =
        [number-of-hosts-to-deploy-to-at-once-per-AZ]
```

一度にデプロイする先のホストの数を決定すると、CodeDeploy はその数のホストに (一度に 1 つのアベイラビリティーゾーンずつ) バッチでデプロイします。オプションでゾーン間に一時停止 (または「ベイク時間」) を挟みます。

例

デプロイが以下のように設定されている場合:

- [total-hosts] は 200
- [minimum-healthy-hosts] は 160
- [total-hosts-per-AZ] は 100
- [minimum-healthy-hosts-per-AZ] は 50

結果...

- [A] = 200 - 160 = 40
- [B] = 100 - 50 = 50
- 40 は 50 より少ない

したがって、CodeDeploy は一度に 40 ホストにデプロイします。

このシナリオでは、次のようにデプロイされます。

1. CodeDeploy は最初のアベイラビリティーゾーンにデプロイします。
   a. CodeDeploy は最初の 40 ホストにデプロイします。
   b. CodeDeploy は次の 40 ホストにデプロイします。
   c. CodeDeploy は残りの 20 ホストにデプロイします。

      これで、1 つ目のアベイラビリティーゾーンへのデプロイが完了しました。
2. (オプション) CodeDeploy は、[監視期間] または [最初のゾーン監視期間を追加] の設定で定義されているように、最初のゾーンへのデプロイが「ベイク」されるまで待機します。問題がなければ、CodeDeploy は続行します。

3. CodeDeploy は 2 番目のアベイラビリティーゾーンにデプロイします。
   a. CodeDeploy は最初の 40 ホストにデプロイします。
   b. CodeDeploy は次の 40 ホストにデプロイします。
   c. CodeDeploy は残りの 20 ホストにデプロイします。

      これで、2 番目かつ最後のアベイラビリティーゾーンへのデプロイが完了しました。

ゾーン設定機能の詳細と、アベイラビリティーゾーンあたりの正常なインスタンスの最小数を指定する方法については、「zonal configuration」を参照してください。

CodeDeploy でデプロイ設定を使用する

デプロイ設定とは、デプロイ中に CodeDeploy が使用する一連のルール、成功条件、および失敗条件です。これらのルールや条件は、EC2/オンプレミスのコンピューティングプラットフォーム、AWS Lambda コンピューティングプラットフォーム、または Amazon ECS コンピューティングプラットフォームのいずれかにデプロイするかによって、異なります。

EC2/オンプレミスコンピューティングプラットフォームのデプロイ設定

EC2/オンプレミスコンピューティングプラットフォームにデプロイする場合、デプロイ設定では、「正常なホストの最小数」とオプションの「ゾーンあたりの正常なホストの最小数」の値を使用することで、デプロイ中のどの時点でも使用可能でなければならないインスタンスの数または割合 (%) を指定します。

が提供する 3 つの事前定義されたデプロイ設定のいずれかを使用する AWS か、カスタムデプロイ設定を作成できます。カスタムデプロイ設定の作成の詳細については、「Create a Deployment Configuration」を参照してください。デプロイ設定を指定しない場合、CodeDeploy は、CodeDeployDefault.OneAtATime デプロイ設定を使用します。

CodeDeploy がデプロイ中にインスタンスのヘルス状態をモニタリングおよび評価する方法の詳細については、「Instance Health」を参照してください。 AWS アカウントに既に登録されているデプロイ設定のリストを表示するには、「」を参照してくださいView Deployment Configuration Details。

EC2/オンプレミスコンピューティングプラットフォームの事前定義されたデプロイ設定

次の表は、定義済みのデプロイ設定を一覧表示します。

> **Note**
>
> zonal configuration 機能 (アベイラビリティーゾーンあたりの正常なホストの数を指定できる機能) をサポートする定義済みのデプロイ設定はありません。この機能を使用する場合は、独自のデプロイ設定を作成する必要があります。

Deployment configuration	説明
CodeDeployDefault.AllAtOnce	インプレースデプロイ: 一度に可能な限り多くのインスタンスへアプリケーションリビジョンをデプロイするよう試みます。アプリケーションリビジョンが 1 つ以上のインスタンスにデプロイされる場合、デプロイ全体のステータスは Succeeded として表示されます。アプリケーションリビジョンがいずれのインスタンスにもデプロイされない場合、デプロイ全体のステータスは Failed として表示されます。9 つのインスタンスの例を使い、CodeDeployDefault.AllAtOnce は、一度に 9 つのインスタンスすべてにデプロイするよう試みます。インスタンスへのデプロイが 1 つでも成功すると、デプロイ全体は成功します。9 つすべてのインスタンスへのデプロイが失敗した場合に限り失敗します。 ブルー/グリーンデプロイ • 置き換え先環境へのデプロイ: インプレースデプロイの CodeDeployDefault.AllAtOnce と同じデプロイルールに従います。 • トラフィックの再ルーティング: 置き換え先環境のすべてのインスタンスに一度にトラフィックをルーティングします。トラフィックが少なくとも 1 つのインスタンスに正常に再ルーティングされた場合が成功です。すべてのインスタンスへの再ルーティングが失敗した時点で失敗です。
CodeDeployDefault.HalfAtATime	インプレースデプロイ: 一度に最大半分のインスタンスにデプロイします (端数は切り捨てられます)。デプロイ全体は、アプリケーションリビジョンが少なくとも

Deployment configuration	説明
	半分のインスタンスにデプロイされた場合は成功です (端数は切り捨てられます)。それ以外の場合、デプロイは失敗です。9 つのインスタンスの例では、4 つまでのインスタンスに同時にデプロイされます。デプロイ全体は 5 つ以上のインスタンスへのデプロイが成功した場合は成功です。それ以外の場合、デプロイは失敗です。 **Note** 複数の Auto Scaling グループのインスタンスにデプロイする場合、CodeDeploy は属する Auto Scaling グループに関係なく、一度に最大半分のインスタンスにデプロイします。例えば、Auto Scaling グループが `ASG1` と `ASG2` の 2 つあり、それぞれに 10 個のインスタンスがあるとします。このシナリオでは、CodeDeploy は `ASG1` のみで 10 個のインスタンスにデプロイできますが、少なくとも半数のインスタンスにデプロイされているため、成功と見なすことができます。 ブルー/グリーンデプロイ • 置き換え先環境へのデプロイ: インプレースデプロイの CodeDeployDefault.HalfAtATime と同じデプロイルールに従います。 • トラフィックの再ルーティング: 置き換え先環境の最大半分のインスタンスに一度にトラフィックをルーティングします。少なくとも半分のインスタンスへの再ルーティングが成

Deployment configuration	説明
	功した場合が成功です。それ以外の場合、は失敗します。

Deployment configuration	説明
CodeDeployDefault.OneAtATime	インプレースデプロイ: 一度に 1 つのインスタンスにのみアプリケーションリビジョンをデプロイします。 複数のインスタンスを含むデプロイグループの場合。 • デプロイ全体はアプリケーションリビジョンがすべてのインスタンスへデプロイされた場合、成功します。このルールの例外は、最後のインスタンスへのデプロイが失敗した場合に、デプロイ全体が成功することです。これは CodeDeploy が一度に 1 つのインスタンスのみ CodeDeployDefault.OneAtATime 設定でオフラインにするためです。 • デプロイ全体はアプリケーションリビジョンが最後のインスタンス以外へのデプロイに失敗すると、ただちに失敗します。 • 9 つのインスタンスを使用する例では、1 つのインスタンスに同時にデプロイされます。最初の 8 つのインスタンスへのデプロイが成功すると、デプロイ全体は成功します。最初の 8 つのインスタンスのいずれかへのデプロイが失敗すると、デプロイ全体は失敗します。 1 つのインスタンスのみを含むデプロイグループでは、1 つのインスタンスへのデプロイが成功した場合にのみ、デプロイは全体は成功します。 ブルー/グリーンデプロイ

Deployment configuration	説明
	• 置き換え先環境へのデプロイ: インプレースデプロイの CodeDeployDefault.OneAtATime と同じデプロイルールに従います。 • トラフィックの再ルーティング: 置き換え先環境で一度に 1 つのインスタンスにトラフィックをルーティングします。トラフィックがすべての置き換え先インスタンスに正常に再ルーティングされた場合が成功です。最初に再ルーティングが失敗した時点で失敗です。このルールの例外は、最後のインスタンスが登録に失敗しても、デプロイ全体が成功することです。

Amazon ECS コンピューティングプラットフォームのデプロイ設定

Amazon ECS コンピューティングプラットフォームにデプロイする場合、デプロイ設定より、更新された Amazon ECS タスクセットにトラフィックを移行する方法を指定します。canary、リニア、または一度にすべてのデプロイ設定を使用してトラフィックをシフトできます。詳細については、「Deployment configuration」を参照してください。

独自の Canary または線形のデプロイ設定を作成することもできます。詳細については、「Create a Deployment Configuration」を参照してください。

Amazon ECS コンピューティングプラットフォームの事前定義されたデプロイ設定

以下の表に、Amazon ECS のデプロイで利用できる事前定義された設定を一覧表示します。

> **Note**
>
> Network Load Balancer を使用する場合、`CodeDeployDefault.ECSAllAtOnce` 優先デプロイ設定のみがサポートされます。

Deployment configuration	説明
CodeDeployDefault.ECSLinear10PercentEvery1Minutes	すべてのトラフィックが移行されるまで、毎分トラフィックの 10 パーセントを移行します。
CodeDeployDefault.ECSLinear10PercentEvery3Minutes	すべてのトラフィックが移行されるまで、3 分ごとにトラフィックの 10 パーセントを移行します。
CodeDeployDefault.ECSCanary10Percent5Minutes	最初の増分でトラフィックの 10 パーセントを移行します。残りの 90 パーセントは 5 分後にデプロイされます。
CodeDeployDefault.ECSCanary10Percent15Minutes	最初の増分でトラフィックの 10 パーセントを移行します。残りの 90% は 15 分後にデプロイされます。
CodeDeployDefault.ECSAllAtOnce	すべてのトラフィックを同時に更新済み Amazon ECS コンテナに移行します。

CloudFormation Blue/Green デプロイのデプロイ設定 (Amazon ECS)

CloudFormation Blue/Green デプロイを介して Amazon ECS コンピューティングプラットフォームにデプロイする場合、デプロイ設定は、更新された Amazon ECS コンテナにトラフィックを移行する方法を指定します。canary、リニア、または一度にすべてのデプロイ設定を使用してトラフィックをシフトできます。詳細については、「Deployment configuration」を参照してください。

Blue CloudFormation /Green デプロイでは、独自のカスタム Canary または線形デプロイ設定を作成することはできません。 CloudFormation を使用して Amazon ECS ブルー/グリーンデプロイを管理するstep-by-stepについては、 CloudFormation ユーザーガイドの「 を使用した CodeDeploy による ECS ブルー/グリーンデプロイの自動化 CloudFormation」を参照してください。

> **Note**
>
> を使用した Amazon ECS ブルー/グリーンデプロイの管理 CloudFormation は、欧州 (ミラノ)、アフリカ (ケープタウン)、およびアジアパシフィック (大阪) リージョンでは利用できません。

AWS Lambda コンピューティングプラットフォームでのデプロイ設定

AWS Lambda コンピューティングプラットフォームにデプロイする場合、デプロイ設定は、アプリケーションの新しい Lambda 関数バージョンにトラフィックを移行する方法を指定します。canary、リニア、または一度にすべてのデプロイ設定を使用してトラフィックをシフトできます。詳細については、「Deployment configuration」を参照してください。

独自の Canary または線形のデプロイ設定を作成することもできます。詳細については、「Create a Deployment Configuration」を参照してください。

AWS Lambda コンピューティングプラットフォームの事前定義されたデプロイ設定

次の表に、 AWS Lambda のデプロイで利用できる事前定義された設定を一覧表示します。

Deployment configuration	説明
CodeDeployDefault.LambdaCanary10Percent5Minutes	最初の増分でトラフィックの 10 パーセントを移行します。残りの 90 パーセントは 5 分後にデプロイされます。
CodeDeployDefault.LambdaCanary10Percent10Minutes	最初の増分でトラフィックの 10 パーセントを移行します。残りの 90 パーセントは 10 分後にデプロイされます。
CodeDeployDefault.LambdaCanary10Percent15Minutes	最初の増分でトラフィックの 10 パーセントを移行します。残りの 90% は 15 分後にデプロイされます。

Deployment configuration	説明
CodeDeployDefault.LambdaCanary10Percent30Minutes	最初の増分でトラフィックの 10 パーセントを移行します。残りの 90 パーセントは 30 分後にデプロイされます。
CodeDeployDefault.LambdaLinear10PercentEvery1Minute	すべてのトラフィックが移行されるまで、毎分トラフィックの 10 パーセントを移行します。
CodeDeployDefault.LambdaLinear10PercentEvery2Minutes	すべてのトラフィックが移行されるまで、2 分ごとにトラフィックの 10 パーセントを移行します。
CodeDeployDefault.LambdaLinear10PercentEvery3Minutes	すべてのトラフィックが移行されるまで、3 分ごとにトラフィックの 10 パーセントを移行します。
CodeDeployDefault.LambdaLinear10PercentEvery10Minutes	すべてのトラフィックが移行されるまで、10 分ごとにトラフィックの 10 パーセントを移行します。
CodeDeployDefault.LambdaAllAtOnce	すべてのトラフィックは、更新された Lambda 関数に一度に移行します。

トピック

- Create a Deployment Configuration
- View Deployment Configuration Details
- Delete a Deployment Configuration

CodeDeploy を使用してデプロイ設定を作成する

CodeDeploy に用意してあるデフォルトのデプロイ設定のいずれも使用しない場合は、次の手順を実行して独自のデプロイ設定を作成できます。

CodeDeploy コンソール AWS CLI、CodeDeploy APIs、または CloudFormation テンプレートを使用して、カスタムデプロイ設定を作成できます。

CloudFormation テンプレートを使用してデプロイ設定を作成する方法については、「」を参照してくださいCloudFormation CodeDeploy リファレンスの テンプレート。

トピック

- デプロイ設定の作成 (コンソール)
- CodeDeploy を使用してデプロイ設定を作成する (AWS CLI)

デプロイ設定の作成 (コンソール)

AWS コンソールを使用してデプロイ設定を作成するには、次の手順を実行します。

コンソールを使用して CodeDeploy でデプロイ設定を作成するには

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで [デプロイ設定] を選択します。

   組み込みのデプロイ設定のリストが表示されます。
3. [Create deployment configuration (デプロイ設定の作成)] を選択します。
4. [デプロイ設定名] に、デプロイ設定の名前を入力します。例えば、**`my-deployment-config`**。
5. [コンピューティングプラットフォーム] で、次のいずれかを選択します。

   - EC2/オンプレミス
   - AWS Lambda
   - Amazon ECS
6. 次のいずれかを行います。

   - EC2/オンプレミスを選択した場合:
     1. [正常なホストの最小数] で、デプロイ中のどの時点でも使用可能でなければならないインスタンスの数または割合を指定します。CodeDeploy がデプロイ中にインスタンスのヘルス状態をモニタリングおよび評価する方法の詳細については、「Instance Health」を参照してください。

2. (オプション) [ゾーン設定] で、[ゾーン設定を有効にする] を選択すると、CodeDeploy は AWS リージョン内で一度に 1 つのアベイラビリティーゾーンにアプリケーションをデプロイします。一度に 1 つのアベイラビリティーゾーンにデプロイすることで、デプロイのパフォーマンスと実行可能性に対する信頼が高まるにつれて、デプロイを徐々に多くのユーザーに公開できます。ゾーン設定を有効にしない場合、CodeDeploy はリージョン内のランダムに選択されたホストにアプリケーションをデプロイします。

   ゾーン設定機能を有効にする場合は、次の点に注意してください。

   - ゾーン設定機能は、Amazon EC2 インスタンスへのインプレースデプロイでのみサポートされます (ブルー/グリーンデプロイやオンプレミスインスタンスはサポートされません)。インプレイスデプロイの詳細については、「デプロイタイプ」を参照してください。
   - ゾーン設定機能は、事前定義済みのデプロイ設定ではサポートされません。ゾーン設定を使用するには、ここで説明しているように、カスタムデプロイ設定を作成する必要があります。
   - CodeDeploy は、デプロイをロールバックする必要がある場合、ランダムなホストにロールバック操作を実行します (CodeDeploy は、想定とは異なり、一度に 1 つのゾーンを順にロールバックしません)。このロールバック動作は、パフォーマンス上の理由から選択されています。ロールバックの詳細については、「CodeDeploy を使用した再デプロイおよびデプロイのロールバック」を参照してください。

3. [ゾーン設定を有効にする] チェックボックスをオンにした場合は、オプションで以下のオプションを指定します。

   - (オプション) [監視期間] で、CodeDeploy が 1 つのアベイラビリティーゾーンへのデプロイを完了してから待機する時間を秒単位で指定します。CodeDeploy は、次のアベイラビリティーゾーンへのデプロイを開始する前に、この時間だけ待機します。デプロイが 1 つのアベイラビリティーゾーンで完了して次のゾーンに進む前に、一定時間を確保して完了を実証 (ベイク) する場合は、監視期間を追加することを検討します。監視期間を指定しない場合、CodeDeploy は次のアベイラビリティーゾーンへのデプロイをすぐに開始します。[監視期間] 設定の仕組みについては、「アベイラビリティーゾーンあたりの正常なインスタンスの最小数について」を参照してください。
   - (オプション) [最初のゾーンに監視期間を追加] を選択すると、最初のアベイラビリティーゾーンにのみ監視期間が設定されます。このオプションは、1 つ目のアベイラビリティーゾーンのベイク時間を確保する場合に設定できます。[最初のゾーンの監視期間を追加] に値を指定しない場合、CodeDeploy は最初のアベイラビリティーゾーンの [監視期間] の値を使用します。

- (オプション) [ゾーンあたりの正常なホストの最小数] で、デプロイ中に使用可能でなければならない、アベイラビリティーゾーンあたりのインスタンスの数または割合を指定します。FLEET_PERCENT を選択してパーセンテージを指定するか、HOST_COUNT を選択して数値を指定します。このフィールドは [正常なホストの最小数] フィールドと連動します。詳細については、「アベイラビリティーゾーンあたりの正常なインスタンスの最小数について」を参照してください。

  [ゾーンあたりの正常なホストの最小数] に値を指定しない場合、CodeDeploy はデフォルト値の `0` パーセントを使用します。

- [AWS Lambda] または [Amazon ECS] を選択した場合:

  1. [タイプ]] で、[リニア] または [Canary] を選択します。
  2. [ステップ] フィールドと [間隔] フィールドで、次のいずれかを行います。
     - [Canary] を選択した場合は、[ステップ] に、移行するトラフィックのパーセンテージを 1 ~ 99 で入力します。この値は、最初の増分で移行されるトラフィックの割合を示します。残りのトラフィックは、2 回目の増分で、選択した間隔後に移行されます。

       [間隔] に、1 番目と 2 番目のトラフィック移行間の分数を入力します。
     - [リニア] を選択した場合は、[ステップ] に、移行するトラフィックのパーセンテージを 1 ~ 99 で入力します。この値は、各間隔の開始時に移行されるトラフィックの割合を示します。

       [間隔] に、各増分移行間の分数を入力します。

7. [Create deployment configuration (デプロイ設定の作成)] を選択します。

   これで、デプロイグループに関連付けることができるデプロイ設定が整いました。

CodeDeploy を使用してデプロイ設定を作成する (AWS CLI)

を使用してデプロイ設定 AWS CLI を作成するには、create-deployment-config コマンドを呼び出します。

次の例では、`ThreeQuartersHealthy` という名前の EC2/オンプレミスデプロイ設定を作成します。このデプロイ設定では、デプロイ中にターゲットインスタンスの 75% が常に正常であることが要求されます。

```
aws deploy create-deployment-config --deployment-config-name ThreeQuartersHealthy --
minimum-healthy-hosts type=FLEET_PERCENT,value=75
```

次の例では、300Total50PerAZ という名前の EC2/オンプレミスデプロイ設定を作成します。このデプロイ設定では、デプロイあたり合計 300 個のターゲットインスタンス、およびアベイラビリティーゾーンあたり 50 個のターゲットインスタンスが常に正常であることが要求されます。また、監視期間を 1 時間に設定します。

```
aws deploy create-deployment-config --deployment-config-name 300Total50PerAZ
 --minimum-healthy-hosts type=HOST_COUNT,value=300 --zonal-config
 '{"monitorDurationInSeconds":3600,"minimumHealthyHostsPerZone":
{"type":"HOST_COUNT","value":50}}'
```

次の例では、 という名前の AWS Lambda デプロイ設定を作成しますCanary25Percent45Minutes。この際、最初の増分でトラフィックの 25 パーセントを移行する Canary トラフィックを使用します。残りの 75 パーセントは 45 分後に移行されます。

```
aws deploy create-deployment-config --deployment-config-name Canary25Percent45Minutes
 --traffic-routing-config
 "type="TimeBasedCanary",timeBasedCanary={canaryPercentage=25,canaryInterval=45}" --
compute-platform Lambda
```

次の例では、Canary25Percent45Minutes という名前の Amazon ECS デプロイ設定を作成します。この際、最初の増分でトラフィックの 25 パーセントを移行する Canary トラフィックを使用します。残りの 75 パーセントは 45 分後に移行されます。

```
aws deploy create-deployment-config --deployment-config-name Canary25Percent45Minutes
 --traffic-routing-config
 "type="TimeBasedCanary",timeBasedCanary={canaryPercentage=25,canaryInterval=45}" --
compute-platform ECS
```

CodeDeploy によるデプロイ設定の詳細の表示

CodeDeploy コンソール、 AWS CLI、または CodeDeploy APIs を使用して、 AWS アカウントに関連付けられたデプロイ設定の詳細を表示できます。事前定義された CodeDeploy デプロイ設定の説

明については、「EC2/オンプレミスコンピューティングプラットフォームの事前定義されたデプロイ設定」を参照してください。

トピック

- デプロイ設定の詳細の表示 (コンソール)
- デプロイ設定の表示 (CLI)

デプロイ設定の詳細の表示 (コンソール)

CodeDeploy コンソールを使用して、デプロイ設定名の一覧を表示するには:

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで [デプロイ] を展開し、[デプロイ設定] を選択します。

   コンソールで、各デプロイ設定のデプロイ設定名および条件の一覧を表示できます。

   > **Note**
   >
   > エントリが表示されない場合は、正しいリージョンが選択されていることを確認しましょう。ナビゲーションバーのリージョンセレクターで、「AWS 全般のリファレンス」の「リージョンとエンドポイント」に一覧表示されているいずれかのリージョンを選択します。CodeDeploy は、これらのリージョンでのみサポートされています。

デプロイ設定の表示 (CLI)

を使用してデプロイ設定の詳細 AWS CLI を表示するには、 `get-deployment-config` コマンドまたは `list-deployment-configs` コマンドを呼び出します。

1 つのデプロイ設定に関する詳細を表示するには、get-deployment-config コマンドを呼び出し、一意のデプロイ設定名を指定します。

複数のデプロイ設定に関する詳細を表示するには、list-deployments コマンドを呼び出します。

CodeDeploy を使用してデプロイ設定を削除する

AWS CLI または CodeDeploy APIs を使用して、 AWS アカウントに関連付けられたカスタムデプロイ設定を削除できます。CodeDeployDefault.AllAtOnce、CodeDeployDefault.HalfAtATime、CodeDeployDefault などの組み込みのデプロイ設定は削除できません。

> **Warning**
>
> まだ使用中のカスタムデプロイ設定は削除できません。未使用のカスタムデプロイ設定を削除すると、それを新しいデプロイおよび新規デプロイグループに関連付けることはできなくなります。このアクションを元に戻すことはできません。

を使用してデプロイ設定 AWS CLI を削除するには、delete-deployment-config コマンドを呼び出し、デプロイ設定名を指定します。デプロイ設定名のリストを表示するには、list-deployment-configs コマンドを呼び出します。

次の例では、ThreeQuartersHealthy という名前のデプロイ設定を削除します。

```
aws deploy delete-deployment-config --deployment-config-name ThreeQuartersHealthy
```

CodeDeploy 内でアプリケーションを用いて作業をする

インスタンスを設定したら、リビジョンをデプロイする前に CodeDeploy でアプリケーションを作成する必要があります。アプリケーション は、CodeDeploy によって使用される単なる名称もしくはコンテナであり、その目的は正しいリビジョン、デプロイ設定、およびデプロイグループが、デプロイ中に正しく参照されることにあります。

次のステップ用に、以下の表の情報を使用します。

コンピューティングプラットフォーム	シナリオ	次のステップの情報
EC2/オンプレミス	まだインスタンスを作成していません。	「CodeDeploy のためにインスタンスを用いた操作」を参照してから、このページに戻ってください。
EC2/オンプレミス	インスタンスは作成しましたが、タグ付けが完了していません。	「Tagging Instances for Deployments」を参照してから、このページに戻ってください。
EC2/オンプレミス, AWS Lambda, および Amazon ECS	まだアプリケーションを作成していません。	「CodeDeploy でアプリケーションを作成する」を参照してください。
EC2/オンプレミス, AWS Lambda, および Amazon ECS	既にアプリケーションを作成しましたが、まだデプロイグループを作成していません。	「CodeDeploy でデプロイグループを作成する」を参照してください。
EC2/オンプレミス, AWS Lambda, および Amazon ECS	既にアプリケーションとデプロイグループを作成しましたが、アプリケーションリビジョンを作成していません。	「CodeDeploy のアプリケーションリビジョンの操作」を参照してください。
EC2/オンプレミス, AWS Lambda, および Amazon ECS	既にアプリケーションとデプロイグループを作成し、アプリケーションリビジョンを	「CodeDeploy でデプロイを作成する」を参照してください。

コンピューティングプラットフォーム	シナリオ	次のステップの情報
	アップロードしました。デプロイの準備が整いました。	

トピック

- CodeDeploy でアプリケーションを作成する
- CodeDeploy を使用してアプリケーション詳細を表示する
- 通知ルールの作成
- CodeDeploy アプリケーション名の変更
- CodeDeploy でアプリケーションを削除する

CodeDeploy でアプリケーションを作成する

アプリケーション は、CodeDeploy によって使用される単なる名称もしくはコンテナであり、その目的は正しいリビジョン、デプロイ設定、およびデプロイグループが、デプロイ中に正しく参照されることにあります。CodeDeploy コンソール、 AWS CLI、CodeDeploy APIs、または CloudFormation テンプレートを使用してアプリケーションを作成できます。

コードまたはアプリケーションリビジョンは、デプロイと呼ばれるプロセスを通してインスタンスにインストールされます。CodeDeploy では、2 種類のデプロイがサポートされます。

- インプレイスデプロイ: デプロイグループの各インスタンス上のアプリケーションが停止され、最新のアプリケーションリビジョンがインストールされて、新バージョンのアプリケーションが開始され検証されます。ロードバランサーを使用し、デプロイ中はインスタンスが登録解除され、デプロイ完了後にサービスに復元されるようにできます。EC2 オンプレミスコンピューティングプラットフォームを使用するデプロイのみが、インプレイスデプロイを使用できます。インプレイスデプロイの詳細については、「インプレースデプロイの概要」を参照してください。
- Blue/Green デプロイ: デプロイの動作は、使用するコンピューティングプラットフォームにより異なります。
  - EC2 オンプレミスコンピューティングプラットフォームの Blue/Green: 以下のステップを使用して、デプロイグループのインスタンス (元の環境) がインスタンスの別のセット (置き換え先環境) に置き換えられます。
    - 置き換え先の環境のインスタンスがプロビジョニングされます。

- 最新のアプリケーションリビジョンは、置き換え先インスタンスにインストールされます。
- オプションの待機時間は、アプリケーションのテストやシステム検証などのアクティビティに対して発生します。
- 置き換え先環境のインスタンスは、1 つまたは複数の Elastic Load Balancing ロードバランサーに登録され、トラフィックは、それらに再ルーティングされます。元の環境のインスタンスは、登録が解除され、終了するか、他の使用のために実行することができます。

> **Note**
>
> EC2/オンプレミスのコンピューティングプラットフォームを使用する場合は、blue/green デプロイが Amazon EC2 インスタンスでのみ機能することに注意してください。

- AWS Lambda または Amazon ECS コンピューティングプラットフォームの Blue/Green: トラフィックは、Canary、線形、または all-at-onceのデプロイ設定に従って増分でシフトされます。
- 経由のブルー/グリーンデプロイ CloudFormation: CloudFormation スタックの更新の一環として、トラフィックは現在のリソースから更新されたリソースに移行されます。現時点では、ECS blue/green デプロイのみがサポートされています。

ブルー/グリーンデプロイの詳細については、「Blue/Green デプロイの概要」を参照してください。

CodeDeploy コンソールを使用してアプリケーションを作成する場合は、最初のデプロイグループを同時に設定します。を使用してアプリケーション AWS CLI を作成する場合は、別のステップで最初のデプロイグループを作成します。

AWS アカウントに既に登録されているアプリケーションのリストを表示するには、「」を参照してくださいCodeDeploy を使用してアプリケーション詳細を表示する。 CloudFormation テンプレートを使用してアプリケーションを作成する方法については、「」を参照してくださいCloudFormation CodeDeploy リファレンスの テンプレート。

デプロイタイプはいずれも、どの送信先にも適用されません。以下の表に、3 種類のデプロイ送信先のデプロイを指定するデプロイタイプを示します。

デプロイ送信先	インプレース	Blue/Green
Amazon EC2	はい	はい

デプロイ送信先	インプレース	Blue/Green
オンプレミス	はい	いいえ
サーバーレス AWS Lambda 関数	いいえ	はい
Amazon ECS アプリケーション	いいえ	はい

トピック

- インプレースデプロイ (コンソール) 用のアプリケーションを作成
- Blue/Green デプロイ (コンソール) のアプリケーションを作成します。
- Amazon ECS サービスデプロイ用のアプリケーションを作成 (コンソール)
- AWS Lambda 関数デプロイ用のアプリケーションを作成する (コンソール)
- アプリケーションの作成 (CLI)

インプレースデプロイ (コンソール) 用のアプリケーションを作成

CodeDeploy コンソールを使用して、インプレースデプロイ用のアプリケーションを作成するには

> **Warning**
>
> 次の場合は、これらの手順を実行しないでください。
>
> - インスタンスを CodeDeploy デプロイで使用する準備ができていません。インスタンスをセットアップするには、CodeDeploy のためにインスタンスを用いた操作 の指示に従い、その後にこのトピックの手順に従います。
> - カスタムデプロイ設定を使用するアプリケーションを作成する必要があり、まだデプロイ設定を作成していません。Create a Deployment Configuration の指示に従った後に、このトピックの手順に従います。
> - 最低限必要な信頼およびアクセス権限を持つ CodeDeploy を信頼するサービスロールがありません。必要なアクセス許可を持つサービスロールを作成し、設定するには、「ステッ

プ 2: CodeDeployのサービスのロールを作成する」の手順に従って、このトピックの手順に戻ります。

- インプレースデプロイのために Elastic Load Balancing で、Classic Load Balancer、Application Load Balancer、または Network Load Balancer を選択したいが、まだ作成していない場合。

CodeDeploy コンソールを使用して、インプレースデプロイ用のアプリケーションを作成するには

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > CodeDeploy の開始方法 で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで、[デプロイ]、[ご利用開始にあたって] の順に選択します。
3. [アプリケーションを作成] を選択します。
4. [アプリケーション名] に、アプリケーションの名前を入力します。
5. [Compute platform (コンピューティングプラットフォーム)] で [EC2/On-Premises (EC2/オンプレミス)] を選択します。
6. [アプリケーションを作成] を選択します。
7. アプリケーションのページで、[デプロイグループ] タブの [デプロイグループの作成] を選択します。
8. [デプロイグループ名] に、デプロイグループを表す名前を入力します。

   > **Note**
   >
   > 他のデプロイグループで使用されている設定 (デプロイグループ名、タグ、Amazon EC2 Auto Scaling グループ名、または両方、およびデプロイ設定を含む) を使用する場合は、このページでこれらの設定を指定します。この新しいデプロイグループと既存のデプロイグループは名前が同じでも、それぞれが別のアプリケーションに関連付けられるため、CodeDeploy では別のデプロイグループとして扱われます。

9. [サービスロール] で、ターゲットインスタンスへのアクセス権を CodeDeploy に付与するサービスロールを選択します。

10. [デプロイタイプ] で、[インプレース] を選択します。
11. [環境設定] で、以下のいずれかを選択します。

    a. Amazon EC2 Auto Scaling グループ: アプリケーションリビジョンをデプロイする Amazon EC2 Auto Scaling グループの名前を入力または選択します。Amazon EC2 Auto Scaling グループの一部として新しい Amazon EC2 インスタンスが起動されると、CodeDeploy ではリビジョンを新しいインスタンスに自動的にデプロイできます。デプロイグループには最大 10 個の Amazon EC2 Auto Scaling グループを追加できます。

    b. [Amazon EC2 インスタンス] または [オンプレミスインスタンス]: [キー] と [値] フィールドに、インスタンスにタグを付けるために使用したキーバリューペアの値を入力します。単一タググループで最大 10 個のキーと値のペアをタグ付けできます。

       i. [値] フィールドでワイルドカードを使用して、似ている Amazon EC2 インスタンス、コストセンター、グループ名などの特定のパターンでタグ付けされているすべてのインスタンスを識別できます。例えば、キー フィールドに 名前 を選択し、値 フィールドに **GRP-*a** を入力すると、CodeDeploy は **GRP-1a**、**GRP-2a**、および **GRP-XYZ-a** などそのパターンに当てはまるすべてのインスタンスを特定します。

       ii. [値] フィールドでは、大文字と小文字が区別されます。

       iii. リストからキーと値のペアを削除するには、[タグの削除] を選択します。

       CodeDeploy で、指定されたキーバリューの各ペアまたは Amazon EC2 Auto Scaling グループ名に一致するインスタンスが検出されると、一致したインスタンスの数が表示されます。インスタンスに関する詳細情報を表示するには、その数をクリックします。

       デプロイ先のインスタンスの基準を絞り込む場合は、タググループを追加 を選択してタググループを作成します。タググループは最大 3 つまで作成し、それぞれに最大 10 個のキーと値のペアを指定できます。デプロイグループで複数のタググループを使用する場合は、すべてのタググループによって識別されたインスタンスのみがデプロイグループに含まれます。つまり、インスタンスがデプロイグループに含まれるには、各グループの少なくとも 1 つのタグが一致する必要があります。

       タググループを使用してデプロイグループを絞り込む方法ついては、「Tagging Instances for Deployments」を参照してください。

12. [デプロイ設定] で、アプリケーションをインスタンスをデプロイするレート (例: 一度に 1 つずつ、一度にすべて) を制御するデプロイ設定を選択します。デプロイ設定の詳細については、CodeDeploy でデプロイ設定を使用する を参照してください。

13. (オプション) [ロードバランサー] で [ロードバランシングを有効にする] を選択し、一覧から、CodeDeploy デプロイ中のインスタンスへのトラフィックを管理する Classic Load Balancer、Application Load Balancer のターゲットグループ、Network Load Balancer のターゲットグループを選択します。最大 10 個の Classic Load Balancer と 10 個のターゲットグループとで、合計 20 個のアイテムを選択できます。デプロイする Amazon EC2 インスタンスが、選択したロードバランサー (Classic Load Balancer) またはターゲットグループ (Application Load Balancer および Network Load Balancer) に登録されていることを確認します。

    デプロイ中、元のインスタンスは選択したロードバランサーとターゲットグループから登録解除され、デプロイ中にトラフィックがこれらのインスタンスにルーティングされないようにします。デプロイが完了すると、各インスタンスは選択したすべての Classic Load Balancer とターゲットグループに再登録されます。

    CodeDeploy デプロイでのロードバランサーの詳細については、Integrating CodeDeploy with Elastic Load Balancing を参照してください。
14. (オプション) アドバンストを展開し、デプロイに含めるオプション (Amazon SNS 通知トリガー、Amazon CloudWatch アラーム、自動ロールバックなど) を設定します。

    詳細については、「デプロイグループの詳細オプションの設定」を参照してください。
15. デプロイグループの作成 を選択します。

次のステップでは、アプリケーションおよびデプロイグループにデプロイするリビジョンを準備します。手順については、「CodeDeploy のアプリケーションリビジョンの操作」を参照してください。

Blue/Green デプロイ (コンソール) のアプリケーションを作成します。

CodeDeploy コンソールを使用して、Blue/Green デプロイ用のアプリケーションを作成するには

> **Note**
>
> AWS Lambda コンピューティングプラットフォームへのデプロイは常に Blue/Green デプロイです。デプロイタイプオプションは指定しません。

> **Warning**
>
> 次の場合は、これらの手順を実行しないでください。

- Blue/Green デプロイプロセス中に置き換えたい CodeDeploy エージェントがインストールされたインスタンスはありません。インスタンスをセットアップするには、CodeDeploy のためにインスタンスを用いた操作 の指示に従い、その後にこのトピックの手順に従います。
- カスタムデプロイ設定を使用するアプリケーションを作成する必要があり、まだデプロイ設定を作成していません。Create a Deployment Configuration の指示に従った後に、このトピックの手順に従います。
- 少なくとも、「ステップ 2: CodeDeployのサービスのロールを作成する」に記載されている信頼とアクセス権限を持つ、CodeDeploy を信頼するサービスロールがない。サービスロールを作成して設定するには、ステップ 2: CodeDeployのサービスのロールを作成する の指示に従い、その後にこのトピックの手順に従います。
- 置き換え先環境でインスタンスを登録するために、Elastic Load Balancing で Classic Load Balancer、Application Load Balancer、または Network Load Balancer を作成していません。詳細については、「CodeDeploy Amazon EC2 デプロイ用の Elastic Load Balancing でロードバランサーをセットアップする」を参照してください。

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > CodeDeploy の開始方法 で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで、[デプロイ]、[ご利用開始にあたって] の順に選択します。
3. [アプリケーション名] に、アプリケーションの名前を入力します。
4. [Compute platform (コンピューティングプラットフォーム)] で [EC2/On-Premises (EC2/オンプレミス)] を選択します。
5. [アプリケーションを作成] を選択します。
6. アプリケーションのページで、[デプロイグループ] タブの [デプロイグループの作成] を選択します。
7. [デプロイグループ名] に、デプロイグループを表す名前を入力します。

> **Note**
>
> 他のデプロイグループで使用されているのと同じ設定 (デプロイグループ名タグ、Amazon EC2 Auto Scaling グループ名、デプロイ設定など) を使用する場合は、このページでこれらの設定を選択します。この新しいデプロイグループと既存のデプロイグループは名前が同じでも、それぞれが別のアプリケーションに関連付けられるため、CodeDeploy では別のデプロイグループとして扱われます。

8. サービスロール で、ターゲットインスタンスへのアクセス権を CodeDeploy に付与するサービスロールを選択します。
9. [デプロイタイプ] で [Blue/Green] を選択します。
10. [Environment configuration] で、置き換え先環境にインスタンスを提供するために使用する方法を選択します:

    a. Amazon EC2 Auto Scaling グループを自動的にコピーする: CodeDeploy は、ユーザーが指定したグループをコピーして Amazon EC2 Auto Scaling グループを作成します。

    b. [Manually provision instances]: デプロイを作成するまで置き換え先環境のインスタンスを特定しません。デプロイを開始する前に、インスタンスを作成する必要があります。代わりに、ここで置換するインスタンスを指定します。

11. ステップ 10 での選択内容に応じて、次のいずれかを実行します:

    - [Amazon EC2 Auto Scaling グループを自動コピーする] を選択している場合: [Amazon EC2 Auto Scaling グループ] で、置き換え先環境のインスタンス用に作成される Amazon EC2 Auto Scaling グループのテンプレートとして使用したい Amazon EC2 Auto Scaling グループの名前を選択または入力します。選択した Amazon EC2 Auto Scaling グループ内の現在正常なインスタンスの数が、置き換え先環境で作成されます。
    - インスタンスの手動プロビジョンを選択している場合 : Amazon EC2 Auto Scaling グループと Amazon EC2 インスタンスのいずれかまたは両方を有効にして、このデプロイグループに追加するインスタンスを指定します。元の環境のインスタンス (つまり、置換対象のインスタンスまたは現在のアプリケーションリビジョンを実行しているインスタンス) を識別するための Amazon EC2 タグ値または Amazon EC2 Auto Scaling グループ名を入力します。

12. [ロードバランサー] で [ロードバランシングを有効にする] を選択し、一覧から、代わりの Amazon EC2 インスタンスを登録する Classic Load Balancer、Application Load Balancer のターゲットグループ、Network Load Balancer のターゲットグループを選択します。各代替インスタンスは、選択したすべての Classic Load Balancer とターゲットグループに登録されます。

最大 10 個の Classic Load Balancer と 10 個のターゲットグループとで、合計 20 個のアイテムを選択できます。

トラフィックは、選択した [トラフィック再ルーティング] と [デプロイ設定] に従って、元のインスタンスから代替インスタンスに再ルーティングされます。

CodeDeploy デプロイ用のロードバランサーの詳細については、「Integrating CodeDeploy with Elastic Load Balancing」を参照してください。

13. [Deployment settings] で、置き換え先環境へトラフィックを再ルーティングするためのデフォルトのオプション、デプロイに使用するデプロイ設定、デプロイ後に元の環境のインスタンスを処理する方法を確認します。

    設定を変更する場合は、次のステップに進みます。それ以外の場合は、ステップ 15 に進みます。

14. Blue/Green デプロイのデプロイ設定を変更するには、以下のいずれかの設定を変更します。

設定	オプション
[Traffic rerouting]	• [トラフィックをすぐに再ルーティングする]: 置き換え先環境のインスタンスがプロビジョニングされ、最新のアプリケーションリビジョンがインストールされるとすぐに、指定のロードバランサーとターゲットグループに自動的に登録され、トラフィックがそれらに再ルーティングされます。元の環境内のインスタンスは、登録解除されます。 • [トラフィックを再ルーティングするかどうかを選択します]: 置き換え先環境のインスタンスは、手動でトラフィックを再ルーティングしないかぎり、指定のロードバランサーとターゲットグループに登録されません。指定した待機時間中にトラフィックが再ルーティングされない場合、デプロイステータスは停止に変更されます。

設定	オプション
Deployment configuration	置き換え先環境のインスタンスがロードバランサーとターゲットグループに登録されているレート (個別、一括など) を選択します。 **Note** トラフィックが置き換え先環境に適切にルーティングされた後、元の環境のインスタンスは、どのデプロイ設定が選択されていても一度にすべて登録解除されます。 詳細については、「CodeDeploy でデプロイ設定を使用する」を参照してください。
Original instances	• [デプロイグループの元のインスタンスを終了する]: トラフィックが置き換え先環境に再ルーティングされた後、ロードバランサーとターゲットグループから登録解除されたインスタンスは、指定した待機時間の後に終了します。 • [デプロイグループの元のインスタンスを実行し続ける]: トラフィックが置き換え先環境に再ルーティングされた後、ロードバランサーとターゲットグループから登録解除されたインスタンスは実行されたままになります。

15. (オプション) アドバンスト で、Amazon SNS 通知トリガー、Amazon CloudWatch アラーム、自動ロールバックなど、デプロイに含めたいオプションを設定します。

    デプロイグループの詳細なオプションを指定する方法の詳細については、「デプロイグループの詳細オプションの設定」を参照してください。

16. デプロイグループの作成 を選択します。

次のステップでは、アプリケーションおよびデプロイグループにデプロイするリビジョンを準備します。手順については、「CodeDeploy のアプリケーションリビジョンの操作」を参照してください。

Amazon ECS サービスデプロイ用のアプリケーションを作成 (コンソール)

CodeDeploy コンソールを使用して Amazon ECS サービスデプロイ用のアプリケーションを作成できます。

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで デプロイ を展開して、ご利用開始にあたって を選択します。
3. [アプリケーションの作成] ページで、[CodeDeploy の使用] を選択します。
4. [アプリケーション名] に、アプリケーションの名前を入力します。
5. コンピューティングプラットフォームから、Amazon ECS を選択します。
6. [アプリケーションを作成] を選択します。
7. アプリケーションのページで、[デプロイグループ] タブの [デプロイグループの作成] を選択します。Amazon ECS デプロイのデプロイグループを作成するために必要なものの詳細については、「Amazon ECS デプロイを開始する前に」を参照してください。
8. [デプロイグループ名] に、デプロイグループを表す名前を入力します。

   > **Note**
   >
   > 他のデプロイグループで使用されているのと同じ設定 (デプロイグループ名、タグ、 グループ名、デプロイ設定など) を使用する場合は、このページでこれらの設定を選択します。この新しいグループと既存のグループは名前が同じでも、それぞれが別のアプリケーションに関連付けられるため、CodeDeploy では別のデプロイグループとして扱われます。

9. サービスロール で、CodeDeploy に Amazon ECS へのアクセスを許可するサービスロールを選択します。詳細については、「ステップ 2: CodeDeployのサービスのロールを作成する」を参照してください。

10. ロードバランサーの名前 から、Amazon ECS サービスにトラフィックを提供するロードバランサーの名前を選択します。
11. [本稼働リスナーポート] から、Amazon ECS サービスへの本稼働トラフィックを提供するリスナーのポートとプロトコルを選択します。
12. (オプション) テストリスナーポート から、デプロイ時に Amazon ECS サービス内の置き換えタスクセットにトラフィックを処理するテストリスナーのポートとプロトコルを選択します。`AfterAllowTestTraffic` フック中に実行する Lambda 関数を、AppSpec ファイル内に一つ以上指定できます。この関数は、検証テストを実行できます。検証テストが失敗すると、デプロイのロールバックが発生します。検証テストに成功すると、デプロイのライフサイクルの次のフック `BeforeAllowTraffic` がトリガーされます。テストリスナーポートが指定されていない場合、`AfterAllowTestTraffic` フック中は何も起こりません。詳細については、「Amazon ECS のデプロイ向けの AppSpec の「hooks」セクション」を参照してください。
13. [Target group 1 name] および [Target group 2 name] から、デプロイ時にトラフィックをルーティングするターゲットグループを選択します。CodeDeploy は、1 つのターゲットグループを Amazon ECS サービスの元のタスクセットにバインドし、もう一方をその置き換えタスクセットにバインドします。詳細については、「Application Load Balancer のターゲットグループ」を参照してください。
14. [トラフィックをすぐに再ルーティングする] または [トラフィックを再ルーティングするタイミングを指定する] を選択し、更新された Amazon ECS サービスにトラフィックを再ルーティングするタイミングを決定します。

    [トラフィックをすぐに再ルーティングする] を選択すると、置き換えタスクセットがプロビジョニングされた後、デプロイによってトラフィックが自動的に再ルーティングされます。

    [トラフィックを再ルーティングするタイミングを指定する] を選択すると、置き換えタスクセットが正常にプロビジョニングされてから待機する日数、時間、分を選択します。この待機時間の間に、AppSpec ファイルで指定された Lambda 関数の検証テストが実行されます。トラフィックが再ルーティングされる前に待機時間が終了した場合、デプロイステータスは `Stopped` に変更されます。
15. 元のリビジョンの終了 では、デプロイが成功してからAmazon ECS サービスの元のタスクセットが終了するまで待機する日数、時間、分数を選択します。
16. (オプション) アドバンスト で、Amazon SNS 通知トリガー、Amazon CloudWatch アラーム、自動ロールバックなど、デプロイに含めたいオプションを設定します。

    詳細については、「デプロイグループの詳細オプションの設定」を参照してください。

AWS Lambda 関数デプロイ用のアプリケーションを作成する (コンソール)

CodeDeploy コンソールを使用して、 AWS Lambda 関数のデプロイ用のアプリケーションを作成できます。

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで デプロイ を展開して、ご利用開始にあたって を選択します。
3. [アプリケーションの作成] ページで、[CodeDeploy の使用] を選択します。
4. [アプリケーション名] にアプリケーションの名前を入力します。
5. [Compute platform (コンピューティングプラットフォーム)] で [AWS Lambda] を選択します。
6. [アプリケーションを作成] を選択します。
7. アプリケーションのページで、[デプロイグループ] タブの [デプロイグループの作成] を選択します。
8. [デプロイグループ名] に、デプロイグループを表す名前を入力します。

   > **Note**
   >
   > 他のデプロイグループで使用されているのと同じ設定 (デプロイグループ名、タグ、 グループ名、デプロイ設定など) を使用する場合は、このページでこれらの設定を選択します。この新しいデプロイグループと既存のデプロイグループは名前が同じでも、それぞれが別のアプリケーションに関連付けられるため、CodeDeploy では別のデプロイグループとして扱われます。

9. サービスロールで、CodeDeploy にアクセス権を付与するサービスロールを選択します AWS Lambda。詳細については、「ステップ 2: CodeDeployのサービスのロールを作成する」を参照してください。
10. 事前定義済みのデプロイ設定を使用する場合は、[デプロイ設定] から 1 つを選択し、ステップ 12 に進みます。カスタム設定を作成するには、次のステップに進みます。

    デプロイ設定の詳細については、AWS Lambda コンピューティングプラットフォームでのデプロイ設定 を参照してください。

11. カスタム設定を作成するには、[デプロイ設定の作成] を選択し、以下の操作を行います。

    a. [デプロイ設定名] に、設定の名前を入力します。

    b. [タイプ] で、設定タイプを選択します。[Canary] を選択すると、トラフィックは 2 回の増分で移行されます。[Linear] を選択すると、トラフィックは毎回同じ間隔 (分) の等しい増分で移行します。

    c. [Step] に、移行するトラフィックの割合を 1～99 で入力します。設定タイプが [Canary] の場合、この値は最初の増分で移行されるトラフィックの割合を示します。残りのトラフィックは、2 回目の増分で、選択した間隔後に移行されます。設定タイプが [Linear] の場合、この値は各間隔の開始時に移行されるトラフィックの割合を示します。

    d. [Interval (間隔)] に、時間 (分数) を入力します。設定タイプが [Canary] の場合、この値は最初と 2 回目のトラフィック移行の時間 (分) を示します。設定タイプが [Linear (リニア)] の場合、この値は各増分の移行間の時間 (分) を示します。

    > **Note**
    >
    > AWS Lambda デプロイの最大長は 2 日、つまり 2,880 分です。したがって、Canary 設定の [Interval] に指定された最大値は、2,800 分です。リニア設定の最大値は、[Step] の値によって異なります。たとえば、トラフィックのリニア移行のステップの割合が 25% の場合、トラフィック移行は 4 回です。最大間隔値は、2,880 を 4 で割るか、720 分になります。

    e. [Create deployment configuration (デプロイ設定の作成)] を選択します。

12. (オプション) アドバンストで、デプロイに含めるオプション (Amazon SNS 通知トリガー、Amazon CloudWatch アラーム、自動ロールバックなど) を設定します。

    詳細については、「デプロイグループの詳細オプションの設定」を参照してください。

13. デプロイグループの作成 を選択します。

アプリケーションの作成 (CLI)

を使用してアプリケーション AWS CLI を作成するには、create-application コマンドを呼び出し、アプリケーションを一意に表す名前を指定します。(アカウントでは AWS 、CodeDeploy アプリケーション名はリージョンごとに 1 回のみ使用できます。 アプリケーション名は異なるリージョンで再利用できます。)

を使用してアプリケーション AWS CLI を作成したら、次のステップとして、リビジョンをデプロイするインスタンスを指定するデプロイグループを作成します。手順については、「CodeDeploy でデプロイグループを作成する」を参照してください。

デプロイグループを作成したら、次にアプリケーションおよびデプロイグループにデプロイするリビジョンを準備します。手順については、「CodeDeploy のアプリケーションリビジョンの操作」を参照してください。

CodeDeploy を使用してアプリケーション詳細を表示する

CodeDeploy コンソール、 AWS CLI、または CodeDeploy APIs を使用して、 AWS アカウントに関連付けられているすべてのアプリケーションの詳細を表示できます。

トピック

- アプリケーションの詳細を表示する (コンソール)
- アプリケーションの詳細を表示する (CLI)

アプリケーションの詳細を表示する (コンソール)

CodeDeploy コンソールを使用してアプリケーションの詳細を表示するには

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで [デプロイ] を展開して [ご利用開始にあたって] を選択します。
3. 追加のアプリケーション詳細を表示するには、リストからアプリケーションの名前を選択します。

アプリケーションの詳細を表示する (CLI)

を使用してアプリケーションの詳細 AWS CLI を表示するには、 get-application コマンド、 batch-get-application コマンド、または list-applications コマンドを呼び出します。

単一のアプリケーションに関する詳細を表示するには、get-application コマンドを呼び出し、アプリケーション名を指定します。

複数のアプリケーションに関する詳細を表示するには、batch-get-applications コマンドを呼び出し、複数のアプリケーションの名前を指定します。

アプリケーション名のリストを表示するには、list-applications コマンドを呼び出します。

通知ルールの作成

通知ルールを使用すると、デプロイの成功や失敗など、デプロイアプリケーションに変更があった場合にユーザーに通知できます。通知ルールは、イベントと、通知の送信に使用される Amazon SNS トピックの両方を指定します。詳細については、「通知とは」を参照してください。

コンソールまたは を使用して AWS CLI 、通知ルールを作成できます AWS CodeDeploy。

通知ルールを作成するには (コンソール)

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy/ で CodeDeploy コンソールを開きます。
2. [Application (アプリケーション)] を選択し、通知を追加するアプリケーションを選択します。
3. アプリケーションページで、[Notify (通知)]、[Create notification rule (通知ルールの作成)] の順に選択します。アプリケーションの [Settings (設定)] ページに移動し、[Create notification rule (通知ルールの作成)] を選択することもできます。
4. [通知名] に、ルールの名前を入力します。
5. Amazon EventBridge に提供された情報のみを通知に含める場合は、[Detail type (詳細タイプ)] で [Basic (基本)] を選択します。Amazon EventBridge に提供される情報に加えて、CodeDeploy または通知マネージャから提供される場合がある情報も含める場合は、[Full (完全)] を選択します。

   詳細については、「通知の内容とセキュリティについて」を参照してください。
6. [Events that trigger notifications (通知をトリガーするイベント)] で、通知を送信するイベントを選択します。

Category	イベント
デプロイメント	失敗

Category	イベント
	成功 起動済み

7. [ターゲット] で、[SNS トピックの作成] を選択します。

> **Note**
>
> トピックを作成すると、トピックへのイベントの発行を CodeDeploy に許可するポリシーが適用されます。CodeDeploy 通知専用に作成されたトピックを使用すると、このデプロイアプリケーションに関する通知を表示するトピックのサブスクリプションリストにのみユーザーを追加することもできます。

[codestar-notifications-] プレフィックスの後にトピックの名前を入力し、[送信] を選択します。

> **Note**
>
> 新しいトピックを作成する代わりに既存の Amazon SNS トピックを使用する場合は、[Targets (ターゲット)] でその ARN を選択します。トピックに適切なアクセスポリシーが設定されていること、およびサブスクライバーリストにデプロイアプリケーションに関する情報の表示を許可されたユーザーのみが含まれていることを確認します。詳細については、「通知用の Amazon SNS トピックを設定する」および「通知の内容とセキュリティについて理解する」を参照してください。

8. ルールの作成を終了するには、[Submit (送信)] を選択します。

9. 通知を受け取るには、そのルールの Amazon SNS トピックにユーザーをサブスクライブする必要があります。詳細については、「ターゲットである Amazon SNS トピックへのユーザーのサブスクライブ」を参照してください。チャットアプリケーションで通知と Amazon Q Developer を統合して、通知を Amazon Chime チャットルームまたはスラックチャネルに送信することもできます。詳細については、「チャットアプリケーションでの通知と Amazon Q Developer の統合の設定」を参照してください。

通知ルールを作成するには (AWS CLI)

1. ターミナルまたはコマンドプロンプトで、create-notification rule コマンドを実行して、JSON スケルトンを生成します。

```
aws codestar-notifications create-notification-rule --generate-cli-skeleton
 > rule.json
```

ファイルには任意の名前を付けることができます。この例では、ファイルの名前を *rule.json* とします。

2. プレーンテキストエディタで JSON ファイルを開き、これを編集してルールに必要なリソース、イベントタイプ、Amazon SNS ターゲットを含めます。次の例は、ID *123456789012* **MyNotificationRule**の AWS アカウントの *MyDeploymentApplication* という名前のアプリケーションの という名前の通知ルールを示しています。デプロイが成功したとき、通知は、完全な詳細タイプで *codestar-##-MyNotificationTopic* という名で Amazon SNS トピックに送信されます：

```
{
    "Name": "MyNotificationRule",
    "EventTypeIds": [
        "codedeploy-application-deployment-succeeded"
    ],
    "Resource": "arn:aws:codebuild:us-east-2:123456789012:MyDeploymentApplication",
    "Targets": [
        {
            "TargetType": "SNS",
            "TargetAddress": "arn:aws:sns:us-east-2:123456789012:codestar-
notifications-MyNotificationTopic"
        }
    ],
    "Status": "ENABLED",
    "DetailType": "FULL"
}
```

ファイルを保存します。

3. 先ほど編集したファイルを使用して、ターミナルまたはコマンドラインで、create-notification-rule コマンドを再度実行し、通知ルールを作成します。

```
aws codestar-notifications create-notification-rule --cli-input-json
 file://rule.json
```

4. 成功すると、コマンドは次のような通知ルールの ARN を返します。

```
{
    "Arn": "arn:aws:codestar-notifications:us-east-1:123456789012:notificationrule/
dc82df7a-EXAMPLE"
}
```

CodeDeploy アプリケーション名の変更

AWS CLI または CodeDeploy APIs を使用して、アプリケーションの名前を変更できます。

アプリケーション名のリストを表示するには、 AWS CLI を使用して list-applications コマンドを呼び出します。

を使用してアプリケーション名 AWS CLI を変更する方法については、「update-application」を参照してください。

CodeDeploy API を使用してのアプリケーション名変更の詳細については、「API_UpdateApplication」を参照してください。

CodeDeploy でアプリケーションを削除する

CodeDeploy コンソール、 AWS CLI、または CodeDeploy API アクションを使用してアプリケーションを削除できます。CodeDeploy の API アクションの使用方法については、「DeleteApplication」を参照してください。

> **Warning**
>
> アプリケーションを削除すると、関連するすべてのデプロイグループの情報およびデプロイの詳細を含む、アプリケーションに関する情報が CodeDeploy システムから削除されます。EC2 オンプレミスのデプロイ用に作成されたアプリケーションを削除しても、インスタンスからアプリケーションのリビジョンが削除されたり、Amazon S3 バケットからリビジョンが削除されたりすることはありません。EC2 オンプレミスのデプロイ用に作成されたアプリケーションを削除しても、Amazon EC2 インスタンスが削除されたり、オンプレミスイン

スタンスが登録解除されたりすることは一切ありません。このアクションを元に戻すことはできません。

トピック

- アプリケーションの削除 (コンソール)
- アプリケーションの削除 (AWS CLI)

アプリケーションの削除 (コンソール)

CodeDeploy コンソールを使用してアプリケーションを削除するには以下を実行してください。

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで Deploy を展開し、Applications を選択します。
3. アプリケーションのリストで、削除するアプリケーションを選択します。

   アプリケーションに関する詳細を含むページが表示されます。
4. 右上にあるアプリケーションの削除を選択します。
5. プロンプトが表示されたら、**delete** を入力し、削除したいアプリケーションが正しいことを確認します。確認したら 削除 を選択します。

アプリケーションの削除 (AWS CLI)

を使用してアプリケーション AWS CLI を削除するには、delete-application コマンドを呼び出し、アプリケーション名を指定します。アプリケーション名のリストを表示するには、list-applications コマンドを呼び出します。

CodeDeploy でのデプロイグループの使用

CodeDeploy アプリケーション用に 1 つ以上のデプロイグループを指定できます。各アプリケーションのデプロイでは、そのデプロイグループの 1 つを使用します。デプロイグループには、デプロイ中に使用される設定と構成が含まれています。ほとんどのデプロイグループ設定は、アプリケーションで使用されるコンピューティングプラットフォームによって異なります。ロールバック、トリガー、アラームなどの一部の設定は、どのコンピューティングプラットフォーム用のデプロイグループでも設定できます。

Amazon ECS コンピューティングプラットフォームのデプロイでのデプロイグループ

Amazon ECS デプロイでは、デプロイグループは Amazon ECS サービス、ロードバランサー、オプションのテストリスナー、2 つのターゲットグループを指定します。また、代替タスクにトラフィックを再ルーティングするタイミングと、デプロイが成功した後で元のタスクセットと Amazon ECS アプリケーションを終了させるタイミングを設定します。

AWS Lambda コンピューティングプラットフォームデプロイのデプロイグループ

AWS Lambda デプロイでは、デプロイグループは、 AWS Lambda 関数の今後のデプロイ用に一連の CodeDeploy 設定を定義します。例えば、デプロイグループでは、新しいバージョンの Lambda 関数にトラフィックをルーティングする方法を指定します。また、アラームとロールバックを指定する場合もあります。 AWS Lambda デプロイグループ内の 1 つのデプロイは、1 つ以上のグループ設定を上書きできます。

EC2 オンプレミスコンピューティングプラットフォームのデプロイでのデプロイグループ

EC2/オンプレミス のデプロイでは、デプロイグループはデプロイをターゲットにした個別のインスタンスのセットです。デプロイグループには、個別にタグ付けされた Amazon EC2 インスタンス、Amazon EC2 Auto Scaling グループ内の Amazon EC2 インスタンス、またはその両方が含まれます。

インプレースデプロイでは、デプロイグループのインスタンスは最新のアプリケーションリビジョンで更新されます。

ブルー/グリーンデプロイでは、1 つ以上のロードバランサーから元のインスタンスを登録解除し、通常は最新のアプリケーションリビジョンが既にインストールされたインスタンスの代替セットを登録して、インスタンスの 1 つのセットから別のセットにトラフィックが転送されます。

複数のデプロイグループを CodeDeploy のアプリケーションに関連付けることができます。これにより、インスタンスの別々のセットに異なるタイミングでアプリケーションリビジョンをデプロイできます。例えば、1 つのデプロイグループを使用して、`Test` というタグが付けられた、コードの品質を確認するインスタンスのセットにアプリケーションリビジョンをデプロイできます。次に、追加の確認のため、`Staging` というタグが付けられたインスタンスがあるデプロイグループに、同じアプリケーションリビジョンをデプロイします。最後に、最新アプリケーションを顧客にリリースする準備ができたら、`Production` というタグが付けられたインスタンスを含むデプロイグループにデプロイします。

複数のタググループを使用して、デプロイグループに含めるインスタンスをさらに絞り込むこともできます。詳細については、「Tagging Instances for Deployments」を参照してください。

CodeDeploy コンソールを使用してアプリケーションを作成する場合は、最初のデプロイグループを同時に設定します。を使用してアプリケーション AWS CLI を作成する場合は、別のステップで最初のデプロイグループを作成します。

AWS アカウントに関連付けられているデプロイグループのリストを表示するには、「」を参照してくださいCodeDeploy を使用したデプロイグループの詳細の表示。

Amazon EC2 インスタンスタグの詳細については、「コンソールでのタグの処理」を参照してください。オンプレミスインスタンスの詳細については、「Working with On-Premises Instances」を参照してください。Amazon EC2 Auto Scaling の情報に関しては、「CodeDeploy と Amazon EC2 Auto Scaling の統合」を参照してください。

トピック

- the section called “デプロイグループの作成”
- the section called “デプロイグループの詳細の表示”
- the section called “デプロイグループの設定を変更する”
- the section called “デプロイグループの詳細オプションの設定”

- the section called “デプロイグループを削除する”

CodeDeploy でデプロイグループを作成する

CodeDeploy コンソール、 AWS CLI、CodeDeploy APIs、または CloudFormation テンプレートを使用してデプロイグループを作成できます。 CloudFormation テンプレートを使用してデプロイグループを作成する方法については、「」を参照してくださいCloudFormation CodeDeploy リファレンスの テンプレート。

CodeDeploy コンソールを使用してアプリケーションを作成する場合は、最初のデプロイグループを同時に設定します。を使用してアプリケーション AWS CLI を作成する場合は、別のステップで最初のデプロイグループを作成します。

デプロイグループ作成の一環として、サービスロールを指定する必要があります。詳細については、「ステップ 2: CodeDeployのサービスのロールを作成する」を参照してください。

トピック

- インプレースデプロイ用のデプロイグループを作成する (コンソール)
- EC2/オンプレミス Blue/Green デプロイ用のデプロイグループを作成する (コンソール)
- Amazon ECS デプロイ用のデプロイグループを作成する (コンソール)
- CodeDeploy Amazon EC2 デプロイ用の Elastic Load Balancing でロードバランサーをセットアップする
- CodeDeploy Amazon ECS デプロイ用のロードバランサー、ターゲットグループ、リスナーをセットアップする
- デプロイグループの作成 (CLI)

インプレースデプロイ用のデプロイグループを作成する (コンソール)

CodeDeploy コンソールを使用して、インプレースデプロイ用のデプロイグループを作成する方法。

> **Warning**
>
> 次の場合は、これらの手順を実行しないでください。
>
> - アプリケーションの最初の CodeDeploy デプロイで使用するインスタンスを準備していない。インスタンスをセットアップするには、CodeDeploy のためにインスタンスを用いた操作 の指示に従い、その後にこのトピックの手順に従います。

- カスタムデプロイ設定を使用してデプロイグループを作成したいが、まだデプロイ設定を作成していない。Create a Deployment Configuration の指示に従った後に、このトピックの手順に従います。
- 少なくとも、「ステップ 2: CodeDeployのサービスのロールを作成する」に記載されている信頼とアクセス権限を持つ、CodeDeploy を信頼するサービスロールがない。サービスロールを作成して設定するには、ステップ 2: CodeDeployのサービスのロールを作成する の指示に従い、その後にこのトピックの手順に従います。
- インプレースデプロイのために Elastic Load Balancing で、Classic Load Balancer、Application Load Balancer、または Network Load Balancer を選択したいが、まだ作成していない場合。

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで [デプロイ] を展開し、[アプリケーション] を選択します。
3. [Applications] ページで、デプロイグループを作成するアプリケーションの名前を選択します。
4. アプリケーションのページで、[デプロイグループ] タブの [デプロイグループの作成] を選択します。
5. [デプロイグループ名] に、デプロイグループを表す名前を入力します。

   > **Note**
   >
   > 他のデプロイグループで使用されている設定 (デプロイグループ名、タグ、Amazon EC2 Auto Scaling グループ名、または両方、およびデプロイ設定を含む) を使用する場合は、このページでこれらの設定を指定します。この新しいデプロイグループと既存のデプロイグループは名前が同じでも、それぞれが別のアプリケーションに関連付けられるため、CodeDeploy では別のデプロイグループとして扱われます。

6. [サービスロール] で、ターゲットインスタンスへのアクセス権を CodeDeploy に付与するサービスロールを選択します。
7. [デプロイタイプ] で、[インプレース] を選択します。

8. [環境設定] で、次の操作を行います。

   a. アプリケーションを Amazon EC2 Auto Scaling グループにデプロイする場合は、[Amazon EC2 Auto Scaling グループ] を選択し、アプリケーションリビジョンをデプロイする先の Amazon EC2 Auto Scaling グループの名前を選択します。Amazon EC2 Auto Scaling グループの一部として新しい Amazon EC2 インスタンスが起動されると、CodeDeploy ではリビジョンを新しいインスタンスに自動的にデプロイできます。デプロイグループには最大 10 個の Amazon EC2 Auto Scaling グループを追加できます。詳細については、「CodeDeploy と Amazon EC2 Auto Scaling の統合」を参照してください。

   b. [Amazon EC2 Auto Scaling グループ] を選択した場合、オプションで [Auto Scaling グループに終了フックを追加] を選択すると、デプロイグループを作成または更新するときに CodeDeploy が Auto Scaling グループに終了フックをインストールします。このフックをインストールすると、CodeDeploy は終了デプロイを実行します。詳細については、「Auto Scaling スケールインイベント中の終了デプロイの有効化」を参照してください。

   c. インスタンスにタグを付ける場合は、[Amazon EC2 インスタンス] または [オンプレミスインスタンス] を選択します。[キー] フィールドと [値] フィールドに、インスタンスにタグを付けるために使用するキーと値のペアの値を入力します。単一タググループで最大 10 個のキーと値のペアをタグ付けできます。

      i. [値] フィールドでワイルドカードを使用して、似ている Amazon EC2 インスタンス、コストセンター、グループ名などの特定のパターンでタグ付けされているすべてのインスタンスを識別できます。例えば、キー フィールドに 名前 を選択し、値 フィールドに **GRP-*a** を入力すると、CodeDeploy は **GRP-1a**、**GRP-2a**、および **GRP-XYZ-a** などそのパターンに当てはまるすべてのインスタンスを特定します。

      ii. [値] フィールドでは、大文字と小文字が区別されます。

      iii. リストからキーと値のペアを削除するには、削除のアイコンを選択します。

      CodeDeploy で、指定されたキーバリューの各ペアまたは Amazon EC2 Auto Scaling グループ名に一致するインスタンスが検出されると、一致したインスタンスの数が表示されます。インスタンスに関する詳細情報を表示するには、数をクリックします。

      デプロイ先のインスタンスの基準を絞り込む場合は、タググループを追加 を選択してタググループを作成します。それぞれ最大 10 個のキーと値のペアを持つタググループを 3 つまで作成できます。デプロイグループで複数のタググループを使用する場合は、すべてのタググループによって識別されたインスタンスのみがデプロイグループに含まれます。つまり、

インスタンスがデプロイグループに含まれるには、各グループの少なくとも 1 つのタグが一致する必要があります。

タググループを使用してデプロイグループを絞り込む方法ついては、「Tagging Instances for Deployments」を参照してください。

9. [Systems Manager を使用したエージェント設定] で、デプロイグループのインスタンスに CodeDeploy エージェントをインストールおよび更新する方法を指定します。CodeDeploy エージェントの詳細については、「CodeDeploy エージェントの使用」を参照してください。Systems Manager の詳細については、「Systems Manager とは」を参照してください。

   a. Never: Systems Manager を使用する CodeDeploy インストールの設定をスキップします。デプロイで使用するには、インスタンスにエージェントがインストールされている必要があります。したがって、CodeDeploy エージェントを別の方法でインストールする場合のみ、このオプションを選択します。

   b. [1 回のみ]: Systems Manager は、デプロイグループ内のすべてのインスタンスに CodeDeploy エージェントを 1 回インストールします。

   c. [現在およびスケジュール更新]: Systems Manager は、設定したスケジュールに従って CodeDeploy エージェントをインストールするステートマネージャーとの関連付けを作成します。ステートマネージャーおよび関連付けの詳細については、「ステートマネージャーについて」を参照してください。

10. [デプロイ設定] で、インスタンスをデプロイするレート (一度に 1 つずつ、一度にすべて、など) を制御するデプロイ設定を選択します。デプロイ設定の詳細については、CodeDeploy でデプロイ設定を使用する を参照してください。

11. (オプション) [ロードバランサー] で [ロードバランシングを有効にする] を選択し、一覧から、CodeDeploy デプロイ中のインスタンスへのトラフィックを管理する Classic Load Balancer、Application Load Balancer のターゲットグループ、Network Load Balancer のターゲットグループを選択します。最大 10 個の Classic Load Balancer と 10 個のターゲットグループとで、合計 20 個のアイテムを選択できます。デプロイする Amazon EC2 インスタンスが、選択したロードバランサー (Classic Load Balancer) またはターゲットグループ (Application Load Balancer および Network Load Balancer) に登録されていることを確認します。

    デプロイ中、元のインスタンスは選択したロードバランサーとターゲットグループから登録解除され、デプロイ中にトラフィックがこれらのインスタンスにルーティングされないようにします。デプロイが完了すると、各インスタンスは選択したすべての Classic Load Balancer とターゲットグループに再登録されます。

CodeDeploy デプロイ用のロードバランサーの詳細については、「Integrating CodeDeploy with Elastic Load Balancing」を参照してください。

> **Warning**
>
> このデプロイグループで Auto Scaling グループと Elastic Load Balancing ロードバランサーの両方を設定し、Auto Scaling グループにロードバランサーをアタッチする場合は、このデプロイグループから CodeDeploy デプロイを作成する前にこのアタッチメントを完了することをお勧めします。デプロイを作成した後にアタッチメントを完了しようとすると、すべてのインスタンスがロードバランサーから予期せず登録解除される可能性があります。

12. (オプション) [アドバンスト] を展開し、デプロイに含めるオプション (Amazon SNS 通知トリガー、Amazon CloudWatch アラーム、Auto Scaling オプション、自動ロールバックなど) を設定します。

    詳細については、「デプロイグループの詳細オプションの設定」を参照してください。

13. デプロイグループの作成 を選択します。

EC2/オンプレミス Blue/Green デプロイ用のデプロイグループを作成する (コンソール)

CodeDeploy コンソールを使用して、Blue/Green デプロイ用のデプロイグループを作成する方法。

> **Warning**
>
> 次の場合は、これらの手順を実行しないでください。
>
> - Blue/Green デプロイプロセス中に置き換えたい CodeDeploy エージェントがインストールされたインスタンスはありません。インスタンスをセットアップするには、CodeDeploy のためにインスタンスを用いた操作 の指示に従い、その後にこのトピックの手順に従います。
> - カスタムデプロイ設定を使用するアプリケーションを作成する必要があり、まだデプロイ設定を作成していません。Create a Deployment Configuration の指示に従った後に、このトピックの手順に従います。

- 少なくとも、「ステップ 2: CodeDeployのサービスのロールを作成する」に記載されている信頼とアクセス権限を持つ、CodeDeploy を信頼するサービスロールがない。サービスロールを作成して設定するには、ステップ 2: CodeDeployのサービスのロールを作成する の指示に従い、その後にこのトピックの手順に従います。
- 置き換え先環境でインスタンスを登録するために、Elastic Load Balancing で Classic Load Balancer または Application Load Balancer を作成していません。詳細については、「CodeDeploy Amazon EC2 デプロイ用の Elastic Load Balancing でロードバランサーをセットアップする」を参照してください。

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで [デプロイ] を展開し、[アプリケーション] を選択します。
3. [Applications] ページで、デプロイグループを作成するアプリケーションの名前を選択します。
4. アプリケーションのページで、[デプロイグループ] タブの [デプロイグループの作成] を選択します。
5. [デプロイグループ名] に、デプロイグループを表す名前を入力します。

   > **Note**
   >
   > 他のデプロイグループで使用されているのと同じ設定 (デプロイグループ名、タグ、Amazon EC2 Auto Scaling グループ名、デプロイ設定など) を使用する場合は、このページでこれらの設定を選択します。この新しいデプロイグループと既存のデプロイグループは名前が同じでも、これらは別のアプリケーションに関連付けられるため、CodeDeploy では別のデプロイグループとして扱われます。

6. [サービスロール] で、ターゲットインスタンスへのアクセス権を CodeDeploy に付与するサービスロールを選択します。
7. [デプロイタイプ] で [Blue/Green] を選択します。
8. [環境設定] で、次の操作を行います。

- 置き換え先環境にインスタンスを提供するために使用する方法を選択します。次のオプションがあります。
  - Amazon EC2 Auto Scaling グループを自動的にコピーする: CodeDeploy は、ユーザーが指定したグループをコピーして Amazon EC2 Auto Scaling グループを作成します。
  - [Manually provision instances]: デプロイを作成するまで置き換え先環境のインスタンスを特定しません。デプロイを開始する前に、インスタンスを作成する必要があります。代わりに、ここで置換するインスタンスを指定します。
- [Amazon EC2 Auto Scaling グループを自動的にコピー] を選択した場合、オプションで [Auto Scaling グループに終了フックを追加] を選択すると、デプロイグループを作成または更新するときに CodeDeploy が Auto Scaling グループに終了フックをインストールします。このフックをインストールすると、CodeDeploy は終了デプロイを実行します。詳細については、「Auto Scaling スケールインイベント中の終了デプロイの有効化」を参照してください。

9. [Systems Manager を使用したエージェント設定] で、デプロイグループのインスタンスに CodeDeploy エージェントをインストールおよび更新する方法を指定します。CodeDeploy エージェントの詳細については、「CodeDeploy エージェントの使用」を参照してください。Systems Manager の詳細については、「Systems Manager とは」を参照してください。

   a. Never: Systems Manager を使用する CodeDeploy インストールの設定をスキップします。デプロイで使用するには、インスタンスにエージェントがインストールされている必要があります。したがって、CodeDeploy エージェントを別の方法でインストールする場合のみ、このオプションを選択します。

   b. [1 回のみ]: Systems Manager は、デプロイグループ内のすべてのインスタンスに CodeDeploy エージェントを 1 回インストールします。

   c. [現在およびスケジュール更新]: Systems Manager は、設定したスケジュールに従って CodeDeploy エージェントをインストールするステートマネージャーとの関連付けを作成します。ステートマネージャーおよび関連付けの詳細については、「ステートマネージャーについて」を参照してください。

10. ステップ 8 での選択内容に応じて、次のいずれかを実行します:

    - [Amazon EC2 Auto Scaling グループを自動コピーする] を選択している場合: [Amazon EC2 Auto Scaling グループ] で、置き換え先環境のインスタンス用に作成される Amazon EC2 Auto Scaling グループのテンプレートとして使用したい Amazon EC2 Auto Scaling グループの名前を選択または入力します。選択した Amazon EC2 Auto Scaling グループ内の現在正常なインスタンスの数が、置き換え先環境で作成されます。

- インスタンスの手動プロビジョニングを選択した場合: Amazon EC2 Auto Scaling グループ、Amazon EC2 Auto Scaling インスタンス、またはその両方を選択して、このデプロイグループに追加するインスタンスを指定します。元の環境のインスタンス (つまり、置換対象のインスタンスまたは現在のアプリケーションリビジョンを実行しているインスタンス) を識別するための Amazon EC2 Auto Scaling タグ値または Amazon EC2 Auto Scaling グループ名を入力します。

11. [ロードバランサー] で [ロードバランシングを有効にする] を選択し、一覧から、代わりの Amazon EC2 インスタンスを登録する Classic Load Balancer、Application Load Balancer のターゲットグループ、Network Load Balancer のターゲットグループを選択します。各代替インスタンスは、選択したすべての Classic Load Balancer とターゲットグループに登録されます。最大 10 個の Classic Load Balancer と 10 個のターゲットグループとで、合計 20 個のアイテムを選択できます。

    トラフィックは、選択した [トラフィック再ルーティング] と [デプロイ設定] に従って、元のインスタンスから代替インスタンスに再ルーティングされます。

    CodeDeploy デプロイ用のロードバランサーの詳細については、「Integrating CodeDeploy with Elastic Load Balancing」を参照してください。

    > ⚠ Warning
    >
    > このデプロイグループで Auto Scaling グループと Elastic Load Balancing ロードバランサーの両方を設定していて、ロードバランサーを Auto Scaling グループにアタッチする場合は、このデプロイグループから CodeDeploy デプロイを作成する前にこのアタッチメントを完了することをお勧めします。デプロイを作成した後にアタッチメントを完了しようとすると、すべてのインスタンスがロードバランサーから予期せず登録解除される可能性があります。

12. [Deployment settings] で、置き換え先環境へトラフィックを再ルーティングするためのデフォルトのオプション、デプロイに使用するデプロイ設定、デプロイ後に元の環境のインスタンスを処理する方法を確認します。

    設定を変更する場合は、次のステップに進みます。それ以外の場合は、ステップ 14 に進みます。

13. Blue/Green デプロイのデプロイ設定を変更するには、以下のいずれかの設定を選択します。

設定	オプション
[Traffic rerouting]	• [トラフィックをすぐに再ルーティングする]: 置き換え先環境のインスタンスがプロビジョニングされ、最新のアプリケーションリビジョンがインストールされるとすぐに、指定のロードバランサーとターゲットグループに自動的に登録され、トラフィックがそれらに再ルーティングされます。元の環境内のインスタンスは、登録解除されます。 • [トラフィックを再ルーティングするかどうかを選択します]: 置き換え先環境のインスタンスは、手動でトラフィックを再ルーティングしないかぎり、指定のロードバランサーとターゲットグループに登録されません。指定した待機時間中にトラフィックが再ルーティングされない場合、デプロイステータスは停止に変更されます。
Deployment configuration	置き換え先環境のインスタンスがロードバランサーとターゲットグループに登録されているレート (個別、一括など) を選択します。 **Note** トラフィックが置き換え先環境に適切にルーティングされた後、元の環境のインスタンスは、どのデプロイ設定が選択されていても一度にすべて登録解除されます。 詳細については、「CodeDeploy でデプロイ設定を使用する」を参照してください。

設定	オプション
Original instances	• [デプロイグループの元のインスタンスを終了する]: トラフィックが置き換え先環境に再ルーティングされた後、ロードバランサーとターゲットグループから登録解除されたインスタンスは、指定した待機時間の後に終了します。次の点に注意してください。 • Auto Scaling グループで Deletion Protection が有効になっている場合 (`prevent-force-deletion` または `prevent-all-deletion`)、CodeDeploy ブルー/グリーンデプロイは、デプロイのクリーンアップ中に削除する前に、元の (ブルー) Auto Scaling グループでこの保護を無効にします。代替 (緑) Auto Scaling グループは、元のグループと同じ削除保護設定を維持します。 • [デプロイグループの元のインスタンスを実行し続ける]: トラフィックが置き換え先環境に再ルーティングされた後、ロードバランサーとターゲットグループから登録解除されたインスタンスは実行されたままになります。

14. (オプション) [アドバンスト] で、デプロイに含めるオプション (Amazon SNS 通知トリガー、Amazon CloudWatch アラーム、Auto Scaling オプション、自動ロールバックなど) を設定します。

    デプロイグループの詳細なオプションを指定する方法の詳細については、「デプロイグループの詳細オプションの設定」を参照してください。

15. デプロイグループの作成 を選択します。

Amazon ECS デプロイ用のデプロイグループを作成する (コンソール)

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで [デプロイ] を展開し、[アプリケーション] を選択します。
3. [Applications table (アプリケーションテーブル)] から、編集するデプロイグループに関連付けられているアプリケーションの名前を選択します。
4. アプリケーションのページの [デプロイグループ] で、編集するデプロイグループの名前を選択します。
5. アプリケーションのページで、[デプロイグループ] タブの [デプロイグループの作成] を選択します。Amazon ECS デプロイのデプロイグループを作成するために必要なものの詳細については、「Amazon ECS デプロイを開始する前に」を参照してください。
6. [デプロイグループ名] に、デプロイグループを表す名前を入力します。

   > **Note**
   >
   > 他のデプロイグループで使用されているのと同じ設定 (デプロイグループ名、タグ、 グループ名、デプロイ設定など) を使用する場合は、このページでこれらの設定を選択します。この新しいグループと既存のグループは名前が同じでも、それぞれが別のアプリケーションに関連付けられるため、CodeDeploy では別のデプロイグループとして扱われます。

7. サービスロール で、CodeDeploy に Amazon ECS へのアクセスを許可するサービスロールを選択します。詳細については、「ステップ 2: CodeDeployのサービスのロールを作成する」を参照してください。
8. ロードバランサーの名前 から、Amazon ECS サービスにトラフィックを提供するロードバランサーの名前を選択します。
9. [本稼働リスナーポート] から、Amazon ECS サービスへの本稼働トラフィックを提供するリスナーのポートとプロトコルを選択します。
10. (オプション) テストリスナーポート から、デプロイ時に Amazon ECS サービス内の置き換えタスクセットにトラフィックを処理するテストリスナーのポートとプロトコルを選択しま

す。AfterAllowTestTraffic フック中に実行する Lambda 関数を、AppSpec ファイル内に一つ以上指定できます。この関数は、検証テストを実行できます。検証テストが失敗すると、デプロイのロールバックが発生します。検証テストに成功すると、デプロイのライフサイクルの次のフック BeforeAllowTraffic がトリガーされます。テストリスナーポートが指定されていない場合、AfterAllowTestTraffic フック中は何も起こりません。詳細については、「Amazon ECS のデプロイ向けの AppSpec の「hooks」セクション」を参照してください。

11. [Target group 1 name] および [Target group 2 name] から、デプロイ時にトラフィックをルーティングするターゲットグループを選択します。CodeDeploy は、1 つのターゲットグループを Amazon ECS サービスの元のタスクセットにバインドし、もう一方をその置き換えタスクセットにバインドします。詳細については、「Application Load Balancer のターゲットグループ」を参照してください。
12. [トラフィックをすぐに再ルーティングする] または [トラフィックを再ルーティングするタイミングを指定する] を選択し、更新された Amazon ECS サービスにトラフィックを再ルーティングするタイミングを決定します。

    [トラフィックをすぐに再ルーティングする] を選択すると、置き換えタスクセットがプロビジョニングされた後、デプロイによってトラフィックが自動的に再ルーティングされます。

    [トラフィックを再ルーティングするタイミングを指定する] を選択すると、置き換えタスクセットが正常にプロビジョニングされてから待機する日数、時間、分を選択します。この待機時間の間に、AppSpec ファイルで指定された Lambda 関数の検証テストが実行されます。トラフィックが再ルーティングされる前に待機時間が終了した場合、デプロイステータスは Stopped に変更されます。
13. 元のリビジョンの終了 では、デプロイが成功してからAmazon ECS サービスの元のタスクセットが終了するまで待機する日数、時間、分数を選択します。
14. (オプション) アドバンスト で、Amazon SNS 通知トリガー、Amazon CloudWatch アラーム、自動ロールバックなど、デプロイに含めたいオプションを設定します。

    詳細については、「デプロイグループの詳細オプションの設定」を参照してください。

CodeDeploy Amazon EC2 デプロイ用の Elastic Load Balancing でロードバランサーをセットアップする

ブルー/グリーンデプロイまたはデプロイグループでオプションのロードバランサーを指定するインプレースデプロイを実行する前に、事前に Elastic Load Balancing で少なくとも 1 つの Classic Load Balancer、Application Load Balancer、Network Load Balancer を作成しておく必要がありま

す。Blue/Green デプロイの場合は、そのロードバランサーを使用して置き換え先環境を構成するインスタンスを登録します。元の環境のインスタンスは、この同じロードバランサーにオプションで登録できます。インプレースデプロイでは、ロードバランサーを使用して CodeDeploy で処理中のインスタンスを登録解除し、作業が完了したら再登録します。

CodeDeploy は、複数のロードバランサーの背後にある Amazon EC2 インスタンスへのブルー/グリーンデプロイとインプレースデプロイをサポートします。例えば、200 個の Amazon EC2 インスタンスがあり、そのうちの 100 個が 2 つの Classic Load Balancer に登録され、さらに 100 個が 2 つの Application Load Balancer の 4 つのターゲットグループに登録されているとします。このシナリオでは、CodeDeploy を使用すると、2 つの Classic Load Balancer、2 つの Application Load Balancer、および 4 つのターゲットグループに分散している場合でも、200 個のインスタンスすべてにブルー/グリーンデプロイとインプレースデプロイを行うことができます。

CodeDeploy は、最大 10 個の Classic Load Balancer と 10 個のターゲットグループ、合計 20 個のアイテムをサポートします。

1 つ以上の Classic Load Balancer を設定するには、Classic Load Balancer のユーザーガイドにある「チュートリアル: Classic Load Balancer の作成」の手順に従ってください。次の点に注意してください。

- ステップ 2: ロードバランサーの定義、[Create LB Inside] で、インスタンスを作成したときに選択したのと同じ VPC を選択します。
- ステップ 5: ロードバランサーへの EC2 インスタンスの登録で、現在デプロイグループにあるインスタンス (インプレースデプロイ)、または元の環境に存在するように指定したインスタンス (Blue/Green デプロイ) を選択します。
- ステップ 7: Load Balancer の作成と検証で、ロードバランサーの DNS アドレスをメモします。

  例えば、ロードバランサーの名前を `my-load-balancer` とした場合、DNS アドレスは `my-load-balancer-1234567890.us-east-2.elb.amazonaws.com` のような形式で表示されます。

1 つ以上の Application Load Balancer を設定するには、以下のトピックのいずれかの指示に従ってください。

- Application Load Balancer の作成
- チュートリアル: を使用して Application Load Balancer を作成する AWS CLI

1 つ以上の Network Load Balancer を設定するには、以下のトピックのいずれかの指示に従ってください。

- Network Load Balancer を作成する
- チュートリアル: を使用して Network Load Balancer を作成する AWS CLI

CodeDeploy Amazon ECS デプロイ用のロードバランサー、ターゲットグループ、リスナーをセットアップする

Amazon ECS コンピューティングプラットフォームを使用してデプロイを実行する前に、Application Load Balancer または Network Load Balancer、2 つのターゲットグループ、および 1 つまたは 2 つのリスナーを作成する必要があります。このトピックでは、Application Load Balancer を作成する方法を説明します。詳細については、「Amazon ECS デプロイを開始する前に」を参照してください。

ターゲットグループの 1 つが、Amazon ECS アプリケーションの元のタスクセットにトラフィックを誘導します。もう 1 つのターゲットグループは、置き換えタスクセットにトラフィックを送信します。デプロイ中、CodeDeploy は置き換えタスクセットを作成し、元のタスクセットから新しいタスクセットにトラフィックを再ルーティングします。CodeDeploy は、各タスクセットに使用されるターゲットグループを決定します。

ロードバランサーは、リスナーを使用してターゲットグループにトラフィックをルーティングします。本稼働リスナーが 1 つ必要です。検証テストの実行中、置き換えタスクセットにトラフィックをルーティングする、オプションのテストリスナーを指定できます。

ロードバランサーでは、2 つのパブリックサブネットを別々のアベイラビリティーゾーンに持つ VPC を使用する必要があります。以下のステップでは、デフォルト VPC を確認し、Amazon EC2 Application Load Balancer を作成し、ロードバランサーに 2 つのターゲットグループを作成する方法を示します。詳細については、「Network Load Balancer のターゲットグループ」を参照してください。

デフォルト VPC、パブリックサブネット、およびセキュリティグループの確認

このトピックでは、Amazon ECS デプロイ中に使用できる Amazon EC2 Application Load Balancer、2 つのターゲットグループ、および 2 つのポートを作成する方法について説明します。ポートの 1 つはオプションであり、デプロイ中に検証テスト用のテストポートにトラフィックをルーティングする場合にのみ必要です。

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/vpc/ で Amazon VPC コンソールを開きます。
2. 使用するデフォルト VPC を確認します。ナビゲーションペインで、[Your VPCs (お使いの VPC)] を選択します。[デフォルト VPC] 列に [はい] と表示されているVPCに注意してください。これがデフォルトの VPC になります。これには、使用するデフォルトのサブネットが含まれます。
3. [サブネット] を選択します。[デフォルトサブネット] 列で [はい] と表示されている 2 つのサブネットのサブネット ID をメモしておきます。これらの ID は、ロードバランサーを作成するときに使用します。
4. 各サブネットを選択し、[Description (説明)] タブを選択します。使用するサブネットが、異なるアベイラビリティーゾーンにあることを確認します。
5. サブネットを選択後、[Route Table] タブを選択します。使用する各サブネットがパブリックサブネットであることを確認するには、インターネットゲートウェイへのリンクのある行がルートテーブルに含まれていることを確認します。
6. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/ec2/ で Amazon EC2 コンソールを開きます。
7. ナビゲーションペインで、[Security Groups] を選択します。
8. 使用するセキュリティグループが使用可能であることを確認し、そのグループ ID ([sg-abcd1234] など) を書き留めます。これは、ロードバランサーを作成するときに使用します。

Amazon EC2 Application Load Balancer、2 つのターゲットグループ 、およびリスナーを作成します (コンソール)

Amazon EC2 コンソールを使用して Amazon EC2 Application Load Balancer を作成するには:

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/ec2/ で Amazon EC2 コンソールを開きます。
2. ナビゲーションペインで、[ロードバランサー] を選択します。
3. [Create Load Balancer] を選択します。
4. [Application Load Balancer] を選択し、[Create] を選択します。
5. [Name] に、ロードバランサーの名前を入力します。
6. [Scheme] で、[インターネット向け] を選択します。
7. [IP address type] で、[ipv4] を選択します。

8. （オプション）ロードバランサーの 2 番目のリスナーポートを設定します。このポートに提供されるテストトラフィックを使用して、デプロイの検証テストを実行できます。

   a. [Load Balancer Protocol (ロードバランサーのプロトコル)] で、[Add listener (リスナーの追加)] を選択します。

   b. 2 番目のリスナーの [Load Balancer Protocol] で、[HTTP] を選択します。

   c. [Load Balancer Port (ロードバランサーポート)] に [**8080**] を入力します。

9. アベイラビリティーゾーン の VPCで、デフォルトの VPC を選択し、使用する 2 つのデフォルトサブネットを選択します。

10. [Next: Configure Security Settings] を選択します。

11. [Next: Configure Security Groups] を選択します。

12. [Select an existing security group (既存のセキュリティグループを選択する)] を選択し、デフォルトのセキュリティグループを選択して、その ID を書き留めます。

13. [Next: Configure Routing] を選択します。

14. [Target group (ターゲットグループ)] で、[New target group (新しいターゲットグループ)] を選択し、最初のターゲットグループを設定します。

   a. [Name] に、ターゲットグループの名前 (例: **target-group-1**) を入力します。

   b. [Target type] で、[IP] を選択します。

   c. [Protocol] で、[HTTP] を選択します。[Port] に「**80**」と入力します。

   d. [Next: Register Targets] を選択します。

15. [Next: Review]、[Create] の順に選択します。

ロードバランサーの 2 番目のターゲットグループを作成するには

1. ロードバランサーがプロビジョニングされたら、Amazon EC2 コンソールを開きます。ナビゲーションペインで、[ターゲットグループ] を選択します。

2. [ターゲットグループの作成] を選択します。

3. [Name] に、ターゲットグループの名前 (例: **target-group-2**) を入力します。

4. [Target type] で、[IP] を選択します。

5. [Protocol] で、[HTTP] を選択します。[Port] に「**80**」と入力します。

6. [VPC] で、デフォルトの VPC を選択します。

7. [作成] を選択します。

> **Note**
>
> Amazon ECS デプロイを実行するには、ロードバランサー用に 2 つのターゲットグループを作成する必要があります。Amazon ECS サービスを作成するときに、いずれかのターゲットグループの ARN を使用します。詳細については、Amazon ECS ユーザーガイド の「ステップ 4: Amazon ECS サービスを作成する」を参照してください。

Amazon EC2 Application Load Balancer、2 つのターゲットグループ、およびリスナーを作成します (CLI)

AWS CLIを使用して Application Load Balancer を作成するには

1. create-load-balancer コマンドを使用して、Application Load Balancer を作成します。異なるアベイラビリティーゾーンにある 2 つのサブネット、およびセキュリティグループを指定します。

```
aws elbv2 create-load-balancer --name bluegreen-alb \
--subnets subnet-abcd1234 subnet-abcd5678 --security-groups sg-abcd1234 --
region us-east-1
```

出力には、次の形式でロードバランサーの Amazon リソースネーム (ARN) が含まれます。

```
arn:aws:elasticloadbalancing:region:aws_account_id:loadbalancer/app/bluegreen-alb/
e5ba62739c16e642
```

2. 最初のターゲットグループを作成するには、create-target-group コマンドを使用します。CodeDeploy は、このターゲットグループのトラフィックを、サービスの元のタスクセットまたは置き換えタスクセットにルーティングします。

```
aws elbv2 create-target-group --name bluegreentarget1 --protocol HTTP --port 80 \
--target-type ip --vpc-id vpc-abcd1234 --region us-east-1
```

出力には、以下の形式で最初のターゲットグループの ARN が含まれます。

```
arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup/
bluegreentarget1/209a844cd01825a4
```

3. 2 つ目のターゲットグループを作成するには、create-target-group コマンドを使用します。CodeDeploy は、最初のターゲットグループによって処理されないタスクセットにターゲットグループのトラフィックをルーティングします。たとえば、最初のターゲットグループが元のタスクセットにトラフィックをルーティングする場合、このターゲットグループは置き換えタスクセットにトラフィックをルーティングします。

```
aws elbv2 create-target-group --name bluegreentarget2 --protocol HTTP --port 80 \
--target-type ip --vpc-id vpc-abcd1234 --region us-east-1
```

出力には、以下の形式で 2 番目のターゲットグループの ARN が含まれます。

```
arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup/
bluegreentarget2/209a844cd01825a4
```

4. create-listener コマンドを使用して、本稼働トラフィックをポート 80 に転送するデフォルトルールを持つリスナーを作成します。

```
aws elbv2 create-listener --load-balancer-arn
 arn:aws:elasticloadbalancing:region:aws_account_id:loadbalancer/app/bluegreen-alb/
e5ba62739c16e642 \
--protocol HTTP --port 80 \
--default-actions
 Type=forward,TargetGroupArn=arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup
bluegreentarget1/209a844cd01825a4 --region us-east-1
```

出力には、以下の形式でリスナーの ARN が含まれます。

```
arn:aws:elasticloadbalancing:region:aws_account_id:listener/app/bluegreen-alb/
e5ba62739c16e642/665750bec1b03bd4
```

5. （オプション）create-listener コマンドを使用して、テストトラフィックをポート 8080 に転送するデフォルトルールを持つ 2 番目のリスナーを作成します。このポートで提供されるテストトラフィックを使用して、デプロイの検証テストを実行できます。

```
aws elbv2 create-listener --load-balancer-arn
 arn:aws:elasticloadbalancing:region:aws_account_id:loadbalancer/app/bluegreen-alb/
e5ba62739c16e642 \
--protocol HTTP --port 8080 \
```

```
--default-actions
 Type=forward,TargetGroupArn=arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup
bluegreentarget2/209a844cd01825a4 --region us-east-1
```

出力には、以下の形式でリスナーの ARN が含まれます。

```
arn:aws:elasticloadbalancing:region:aws_account_id:listener/app/bluegreen-alb/
e5ba62739c16e642/665750bec1b03bd4
```

デプロイグループの作成 (CLI)

を使用してデプロイグループ AWS CLI を作成するには、create-deployment-group コマンドを呼び出し、以下を指定します。

- アプリケーション名。アプリケーション名のリストを表示するには、[list-applications] コマンドを呼び出します。
- デプロイグループの名前。指定したアプリケーションに対して、この名前でデプロイグループが作成されます。デプロイグループは、1 つのアプリケーションにのみ関連付けることができます。
- デプロイグループに含めるインスタンスを識別するタグ、タググループ、または Amazon EC2 Auto Scaling グループ名に関する情報。
- CodeDeploy が他の AWS サービスとやり取りするときに AWS アカウントに代わって動作できるようにするサービスロールの Amazon リソースネーム (ARN) 識別子。サービスロール ARN を取得するには、「サービスロール ARN の取得 (CLI)」を参照してください。サービスロールの詳細については、IAM ユーザーガイド の「ロールに関する用語と概念」を参照してください。
- デプロイグループに関連付けるデプロイのタイプ (インプレースまたは Blue/Green) についての情報。
- (オプション) 既存のデプロイ設定の名前。デプロイ設定のリストを表示するには、View Deployment Configuration Details を参照してください。指定されない場合、CodeDeploy は、デフォルトのデプロイ設定を使用します。
- (オプション) Amazon Simple Notification Service トピックに登録しているユーザーにデプロイとインスタンスのイベントに関する通知をプッシュするトリガーを作成するコマンド。詳細については、「Monitoring Deployments with Amazon SNS Event Notifications」を参照してください。
- (オプション) アラームで指定したメトリクスが定義済みのしきい値を下回る/上回るとアクティブ化される、既存の CloudWatch アラームをデプロイグループに追加するコマンド。

- (オプション) デプロイが失敗した時、または CloudWatch アラームが作動した時に、既知の正常なリビジョンのデプロイにロールバックするコマンド。
- (オプション) Auto Scaling スケールインイベント中にライフサイクルイベントフックを生成するためのデプロイ用のコマンド。詳細については、「Amazon EC2 Auto Scaling と CodeDeploy の連携」を参照してください。
- インプレースデプロイの場合:
  - (オプション) デプロイプロセスでインスタンスへのトラフィックを管理する、Elastic Load Balancing の Classic Load Balancer、Application Load Balancer、または Network Load Balancer の名前。
- Blue/Green デプロイの場合。
  - Blue/Green デプロイプロセスの設定。
    - 置き換え先環境の新しいインスタンスをプロビジョニングする方法。
    - トラフィックを置き換え先環境にすぐに再ルーティングするか、またはトラフィックを手動で再ルーティングするために指定された期間待機するか。
    - 元の環境内のインスタンスを終了するかどうか。
  - 置き換え先環境で登録されたインスタンスに使用する Elastic Load Balancing の Classic Load Balancer、Application Load Balancer または Network Load Balancer の名前。

> **Warning**
>
> デプロイグループで Auto Scaling グループと Elastic Load Balancing ロードバランサーの両方を設定していて、ロードバランサーを Auto Scaling グループにアタッチする場合は、このデプロイグループから CodeDeploy デプロイを作成する前にこのアタッチメントを完了することをお勧めします。デプロイを作成した後にアタッチメントを完了しようとすると、すべてのインスタンスがロードバランサーから予期せず登録解除される可能性があります。

CodeDeploy を使用したデプロイグループの詳細の表示

CodeDeploy コンソール、 AWS CLI、または CodeDeploy APIs を使用して、アプリケーションに関連付けられたすべてのデプロイグループの詳細を表示できます。

トピック

- デプロイグループの詳細の表示 (コンソール)

- デプロイグループの詳細の表示 (CLI)

デプロイグループの詳細の表示 (コンソール)

CodeDeploy コンソールを使用してデプロイグループの詳細を表示するには:

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

> **Note**
>
> 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで [デプロイ] を展開し、[アプリケーション] を選択します。
3. [アプリケーション] ページで、デプロイグループに関連付けられたアプリケーション名を選択します。

> **Note**
>
> エントリが表示されない場合は、正しいリージョンが選択されていることを確認します。ナビゲーションバーのリージョンセレクターで、「AWS 全般のリファレンス」の「リージョンとエンドポイント」に一覧表示されているいずれかのリージョンを選択します。CodeDeploy は、これらのリージョンでのみサポートされています。

4. 個別のデプロイグループに関する詳細を表示するには、[デプロイグループ] タブで、デプロイグループの名前を選択します。

デプロイグループの詳細の表示 (CLI)

を使用してデプロイグループの詳細 AWS CLI を表示するには、 `get-deployment-group` コマンドまたは `list-deployment-groups` コマンドを呼び出します。

1 つのデプロイの詳細を表示するには、コマンド デプロイグループ入手 を呼び出して、以下を指定します。

- デプロイグループに関連付けられたアプリケーション名。アプリケーション名を取得するには、[リストアプリケーション] コマンドを呼び出します。

- デプロイグループ名。デプロイグループ名を取得するには、[リストデプロイグループ] コマンドを呼び出します。

デプロイグループ名のリストを表示するには、[リストデプロイグループ] コマンドを呼び出して、デプロイグループに関連付けられたアプリケーション名を指定します。アプリケーション名を取得するには、リストアプリケーション コマンドを呼び出します。

CodeDeploy を使用して、デプロイグループの設定を変更します。

CodeDeploy コンソール、 AWS CLI、または CodeDeploy APIs を使用して、デプロイグループの設定を変更できます。

> **Warning**
>
> デプロイグループが、まだ作成していないカスタムデプロイグループを使用する場合、これらの手順を使用しないでください。代わりに、「Create a Deployment Configuration」の手順に従って、このトピックに戻ります。デプロイグループが、別のまだ作成していないサービスロールを使用する場合、これらの手順を使用しないでください。サービスロールは、最低でも、「ステップ 2: CodeDeployのサービスのロールを作成する」で説明されているアクセス権限を持つ CodeDeploy を信頼する必要があります。正しいアクセス許可を持つサービスロールを作成し、設定するには、「ステップ 2: CodeDeployのサービスのロールを作成する」の手順に従って、このトピックに戻ります。

トピック

- デプロイグループの設定を変更する (コンソール)
- デプロイグループの設定を変更する (CLI)

デプロイグループの設定を変更する (コンソール)

CodeDeploy コンソールを使用してデプロイグループの設定を変更するには。

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

> **Note**
>
> 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで [デプロイ] を展開し、[アプリケーション] を選択します。
3. アプリケーションのリストで、変更するデプロイグループに関連付けられているアプリケーションの名前を選択します。

> **Note**
>
> エントリが表示されない場合、正しいリージョンが選択されていることを確認してください。ナビゲーションバーのリージョンセレクターで、「AWS 全般のリファレンス」の「リージョンとエンドポイント」に一覧表示されているいずれかのリージョンを選択します。CodeDeploy は、これらのリージョンでのみサポートされています。

4. [デプロイグループ] タブを選択し、変更するデプロイグループの名前を選択します。
5. デプロイグループページで、編集を選択します。
6. 必要に応じてデプロイグループのオプションを編集します。

   デプロイグループのコンポーネントの詳細については、「CodeDeploy でデプロイグループを作成する」を参照してください。
7. [Save changes] (変更の保存) をクリックします。

デプロイグループの設定を変更する (CLI)

を使用してデプロイグループの設定を変更するには、以下を指定 AWS CLI して update-deployment-group コマンドを呼び出します。

- EC2/オンプレミスおよび AWS Lambda デプロイの場合:
  - アプリケーション名。アプリケーション名のリストを表示するには、[list-applications] コマンドを呼び出します。
  - 現在のデプロイグループ名。デプロイグループ名のリストを表示するには、[list-deployment-groups] コマンドを呼び出します。
  - (オプション) 別のデプロイグループ名。

  - (オプション) 他の サービスとやり取りするときに CodeDeploy が AWS アカウントに代わって動作することを許可する AWS サービスロールに対応する別の Amazon リソースネーム (ARN)。サービスロール ARN を取得するには、「サービスロール ARN の取得 (CLI)」を参照してください。サービスロールの詳細については、IAM ユーザーガイド の「ロールに関する用語と概念」を参照してください。
  - (オプション) デプロイ設定の名前。デプロイ設定のリストを表示するには、View Deployment Configuration Details を参照してください。(指定されない場合、CodeDeploy は、デフォルトのデプロイ設定を使用します。)
  - (オプション) アラームで指定されたメトリクスが定義したしきい値を下回る、または、超える場合にアクティベートするデプロイグループに 1 つ以上の既存 CloudWatch アラームを追加するコマンド。
  - (オプション) デプロイが失敗した時、または CloudWatch アラームがアクティブ化された時に、既知の正常なリビジョンのデプロイにロールバックするコマンド。
  - (オプション) Auto Scaling スケールインイベント中にライフサイクルイベントフックを生成するためのデプロイ用のコマンド。詳細については、「Amazon EC2 Auto Scaling と CodeDeploy の連携」を参照してください。
  - (オプション) Amazon Simple Notification Service のトピックに発行するトリガーを作成または更新するコマンドにより、そのトピックのサブスクライバーがこのデプロイグループのデプロイおよびインスタンスイベントに関する通知を受け取ります。詳細については、「Monitoring Deployments with Amazon SNS Event Notifications」を参照してください。
- EC2/オンプレミスのデプロイのみ:
  - (オプション) デプロイグループに含まれるインスタンスを一意に識別する代替タグまたはタググループ。
  - (オプション) デプロイグループに追加する代替の Amazon EC2 Auto Scaling グループの名前。
- Amazon ECS のデプロイのみの場合:
  - デプロイする Amazon ECS サービス
  - Application Load Balancer または Network Load Balancer を含むロードバランサーの情報、Amazon ECS デプロイに必要なターゲットグループ、および本番稼働用とオプションのテストリスナー情報。

デプロイグループの詳細オプションの設定

デプロイグループを作成または更新する場合は、そのデプロイグループのデプロイをより詳細に制御および監視できるように、数多くのオプションを設定できます。

このページの情報を使用して、次のトピックで、デプロイグループを使用するときに詳細オプションを設定できます。

- CodeDeploy でアプリケーションを作成する
- CodeDeploy でデプロイグループを作成する
- CodeDeploy を使用して、デプロイグループの設定を変更します。

Amazon SNS 通知トリガー: CodeDeploy デプロイグループにトリガーを追加すると、そのデプロイグループでのデプロイに関連するイベントに関する通知を受信できます。これらの通知は、トリガーのアクションの一部にした Amazon SNS トピックをサブスクライブする受信者に送信されます。

このトリガーが指す Amazon SNS トピックを既に設定している必要があり、CodeDeploy には、このデプロイグループからトピックに発行するためのアクセス許可が必要です。これらのセットアップ手順をまだ完了していない場合は、後でデプロイグループにトリガーを追加できます。

このアプリケーションのデプロイグループのデプロイイベントに関する通知を受信するトリガーを今すぐ作成する場合は、[Create trigger] を選択します。

Amazon EC2 インスタンスにデプロイする場合、インスタンスの通知を作成して、関連する通知を受け取ることができます。

詳細については、「Monitoring Deployments with Amazon SNS Event Notifications」を参照してください。

Amazon CloudWatch アラーム: 指定した期間にわたって単一のメトリクスを監視し、複数の期間にわたり所定の閾値に対するメトリクスの相対値に基づいて 1 つまたは複数のアクションを実行する CloudWatch アラームを作成できます。Amazon EC2 デプロイの場合は、CodeDeploy オペレーションで使用しているインスタンスまたは Amazon EC2 Auto Scaling グループに対してアラームを作成できます。 AWS Lambda および Amazon ECS デプロイでは、Lambda 関数でエラーのアラームを作成できます。

Amazon CloudWatch アラームが、メトリクスが定義された閾値を下回ったり上回ったりすることを検出すると、デプロイが停止するように設定することができます。

デプロイグループに追加する前に、CloudWatch でアラームを作成しておく必要があります。

1. デプロイグループにアラームモニタリングを追加するには、[アラーム]、[アラームの追加] の順に選択します。

2. このデプロイをモニタリングするためにセットアップ済みの CloudWatch アラームの名前を入力します。

   CloudWatch で作成されたのとまったく同じ CloudWatch アラームを入力する必要があります。アラームのリストを表示するには、https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開いて、[ALARM] を選択します。

追加のオプション:

- 追加したアラームを想定せずにデプロイを続行する場合は、[Ignore alarm configuration] を選択します。

  この選択は、後で同じアラームを再び追加しなくても、デプロイグループのアラームの監視を一時的に非アクティブ化する場合に便利です。

- (オプション) CodeDeploy が Amazon CloudWatch からアラームステータスを取得できないときでもデプロイを続行する場合は、[アラームの状態が利用できない場合でも、デプロイを続行する] を選択してください。

> **Note**
>
> このオプションは、CodeDeploy API での AlarmConfiguration オブジェクトの ignorePollAlarmFailure に相当します。

詳細については、「CodeDeploy での CloudWatch アラームを使用したデプロイのモニタリング」を参照してください。

自動ロールバック: デプロイが失敗した場合、または監視しきい値が満たされた場合に、自動的にロールバックするようにデプロイグループまたはデプロイを設定できます。この場合、アプリケーションリビジョンの最後の既知の正常なバージョンがデプロイされます。コンソールを使用してアプリケーションを作成する場合、デプロイグループを作成する場合、またはデプロイグループを更新する場合、デプロイグループのオプション設定を設定できます。新しいデプロイを作成するとき、デプロイグループに指定された自動ロールバック設定をオーバーライドすることもできます。

- 次のいずれかまたは両方を選択して何か問題が発生した場合、デプロイを有効化して最新の既知の正常なリビジョンにロールバックすることができます。

- デプロイが失敗したときにロールバックする。CodeDeploy は、新しいデプロイとして、最後の既知の正常なリビジョンを再デプロイします。
- アラームのしきい値が一致したときにロールバックする。前のステップでこのアプリケーションにアラームを追加した場合、CodeDeploy は、指定された 1 つ以上のアラームがアクティブ化されたときに、最後の既知の正常なリビジョンを再デプロイします。

> **Note**
>
> ロールバック設定を一時的に無視するには、[Disable rollbacks] を選択します。この選択は、後で再び同じ設定をセットアップせずに自動ロールバックを一時的に無効にする場合に便利です。

詳細については、「CodeDeploy を使用した再デプロイおよびデプロイのロールバック」を参照してください。

古いインスタンスの自動更新: 特定の状況下では、CodeDeploy がアプリケーションの古いリビジョンを Amazon EC2 インスタンスにデプロイすることがあります。たとえば、CodeDeploy デプロイの進行中に EC2 インスタンスが Auto Scaling グループ (ASG) で起動された場合、それらのインスタンスには、最新ではなく古いリビジョンのアプリケーションが表示されます。これらのインスタンスを最新の状態にするために、CodeDeploy は自動的に最初のインスタンスの直後にフォローオンのデプロイを開始し、古いインスタンスを更新します。古い EC2 インスタンスを古いリビジョンに残すようにこのデフォルトの動作を変更する場合は、CodeDeploy API または AWS Command Line Interface (CLI) を使用して変更できます。

API を使用して古くなったインスタンスの自動更新を設定するには、`outdatedInstancesStrategy` または `UpdateDeploymentGroup` のアクションに `CreateDeploymentGroup` リクエストパラメータを含めます。詳細については、「AWS CodeDeploy API リファレンス」を参照してください。

を使用して自動更新を設定するには AWS CLI、次のいずれかのコマンドを使用します。

```
aws deploy update-deployment-group arguments --outdated-instances-strategy
UPDATE|IGNORE
```

または...

```
aws deploy create-deployment-group arguments --outdated-instances-strategy
UPDATE|IGNORE
```

... ここで、*##* は、デプロイに必要な引数に置き換えられ、*UPDATE|IGNORE* は自動更新を有効にする場合は UPDATE、無効にする場合は IGNORE に置き換わります。

例:

```
aws deploy update-deployment-group --application-name "MyApp" --current-
deployment-group-name "MyDG" --region us-east-1 --outdated-instances-
strategy IGNORE
```

これらの AWS CLI コマンドの詳細については、AWS CLI「 コマンドリファレンス」を参照してください。

CodeDeploy でデプロイグループを削除する

CodeDeploy コンソール、 AWS CLI、または CodeDeploy APIs を使用して、 AWS アカウントに関連付けられたデプロイグループを削除できます。

> **Warning**
>
> デプロイグループを削除すると、そのデプロイグループに関連付けられるすべての詳細もまた CodeDeploy から削除されます。デプロイグループで使用するインスタンスは変更されません。このアクションを元に戻すことはできません。

トピック

- デプロイグループを削除する (コンソール)
- デプロイグループを削除する (CLI)

デプロイグループを削除する (コンソール)

CodeDeploy コンソールを使用してデプロイグループを削除するには。

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

> **Note**
>
> 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで [デプロイ] を展開し、[アプリケーション] を選択します。
3. アプリケーションのリストで、デプロイグループに関連付けられるアプリケーションの名前を選択します。
4. [アプリケーションの詳細] ページの [デプロイグループ] タブで、削除するデプロイグループの名前を選択します。
5. [デプロイの詳細] ページで、[削除] を選択します。
6. 求められたら、デプロイグループの名前を入力して削除を確認してから、[Delete] を選択します。

デプロイグループを削除する (CLI)

を使用してデプロイグループ AWS CLI を削除するには、delete-deployment-group コマンドを呼び出し、以下を指定します。

- デプロイグループに関連付けられたアプリケーションの名前。アプリケーション名のリストを表示するには、[list-applications] コマンドを呼び出します。
- アプリケーションに関連付けられたデプロイグループの名前。デプロイグループ名のリストを表示するには、[list-deployment-groups] コマンドを呼び出します。

CodeDeploy のアプリケーションリビジョンの操作

CodeDeploy では、リビジョンには CodeDeploy がインスタンスにデプロイするソースファイルのバージョン、または CodeDeploy がインスタンスで実行するスクリプトが含まれています。

リビジョンを計画し、このリビジョンに AppSpec ファイルを追加してから、リビジョンを Amazon S3 または GitHub にプッシュします。リビジョンをプッシュしたら、デプロイできます。

トピック

- CodeDeploy のリビジョンを計画する
- CodeDeploy 用のアプリケーション仕様ファイルをリビジョンに追加
- CodeDeploy リポジトリタイプを選択する
- Amazon S3 に CodeDeploy のリビジョンをプッシュする (EC2/オンプレミスのデプロイのみ)
- CodeDeploy を使用したアプリケーションリビジョン詳細の表示
- CodeDeploy で Amazon S3 にアプリケーションリビジョンを登録する

CodeDeploy のリビジョンを計画する

良い計画は、リビジョンのデプロイをより簡単にします。

AWS Lambda または Amazon ECS コンピューティングプラットフォームへのデプロイの場合、リビジョンは AppSpec ファイルと同じです。次の情報は適用されません。詳細については、「CodeDeploy 用のアプリケーション仕様ファイルをリビジョンに追加」を参照してください。

EC2/オンプレミスコンピューティングプラットフォームにデプロイする場合、開発マシンで空のルートディレクトリ (フォルダ) を作成することから始めます。これはインスタンスで実行するインスタンスまたはスクリプトにデプロイするソースファイル (テキストファイルやバイナリファイル、実行ファイル、パッケージなど) を保存します。

例えば、Linux、macOS、Unix の `/tmp/` のルートフォルダ、または Windows の `c:\temp` のルートフォルダ:

```
/tmp/ or c:\temp (root folder)
  |--content (subfolder)
  |    |--myTextFile.txt
```

```
|    |--mySourceFile.rb
|    |--myExecutableFile.exe
|    |--myInstallerFile.msi
|    |--myPackage.rpm
|    |--myImageFile.png
|--scripts (subfolder)
|    |--myShellScript.sh
|    |--myBatchScript.bat
|    |--myPowerShellScript.ps1
|--appspec.yml
```

また、ルートフォルダには、以下に示すとおり、アプリケーション仕様ファイル (AppSpec ファイル) が含まれている必要があります。詳細については、「CodeDeploy 用のアプリケーション仕様ファイルをリビジョンに追加」を参照してください。

CodeDeploy 用のアプリケーション仕様ファイルをリビジョンに追加

このトピックでは、AppSpec ファイルをデプロイに追加する方法について説明します。また、 AWS Lambda および EC2/オンプレミスデプロイ用の AppSpec ファイルを作成するためのテンプレートも含まれています。

トピック

- Amazon ECS デプロイ用の AppSpec ファイルを追加する
- AWS Lambda デプロイ用の AppSpec ファイルを追加する
- EC2/オンプレミスデプロイに AppSpec ファイルを追加する

Amazon ECS デプロイ用の AppSpec ファイルを追加する

Amazon ECS コンピューティングプラットフォームへのデプロイの場合:

- AppSpec ファイルは、デプロイに使用される Amazon ECS タスク定義、トラフィックをルーティングするコンテナ名とポートマッピング、およびオプションとしてデプロイライフサイクルイベントの後で実行される Lambda 関数を指定します。
- リビジョンは、AppSpec ファイルと同じです。
- AppSpec ファイルは、JSON または YAML を使用して書き込むことができます。

- AppSpec ファイルは、テキストファイルとして保存するか、デプロイ作成時にコンソールに直接入力することができます。詳細については、「Amazon ECS コンピューティングプラットフォームのデプロイの作成 (コンソール)」を参照してください。

AppSpec ファイルを作成するには

1. JSON または YAML テンプレートをテキストエディタ、またはコンソールの AppSpec エディタにコピーします。
2. 必要に応じてテンプレートを変更します。
3. JSON または YAML バリデータを使用して、AppSpec ファイルを検証します。AppSpec エディタを使用している場合は、[デプロイの作成] を選択すると、このファイルが検証されます。
4. テキストエディタを使用している場合は、ファイルを保存します。を使用してデプロイ AWS CLI を作成する場合は、ハードドライブまたは Amazon S3 バケットにある AppSpec ファイルを参照してください。コンソールを使用している場合は、AppSpec ファイルを Amazon S3 にプッシュする必要があります。

Amazon ECS デプロイ用の手順付き YAML AppSpec ファイルテンプレート

以下に示しているのは、利用できるすべてのオプションを含む Amazon ECS デプロイ用の AppSpec ファイルの YAML テンプレートです。hooks セクションで使用するライフサイクルイベントについては、「Amazon ECS のデプロイ向けの AppSpec の「hooks」セクション」を参照してください。

```
# This is an appspec.yml template file for use with an Amazon ECS deployment in
 CodeDeploy.
# The lines in this template that start with the hashtag are
#   comments that can be safely left in the file or
#   ignored.
# For help completing this file, see the "AppSpec File Reference" in the
#   "CodeDeploy User Guide" at
#   https://docs.aws.amazon.com/codedeploy/latest/userguide/app-spec-ref.html
version: 0.0
# In the Resources section, you must specify the following: the Amazon ECS service,
 task definition name,
# and the name and port of the load balancer to route traffic,
# target version, and (optional) the current version of your AWS Lambda function.
Resources:
  - TargetService:
      Type: AWS::ECS::Service
      Properties:
```

```
        TaskDefinition: "" # Specify the ARN of your task definition
 (arn:aws:ecs:region:account-id:task-definition/task-definition-family-name:task-
definition-revision-number)
        LoadBalancerInfo:
          ContainerName: "" # Specify the name of your Amazon ECS application's
 container
          ContainerPort: "" # Specify the port for your container where traffic
 reroutes
# Optional properties
        PlatformVersion: "" # Specify the version of your Amazon ECS Service
        NetworkConfiguration:
          AwsvpcConfiguration:
            Subnets: ["",""] # Specify one or more comma-separated subnets in your
 Amazon ECS service
            SecurityGroups: ["",""] # Specify one or more comma-separated security
 groups in your Amazon ECS service
            AssignPublicIp: "" # Specify "ENABLED" or "DISABLED"
# (Optional) In the Hooks section, specify a validation Lambda function to run during
# a lifecycle event.
Hooks:
# Hooks for Amazon ECS deployments are:
    - BeforeInstall: "" # Specify a Lambda function name or ARN
    - AfterInstall: "" # Specify a Lambda function name or ARN
    - AfterAllowTestTraffic: "" # Specify a Lambda function name or ARN
    - BeforeAllowTraffic: "" # Specify a Lambda function name or ARN
    - AfterAllowTraffic: "" # Specify a Lambda function name or ARN
```

Amazon ECS デプロイテンプレート用の JSON AppSpec ファイル

以下に示しているのは、利用できるすべてのオプションを含む Amazon ECS デプロイ用の AppSpec ファイルの JSON テンプレートです。テンプレートの使用方法については、前のセクションの YAML バージョンのコメントを参照してください。hooks セクションで使用するライフサイクルイベントについては、「Amazon ECS のデプロイ向けの AppSpec の「hooks」セクション」を参照してください。

```
{
 "version": 0.0,
 "Resources": [
  {
   "TargetService": {
    "Type": "AWS::ECS::Service",
    "Properties": {
       "TaskDefinition": "",
```

```
       "LoadBalancerInfo": {
        "ContainerName": "",
        "ContainerPort":
       },
       "PlatformVersion": "",
       "NetworkConfiguration": {
        "AwsvpcConfiguration": {
         "Subnets": [
          "",
          ""
         ],
         "SecurityGroups": [
          "",
          ""
         ],
         "AssignPublicIp": ""
        }
       }
     }
   }
  }
 ],
 "Hooks": [
  {
   "BeforeInstall": ""
  },
  {
   "AfterInstall": ""
  },
  {
   "AfterAllowTestTraffic": ""
  },
  {
   "BeforeAllowTraffic": ""
  },
  {
   "AfterAllowTraffic": ""
  }
 ]
}
```

AWS Lambda デプロイ用の AppSpec ファイルを追加する

AWS Lambda コンピューティングプラットフォームへのデプロイの場合:

- AppSpec ファイルには、デプロイされる Lambda 関数に関する手順が含まれており、デプロイ検証で使用されます。
- リビジョンは、AppSpec ファイルと同じです。
- AppSpec ファイルは、JSON または YAML を使用して書き込むことができます。
- AppSpec ファイルは、テキストファイルとして保存するか、デプロイ作成時にコンソールの AppSpec エディタに直接入力することができます。詳細については、「AWS Lambda コンピューティングプラットフォームのデプロイを作成する (コンソール)」を参照してください。

AppSpec ファイルを作成するには

1. JSON または YAML テンプレートをテキストエディタ、またはコンソールの AppSpec エディタにコピーします。
2. 必要に応じてテンプレートを変更します。
3. JSON または YAML バリデータを使用して、AppSpec ファイルを検証します。AppSpec エディタを使用している場合は、[デプロイの作成] を選択すると、このファイルが検証されます。
4. テキストエディタを使用している場合は、ファイルを保存します。を使用してデプロイ AWS CLI を作成する場合は、ハードドライブまたは Amazon S3 バケットにある AppSpec ファイルを参照してください。コンソールを使用している場合は、AppSpec ファイルを Amazon S3 にプッシュする必要があります。

手順を含む AWS Lambda デプロイ用の YAML AppSpec ファイルテンプレート

hooks セクションで使用するライフサイクルイベントについては、「AWS Lambda デプロイの AppSpec 'hooks' セクション」を参照してください。

```
# This is an appspec.yml template file for use with an AWS Lambda deployment in
 CodeDeploy.
# The lines in this template starting with the hashtag symbol are
#   instructional comments and can be safely left in the file or
#   ignored.
# For help completing this file, see the "AppSpec File Reference" in the
```

```
#   "CodeDeploy User Guide" at
#   https://docs.aws.amazon.com/codedeploy/latest/userguide/app-spec-ref.html
version: 0.0
# In the Resources section specify the name, alias,
# target version, and (optional) the current version of your AWS Lambda function.
Resources:
  - MyFunction: # Replace "MyFunction" with the name of your Lambda function
      Type: AWS::Lambda::Function
      Properties:
        Name: "" # Specify the name of your Lambda function
        Alias: "" # Specify the alias for your Lambda function
        CurrentVersion: "" # Specify the current version of your Lambda function
        TargetVersion: "" # Specify the version of your Lambda function to deploy
# (Optional) In the Hooks section, specify a validation Lambda function to run during
# a lifecycle event. Replace "LifeCycleEvent" with BeforeAllowTraffic
# or AfterAllowTraffic.
Hooks:
    - LifeCycleEvent: "" # Specify a Lambda validation function between double-quotes.
```

AWS Lambda デプロイテンプレートの JSON AppSpec ファイル

以下のテンプレートで、"MyFunction" を AWS Lambda 関数名に置き換えます。オプションの Hooks セクションで、ライフサイクルイベントを BeforeAllowTraffic または AfterAllowTraffic に置き換えます。

Hooks セクションで使用するライフサイクルイベントについては、「AWS Lambda デプロイの AppSpec 'hooks' セクション」を参照してください。

```
{
  "version": 0.0,
  "Resources": [{
   "MyFunction": {
    "Type": "AWS::Lambda::Function",
    "Properties": {
     "Name": "",
     "Alias": "",
     "CurrentVersion": "",
     "TargetVersion": ""
    }
   }
  }],
  "Hooks": [{
    "LifeCycleEvent": ""
```

```
    }
  ]
}
```

EC2/オンプレミスデプロイに AppSpec ファイルを追加する

AppSpec ファイルがない場合、CodeDeploy はアプリケーションリビジョンのソースファイルを宛先にマッピングしたり、EC2/オンプレミスコンピューティングプラットフォームにデプロイするスクリプトを実行したりすることはできません。

各リビジョンには、AppSpec ファイルを 1 つだけ含める必要があります。

AppSpec ファイルをリビジョンに追加するには:

1. テンプレートにテキストエディターをコピーします。
2. 必要に応じてテンプレートを変更します。
3. YAML バリデータを使用して、AppSpec ファイルの有効性をチェックします。
4. リビジョンのルートディレクトリに `appspec.yml` としてファイルを保存します。
5. AppSpec ファイルをルートディレクトリに配置したことを確認するには、次のいずれかのコマンドを実行します。

   - Linux、macOS、Unix の場合:

     ```
     find /path/to/root/directory -name appspec.yml
     ```

     AppSpec ファイルがその場所に見つからない場合は、何も出力されません。

   - Windows の場合:

     ```
     dir path\to\root\directory\appspec.yml
     ```

     そこに AppSpec ファイルが保存されていない場合は、[File Not Found] エラーが表示されます。

6. リビジョンを Amazon S3 または GitHub にプッシュします。

   手順については、「Amazon S3 に CodeDeploy のリビジョンをプッシュする (EC2/オンプレミスのデプロイのみ)」を参照してください。

EC2/オンプレミスデプロイ用の手順付き AppSpec ファイルテンプレート

> **Note**
>
> Windows Server インスタンスへのデプロイでは、runas 要素をサポートしていません。Windows Server インスタンスにデプロイする場合は、AppSpec ファイルに含めないでください。

```
# This is an appspec.yml template file for use with an EC2/On-Premises deployment in
 CodeDeploy.
# The lines in this template starting with the hashtag symbol are
#   instructional comments and can be safely left in the file or
#   ignored.
# For help completing this file, see the "AppSpec File Reference" in the
#   "CodeDeploy User Guide" at
#   https://docs.aws.amazon.com/codedeploy/latest/userguide/app-spec-ref.html
version: 0.0
# Specify "os: linux" if this revision targets Amazon Linux,
#   Red Hat Enterprise Linux (RHEL), or Ubuntu Server
#   instances.
# Specify "os: windows" if this revision targets Windows Server instances.
# (You cannot specify both "os: linux" and "os: windows".)
os: linux
# os: windows
# During the Install deployment lifecycle event (which occurs between the
#   BeforeInstall and AfterInstall events), copy the specified files
#   in "source" starting from the root of the revision's file bundle
#   to "destination" on the Amazon EC2 instance.
# Specify multiple "source" and "destination" pairs if you want to copy
#   from multiple sources or to multiple destinations.
# If you are not copying any files to the Amazon EC2 instance, then remove the
#   "files" section altogether. A blank or incomplete "files" section
#   may cause associated deployments to fail.
files:
  - source:
    destination:
  - source:
    destination:
# For deployments to Amazon Linux, Ubuntu Server, or RHEL instances,
#   you can specify a "permissions"
#   section here that describes special permissions to apply to the files
```

```
#   in the "files" section as they are being copied over to
#   the Amazon EC2 instance.
#   For more information, see the documentation.
# If you are deploying to Windows Server instances,
#   then remove the
#   "permissions" section altogether. A blank or incomplete "permissions"
#   section may cause associated deployments to fail.
permissions:
  - object:
    pattern:
    except:
    owner:
    group:
    mode:
    acls:
      -
    context:
      user:
      type:
      range:
    type:
      -
# If you are not running any commands on the Amazon EC2 instance, then remove
#   the "hooks" section altogether. A blank or incomplete "hooks" section
#   may cause associated deployments to fail.
hooks:
# For each deployment lifecycle event, specify multiple "location" entries
#   if you want to run multiple scripts during that event.
# You can specify "timeout" as the number of seconds to wait until failing the
 deployment
#   if the specified scripts do not run within the specified time limit for the
#   specified event. For example, 900 seconds is 15 minutes. If not specified,
#   the default is 1800 seconds (30 minutes).
#   Note that the maximum amount of time that all scripts must finish executing
#   for each individual deployment lifecycle event is 3600 seconds (1 hour).
#   Otherwise, the deployment will stop and CodeDeploy will consider the deployment
#   to have failed to the Amazon EC2 instance. Make sure that the total number of
 seconds
#   that are specified in "timeout" for all scripts in each individual deployment
#   lifecycle event does not exceed a combined 3600 seconds (1 hour).
# For deployments to Amazon Linux, Ubuntu Server, or RHEL instances,
#   you can specify "runas" in an event to
#   run as the specified user. For more information, see the documentation.
#   If you are deploying to Windows Server instances,
```

```
#   remove "runas" altogether.
# If you do not want to run any commands during a particular deployment
#   lifecycle event, remove that event declaration altogether. Blank or
#   incomplete event declarations may cause associated deployments to fail.
# During the ApplicationStop deployment lifecycle event, run the commands
#   in the script specified in "location" starting from the root of the
#   revision's file bundle.
  ApplicationStop:
    - location:
      timeout:
      runas:
    - location:
      timeout:
      runas:
# During the BeforeInstall deployment lifecycle event, run the commands
#   in the script specified in "location".
  BeforeInstall:
    - location:
      timeout:
      runas:
    - location:
      timeout:
      runas:
# During the AfterInstall deployment lifecycle event, run the commands
#   in the script specified in "location".
  AfterInstall:
    - location:
      timeout:
      runas:
    - location:
      timeout:
      runas:
# During the ApplicationStart deployment lifecycle event, run the commands
#   in the script specified in "location".
  ApplicationStart:
    - location:
      timeout:
      runas:
    - location:
      timeout:
      runas:
# During the ValidateService deployment lifecycle event, run the commands
#   in the script specified in "location".
  ValidateService:
```

```
    - location:
      timeout:
      runas:
    - location:
      timeout:
      runas:
```

CodeDeploy リポジトリタイプを選択する

CodeDeploy に必要なファイルのストレージの場所は リポジトリと呼ばれます。使用するリポジトリは、デプロイで使用するコンピューティングプラットフォームによって異なります。

- EC2/オンプレミス: 1 つ以上のインスタンスにアプリケーションコードをデプロイするには、コードをアーカイブファイルにバンドルし、これをデプロイ処理中に CodeDeploy がアクセスできるリポジトリに配置する必要があります。デプロイ可能なコンテンツと AppSpec ファイルをアーカイブファイルにバンドルしてから、CodeDeploy でサポートされているいずれかのレポジトリタイプにアップロードします。
- AWS Lambda と Amazon ECS: デプロイには AppSpec ファイルが必要です。これは、デプロイ中に次のいずれかの方法でアクセスできます。
  - Amazon S3 バケットから。
  - コンソールの AppSpec エディタへ直接入力されたテキストから 詳細については、「AWS Lambda コンピューティングプラットフォームのデプロイを作成する (コンソール)」および「Amazon ECS コンピューティングプラットフォームのデプロイの作成 (コンソール)」を参照してください。
  - を使用する場合は AWS CLI、ハードドライブまたはネットワークドライブにある AppSpec ファイルを参照できます。詳細については、「AWS Lambda コンピューティングプラットフォームのデプロイを作成する (CLI)」および「Amazon ECS コンピューティングプラットフォームのデプロイの作成 (CLI)」を参照してください。

現在、CodeDeploy は次のリポジトリタイプをサポートしています。

リポジトリタイプ	リポジトリの詳細	サポートされているコンピューティングプラットフォーム
Amazon S3	Amazon Simple Storage Service (Amazon S3) は、安	以下のコンピューティングプラットフォームを使用するデ

	全でスケーラブルなオブジェクトストレージ向けの AWS ソリューションです。Amazon S3 は、データをオブジェクトとして バケット に保存します。オブジェクトは、ファイルと、オプションとしてそのファイルを記述する任意のメタデータで構成されています。 Amazon S3 にオブジェクトを保存するには、バケットにファイルをアップロードします。ファイルをアップロードする際に、オブジェクトにアクセス権限とメタデータを設定することができます。 詳細はこちら: • Amazon S3 にバケットを作成する • Amazon S3 に CodeDeploy のリビジョンをプッシュする (EC2/オンプレミスのデプロイのみ) • CodeDeploy を使用した Amazon S3 からの自動的なデプロイ	プロイでは、Amazon S3 バケット にリビジョンを保存できます。 • EC2/オンプレミス • AWS Lambda • Amazon ECS

GitHub	アプリケーションリビジョンを GitHub リポジトリに保存できます。そのリポジトリのソースコードが変更されるたびに、GitHub リポジトリからデプロイをトリガーできます。 詳細はこちら: • GitHub と CodeDeploy との統合 • チュートリアル: CodeDeploy を使用して、GitHub からアプリケーションをデプロイします。	EC2/オンプレミスデプロイでのみ GitHub リポジトリにリビジョンを保存できます。

Bitbucket	Bitbucket Pipelines の CodeDeploy パイプ を使用すると、EC2 インスタンスのデプロイグループにコードをデプロイすることができます。Bitbucket Pipelines は、Bitbucket Deployments など、継続的インテグレーションと継続的デリバリー (CI/CD) 機能を提供します。CodeDeploy パイプはまず、指定された S3 バケットにアーティファクトをプッシュし、次にバケットからコードアーティファクトをデプロイします。 詳細はこちら: • Bitbucket の CodeDeploy パイプを参照する	EC2/オンプレミスデプロイでのみ BitBucket リポジトリにリビジョンを保存できます。

> **Note**
>
> AWS Lambda デプロイは Amazon S3 リポジトリでのみ機能します。

Amazon S3 に CodeDeploy のリビジョンをプッシュする (EC2/オンプレミスのデプロイのみ)

「CodeDeploy のリビジョンを計画する」で説明されているようにリビジョンを計画して、「CodeDeploy 用のアプリケーション仕様ファイルをリビジョンに追加」で説明されているように AppSpec ファイルをリビジョンに追加したら、コンポーネントファイルをバンドルして、リビジョンを Amazon S3 にプッシュすることができます。Amazon EC2 インスタンスにデプロイする場合は、リビジョンをプッシュしたら、CodeDeploy を使用して、Amazon S3 からインスタンスへのリビジョンをデプロイできます。

> **Note**
>
> CodeDeploy は、GitHub にプッシュされたリビジョンのデプロイにも使用できます。詳細については、GitHub のドキュメントを参照してください。

「CodeDeploy の開始方法」で説明している AWS CLIのセットアップの指示に従っていることを前提としています。これは、後で説明する push コマンドを呼び出す場合に特に重要です。

Amazon S3 バケットがあることを確認してください。バケットの作成 の手順に従います。

デプロイ先が Amazon EC2 インスタンスの場合、ターゲット Amazon S3 バケットを作成するか、ターゲットインスタンスと同じリージョンに存在する必要があります。例えば、米国東部 (バージニア北部) リージョンのインスタンスと米国西部 (オレゴン) リージョンの他のインスタンスにリビジョンをデプロイしたい場合は、米国東部 (バージニア北部) リージョンのバケットにリビジョンのコピーを一つ、米国西部 (オレゴン) の別のバケットに同じリビジョンの別のコピーを持っている必要があります。このシナリオでは、リビジョンがリージョンとバケットの両方で同じであっても、米国東部 (バージニア北部) リージョンと米国西部 (オレゴン) リージョンの 2 つの異なるデプロイを作成する必要があります。

Amazon S3 バケットへのアップロードには、許可が必要です。Amazon S3 バケットポリシーで、これらのアクセス許可を指定できます。たとえば、次の Amazon S3 バケットポリシーでは、ワイルドカード文字 (*) を使用すると、 AWS アカウント111122223333は という名前の Amazon S3 バケット内の任意のディレクトリにファイルをアップロードできますamzn-s3-demo-bucket。

```
{
    "Statement": [
        {
            "Action": [
                "s3:PutObject"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
            "Principal": {
                "AWS": [
                    "111122223333"
                ]
            }
        }
    ]
```

```
}
```

AWS アカウント ID を表示するには、AWS 「アカウント ID の検索」を参照してください。

Amazon S3 バケットポリシーを生成しアタッチする方法については、「バケットポリシーの例」を参照してください。

push コマンドを呼び出すユーザーは、少なくとも、各ターゲット Amazon S3 バケットにリビジョンをアップロードするアクセス権限が必要です。例えば、次のポリシーでは、ユーザーが `amzn-s3-demo-bucket` という名前の Amazon S3 バケット内の任意の場所でリビジョンをアップロードできるようにします。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
        }
    ]
}
```

IAM ポリシーを作成しアタッチする方法については、「ポリシーの使用」を参照してください。

を使用してリビジョンをプッシュする AWS CLI

> **Note**
>
> push コマンドは、アプリケーションのアーティファクトおよび AppSpec ファイルをリビジョンにバンドルします。このリビジョンのファイル形式は、圧縮された ZIP ファイルです。各 は JSON 形式または YAML 形式の AppSpec ファイルであるリビジョンを想定しているため、 AWS Lambda または Amazon ECS デプロイで コマンドを使用することはできません。

push コマンドを呼び出してリビジョンをバンドルしてプッシュし、デプロイします。パラメータは次のとおりです。

- --application-name: (文字列) 必須。アプリケーションバージョンに関連付けられる CodeDeploy アプリケーションの名前。
- --s3-location: (文字列) 必須。Amazon S3 にアップロードされるアプリケーションリビジョンの場所に関する情報。Amazon S3 バケットとキーを指定する必要があります。キーは、リビジョンの名前です。CodeDeploy はコンテンツをアップロードする前に zip 形式で圧縮します。形式 `s3://amzn-s3-demo-bucket/your-key.zip` を使用します。
- --ignore-hidden-files または --no-ignore-hidden-files: (ブール値) オプション。`--no-ignore-hidden-files` フラグ (デフォルト) を使用して、隠しファイルをバンドルし Amazon S3 にアップロードします。隠しファイルをバンドルして Amazon S3 へアップロードしない `--ignore-hidden-files` フラグを使用する。
- --source(文字列) オプション。zip 圧縮され Amazon S3 にアップロードされる開発マシン上の、デプロイするコンテンツおよび AppSpec ファイルの場所。この場所は、現在のディレクトリに対する相対パスとして指定されます。相対パスが指定されていない場合、または 1 つのピリオド (「.」) がパスとして使用される場合、現在のディレクトリが使用されます。
- --description(文字列) オプション。アプリケーションリビジョンを要約したコメントです。指定しない場合、デフォルトの文字列「Uploaded by AWS CLI 'time' UTC」が使用されます。ここで、「time」は協定世界時 (UTC) の現在のシステム時刻です。

を使用して AWS CLI 、Amazon EC2 デプロイのリビジョンをプッシュできます。プッシュコマンドの例は、次のようになります。

Linux、macOS、または Unix の場合:

```
aws deploy push \
  --application-name WordPress_App \
  --description "This is a revision for the application WordPress_App" \
  --ignore-hidden-files \
  --s3-location s3://amzn-s3-demo-bucket/WordPressApp.zip \
  --source .
```

Windows の場合:

```
aws deploy push --application-name WordPress_App --description "This is a revision for
 the application WordPress_App" --ignore-hidden-files --s3-location s3://amzn-s3-demo-
bucket/WordPressApp.zip --source .
```

このコマンドは次のことを行います。

- バンドルされたファイルを WordPress_App という名前のアプリケーションに関連付けます。
- リビジョンに説明をアタッチします。
- 隠しファイルを無視します。
- リビジョンに WordPressApp.zip という名前を付け、amzn-s3-demo-bucket というバケットにプッシュします。
- ルートディレクトリ内のすべてのファイルをリビジョンにバンドルします。

プッシュが成功したら、 AWS CLI または CodeDeploy コンソールを使用して Amazon S3 からリビジョンをデプロイできます。を使用してこのリビジョンをデプロイするには AWS CLI:

Linux、macOS、または Unix の場合:

```
aws deploy create-deployment \
  --application-name WordPress_App \
  --deployment-config-name your-deployment-config-name \
  --deployment-group-name your-deployment-group-name \
  --s3-location bucket=amzn-s3-demo-bucket,key=WordPressApp.zip,bundleType=zip
```

Windows の場合:

```
aws deploy create-deployment --application-name WordPress_App --deployment-config-
name your-deployment-config-name --deployment-group-name your-deployment-group-name --
s3-location bucket=amzn-s3-demo-bucket,key=WordPressApp.zip,bundleType=zip
```

詳細については、「CodeDeploy でデプロイを作成する」を参照してください。

CodeDeploy を使用したアプリケーションリビジョン詳細の表示

CodeDeploy コンソール、 AWS CLI、または CodeDeploy APIs を使用して、指定したアプリケーションの AWS アカウントに登録されているすべてのアプリケーションリビジョンの詳細を表示できます。

リビジョンの登録の詳細については、「CodeDeploy で Amazon S3 にアプリケーションリビジョンを登録する」を参照してください。

トピック

- アプリケーションリビジョンの詳細の表示 (コンソール)
- アプリケーションリビジョンの詳細の表示 (CLI)

アプリケーションリビジョンの詳細の表示 (コンソール)

アプリケーションリビジョンの詳細を表示するには:

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

> **Note**
>
> 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで [デプロイ] を展開し、[アプリケーション] を選択します。

> **Note**
>
> エントリが表示されない場合は、正しいリージョンが選択されていることを確認しましょう。ナビゲーションバーのリージョンセレクターで、「AWS 全般のリファレンス」の「リージョンとエンドポイント」に一覧表示されているいずれかのリージョンを選択します。CodeDeploy は、これらのリージョンでのみサポートされています。

3. 表示するリビジョンと関連するアプリケーションの名前を選択します。
4. [アプリケーションの詳細] ページで [リビジョン] タブを選択し、アプリケーションに登録されているリビジョンを一覧表示します。リビジョンを選択し、[詳細を表示] を選択します。

アプリケーションリビジョンの詳細の表示 (CLI)

を使用してアプリケーションリビジョン AWS CLI を表示するには、 get-application-revision コマンドまたは list-application-revisions コマンドを呼び出します。

> **Note**
>
> GitHub のリファレンスは、EC2 オンプレミスデプロイに対するデプロイにのみ適用されます。 AWS Lambda デプロイのリビジョンは GitHub では機能しません。

単一のアプリケーションリビジョンに関する詳細を表示するには、get-application-revision コマンドを呼び出し、以下を指定します。

- アプリケーション名。アプリケーション名を取得するには、list-applications コマンドを呼び出します。
- GitHub に保存されたリビジョンの場合、GitHub レポジトリ名と、リポジトリにプッシュされたアプリケーションリビジョンを参照するコミットの ID。
- Amazon S3 に保存されたリビジョンの場合、リビジョンを含む Amazon S3 バケット名、アップロードされたアーカイブファイルの名前とファイル形式、さらにオプションで、アーカイブファイルの Amazon S3 バージョン ID と ETag。バージョン IDかETag、またはその両方を register-application-revision の呼び出し中に指定した場合、ここで指定する必要があります。

複数のアプリケーションリビジョンに関する詳細を表示するには、list-application-revisions コマンドを呼び出し、以下を指定します。

- アプリケーション名。アプリケーション名を取得するには、list-applications コマンドを呼び出します。
- オプションで、Amazon S3 アプリケーションリビジョン詳細のみを表示する場合、リビジョンを含む Amazon S3 バケット名。
- オプションで、Amazon S3 アプリケーションリビジョンの詳細のみを表示する場合、 Amazon S3 アプリケーションリビジョンの検索を制限するプレフィックス文字列 (指定されていない場合、CodeDeploy は一致するすべての Amazon S3 アプリケーションリビジョンを表示します。)
- オプションで、各リビジョンがデプロイグループのターゲットリビジョンかどうかに基づいてリビジョンの詳細を表示するかどうか (指定されていない場合、CodeDeploy は一致するすべてのリビジョンを表示します。)
- オプションで、リビジョンの詳細の一覧をソートする際の基準とする列名と順序 (指定されていない場合、CodeDeploy は任意の順序で結果を表示します。)

すべてのリビジョン、または Amazon S3 に保存されたリビジョンのみを表示できます。GitHub に保存されているリビジョンのみを表示することはできません。

CodeDeploy で Amazon S3 にアプリケーションリビジョンを登録する

既に push コマンドを呼び出してアプリケーションリビジョンを Amazon S3 にプッシュしている場合、リビジョンを登録する必要はありません。ただし、他の方法で Amazon S3 にリビジョンをアップロードし、リビジョンを CodeDeploy コンソールに表示するか、 AWS CLIを通じて表示する場合は、次のステップに従って最初にリビジョンを登録します。

アプリケーションリビジョンを GitHub リポジトリにプッシュし、そのリビジョンを CodeDeploy コンソールまたは で表示する場合は AWS CLI、以下のステップも実行する必要があります。

Amazon S3 AWS CLI または GitHub でアプリケーションリビジョンを登録するには、 または CodeDeploy APIs のみを使用できます。 Amazon S3

トピック

- CodeDeploy で Amazon S3 にリビジョンを登録する (CLI)
- CodeDeploy による GitHub でのリビジョンの登録 (CLI)

CodeDeploy で Amazon S3 にリビジョンを登録する (CLI)

1. Amazon S3 にリビジョンをアップロードする
2. register-application-revision コマンドを呼び出し、以下を指定します。
   - アプリケーション名。アプリケーション名のリストを表示するには、[list-applications] コマンドを呼び出します。
   - 登録するリビジョンに関する情報:
     - リビジョンを含む Amazon S3 バケットの名前。
     - アップロードされたリビジョンの名前とファイル形式。 AWS Lambda デプロイの場合、リビジョンは、JSON または YAML で書き込まれた AppSpec ファイルです。EC2/オンプレミスデプロイの場合、このリビジョンには、CodeDeploy からインスタンスにデプロイされるソースファイルか、CodeDeploy によってインスタンスで実行されるスクリプトのバージョンが含まれます。

> **Note**
>
> tar および圧縮 tar アーカイブファイル形式 (.tar および .tar.gz) は、Windows Server インスタンスではサポートされていません。

  - (オプション) リビジョンの Amazon S3 バージョン識別子。(バージョン識別子を指定しない場合、CodeDeploy は最新バージョンを使用します)。
  - (オプション) のリビジョン ETag。(ETag が指定されていない場合、CodeDeploy はオブジェクトの検証をスキップします)
- (オプション) リビジョンに関連付ける説明。

Amazon S3 のリビジョンに関する情報は、register-application-revision 呼び出しの一部としてこの構文を使用して、コマンドラインで指定できます。(`version` および `eTag` はオプションです。)

EC2/オンプレミスのデプロイ向けリビジョンファイルの場合

```
--s3-location bucket=string,key=string,bundleType=tar|tgz|
zip,version=string,eTag=string
```

AWS Lambda デプロイのリビジョンファイルの場合:

```
--s3-location bucket=string,key=string,bundleType=JSON|YAML,version=string,eTag=string
```

CodeDeploy による GitHub でのリビジョンの登録 (CLI)

> **Note**
>
> AWS Lambda デプロイは GitHub では機能しません。

1. GitHub リポジトリにリビジョンをアップロードします。
2. register-application-revision コマンドを呼び出し、以下を指定します。
   - アプリケーション名。アプリケーション名のリストを表示するには、[list-applications] コマンドを呼び出します。
   - 登録するリビジョンに関する情報:

    - スラッシュ (/)、リポジトリの名前が後に続く、リポジトリに割り当てられたリビジョンを含む GitHub ユーザーまたは組織の名前。
    - リポジトリのリビジョンを参照するコミットの ID。
- (オプション) リビジョンに関連付ける説明。

GitHub のリビジョンについての情報は、この構文を register-application-revision の一部として使用することでコマンドラインで指定できます。

```
--github-location repository=string,commitId=string
```

CodeDeploy でのデプロイグループの使用

CodeDeploy では、デプロイはプロセスであり、プロセスには 1 つ以上のインスタンスのコンテンツをインストールするコンポーネントが含まれます。このコンテンツはコード、ウェブ、設定ファイル、実行可能ファイル、パッケージ、スクリプトなどによって構成できます。CodeDeploy は、指定した設定ルールに従って、ソースリポジトリに格納されたコンテンツをデプロイします。

EC2 オンプレミスコンピューティングプラットフォームを使用する場合、インスタンスの同じセットへの2 つのデプロイは同時に実行できます。

CodeDeploy は、2 つのデプロイタイプのオプション、インプレースデプロイおよび Blue/Green デプロイを提供します。

- インプレイスデプロイ: デプロイグループの各インスタンス上のアプリケーションが停止され、最新のアプリケーションリビジョンがインストールされて、新バージョンのアプリケーションが開始され検証されます。ロードバランサーを使用し、デプロイ中はインスタンスが登録解除され、デプロイ完了後にサービスに復元されるようにできます。EC2 オンプレミスコンピューティングプラットフォームを使用するデプロイのみが、インプレイスデプロイを使用できます。インプレイスデプロイの詳細については、「インプレースデプロイの概要」を参照してください。
- Blue/Green デプロイ: デプロイの動作は、使用するコンピューティングプラットフォームにより異なります。
  - EC2 オンプレミスコンピューティングプラットフォームの Blue/Green: 以下のステップを使用して、デプロイグループのインスタンス (元の環境) がインスタンスの別のセット (置き換え先環境) に置き換えられます。
    - 置き換え先の環境のインスタンスがプロビジョニングされます。
    - 最新のアプリケーションリビジョンは、置き換え先インスタンスにインストールされます。
    - オプションの待機時間は、アプリケーションのテストやシステム検証などのアクティビティに対して発生します。
    - 置き換え先環境のインスタンスは、1 つまたは複数の Elastic Load Balancing ロードバランサーに登録され、トラフィックは、それらに再ルーティングされます。元の環境のインスタンスは、登録が解除され、終了するか、他の使用のために実行することができます。

> **Note**
>
> EC2/オンプレミスのコンピューティングプラットフォームを使用する場合は、blue/green デプロイが Amazon EC2 インスタンスでのみ機能することに注意してください。

- AWS Lambda または Amazon ECS コンピューティングプラットフォームの Blue/Green: トラフィックは、Canary、線形、または all-at-onceデプロイ設定に従って増分でシフトされます。
- 経由のブルー/グリーンデプロイ CloudFormation: スタック CloudFormation の更新の一環として、トラフィックは現在のリソースから更新されたリソースに移行されます。現時点では、ECS blue/green デプロイのみがサポートされています。

ブルー/グリーンデプロイの詳細については、「Blue/Green デプロイの概要」を参照してください。

Amazon S3 からの自動的なデプロイの詳細については、「CodeDeployを使用して Amazon S3 から自動的にデプロイする」を参照してください。

トピック

- CodeDeploy でデプロイを作成する
- CodeDeploy デプロイの詳細を表示する
- CodeDeploy EC2/オンプレミスデプロイのログデータの表示
- CodeDeploy でデプロイの停止
- CodeDeploy を使用した再デプロイおよびデプロイのロールバック
- 別の AWS アカウントにアプリケーションをデプロイする
- CodeDeploy エージェントを使用してローカルマシン上のデプロイパッケージを検証する

CodeDeploy でデプロイを作成する

CodeDeploy コンソール、 AWS CLI、または CodeDeploy APIs を使用して、既に Amazon S3 にプッシュしたアプリケーションリビジョンをインストールするデプロイを作成できます。デプロイが EC2/オンプレミスコンピューティングプラットフォームの場合は、デプロイグループのインスタンスに GitHub をインストールします。

デプロイを作成するプロセスは、デプロイで使用されるコンピューティングプラットフォームによって異なります。

トピック

- デプロイの前提条件
- Amazon ECS コンピューティングプラットフォームのデプロイの作成 (コンソール)
- AWS Lambda コンピューティングプラットフォームのデプロイを作成する (コンソール)

- EC2/オンプレミスコンピューティングプラットフォームのデプロイ作成 (コンソール)
- Amazon ECS コンピューティングプラットフォームのデプロイの作成 (CLI)
- AWS Lambda コンピューティングプラットフォームのデプロイを作成する (CLI)
- EC2/ オンプレミスコンピューティングプラットフォームのデプロイ作成 (CLI)
- を使用して Amazon ECS ブルー/グリーンデプロイを作成する CloudFormation

デプロイの前提条件

デプロイを開始する前に、以下のステップが完了していることを確認します。

AWS Lambda コンピューティングプラットフォームでのデプロイの前提条件

- 少なくとも 1 つのデプロイグループを含むアプリケーションを作成します。詳細については、「CodeDeploy でアプリケーションを作成する」および「CodeDeploy でデプロイグループを作成する」を参照してください。
- デプロイする Lambda 関数バージョンを指定するアプリケーションリビジョン (AppSpec ファイル とも呼ばれる) を準備します。AppSpec ファイルでは、Lambda 関数を指定してデプロイを検証することもできます。詳細については、CodeDeploy のアプリケーションリビジョンの操作 を参照してください。
- デプロイにカスタムデプロイ設定を使用する場合、デプロイプロセスを開始する前にカスタムデプロイ設定を作成します。詳細については、「Create a Deployment Configuration」を参照してください。

EC2/オンプレミスコンピューティングプラットフォームのデプロイ前提条件

- インプレースデプロイの場合は、デプロイ先のインスタンスを作成または設定します。詳細については、「CodeDeploy のためにインスタンスを用いた操作」を参照してください。Blue/Green デプロイのために、置き換え先環境のテンプレートとして使用する既存の Amazon EC2 Auto Scaling グループがあるか、元の環境として指定する 1 つ以上のインスタンスまたは Amazon EC2 Auto Scaling グループがあります。詳細については、「チュートリアル: CodeDeploy を使用して、Auto Scaling グループにアプリケーションをデプロイします。」および「CodeDeploy と Amazon EC2 Auto Scaling の統合」を参照してください。
- 少なくとも 1 つのデプロイグループを含むアプリケーションを作成します。詳細については、「CodeDeploy でアプリケーションを作成する」および「CodeDeploy でデプロイグループを作成する」を参照してください。

- デプロイグループのインスタンスにデプロイするアプリケーションリビジョンを準備します。詳細については、「CodeDeploy のアプリケーションリビジョンの操作」を参照してください。
- デプロイにカスタムデプロイ設定を使用する場合、デプロイプロセスを開始する前にカスタムデプロイ設定を作成します。詳細については、「Create a Deployment Configuration」を参照してください。
- Amazon S3 バケットからアプリケーションリビジョンをデプロイする場合、バケットはデプロイグループのインスタンスと同じ AWS リージョンにあります。
- Amazon S3 バケットからアプリケーションのリビジョンをデプロイする場合、Amazon S3 バケットポリシーをバケットに適用済みです。このポリシーでは、アプリケーションリビジョンをダウンロードするために必要なアクセス許可をインスタンスに付与します。

  例えば、次の Amazon S3 バケットポリシーは、ARN `arn:aws:iam::444455556666:role/CodeDeployDemo` を含む IAM インスタンスプロファイルがアタッチされた Amazon EC2 インスタンスが、`amzn-s3-demo-bucket` という名前の Amazon S3 バケットの任意の場所からダウンロードすることを許可します。

```
{
    "Statement": [
        {
            "Action": [
                "s3:Get*",
                "s3:List*"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::444455556666:role/CodeDeployDemo"
                ]
            }
        }
    ]
}
```

  次の Amazon S3 バケットポリシーは、ARN `arn:aws:iam::444455556666:user/CodeDeployUser` を含む IAM ユーザーに関連付けられたオンプレミスインスタンスが、`amzn-s3-demo-bucket` という名前の Amazon S3 バケット内の任意の場所からダウンロードすることを許可します。

```
{
    "Statement": [
        {
            "Action": [
                "s3:Get*",
                "s3:List*"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::444455556666:user/CodeDeployUser"
                ]
            }
        }
    ]
}
```

Amazon S3 バケットポリシーを生成しアタッチする方法の詳細については、「バケットポリシーの例」を参照してください。

- Blue/Green デプロイを作成する場合、またはインプレースデプロイのためにデプロイグループにオプションの Classic Load Balancer、Application Load Balancer、Network Load Balancer を指定している場合、Amazon VPC を使用して、少なくとも 2 つのサブネットを含む VPC を作成していることになります。(CodeDeploy で使用する Elastic Load Balancing では、ロードバランサーグループ内のすべてのインスタンスが 1 つの VPC 内にあることが必要です)。

  VPC を作成済みでない場合は、「Amazon VPC 入門ガイド」を参照してください。

- ブルー/グリーンデプロイを作成する場合、Elastic Load Balancing に、少なくとも 1 つの Classic Load Balancer、Application Load Balancer または Network Load Balancerを設定し、これを使用して元の環境を構成するインスタンスを登録しています。

> **Note**
>
> 置き換え先環境内のインスタンスは後でロードバランサーを使用して登録されます。

ロードバランサーの設定の詳細については、「CodeDeploy Amazon EC2 デプロイ用の Elastic Load Balancing でロードバランサーをセットアップする」および「CodeDeploy Amazon ECS デ

プロイ用のロードバランサー、ターゲットグループ、リスナーをセットアップする」を参照してください。

を使用した Blue/Green デプロイのデプロイ前提条件 CloudFormation

- テンプレートでは、CodeDeploy アプリケーションまたはデプロイグループのリソースをモデル化する必要はありません。
- テンプレートには、少なくとも 2 つのサブネットを含む Amazon VPC を使用する VPC のリソースを含める必要があります。
- テンプレートには、トラフィックをターゲットグループに誘導するために使用される Elastic Load Balancing の 1 つまたは複数の Classic Load Balancer、Application Load Balancer、または Network Load Balancer のリソースを含める必要があります。

Amazon ECS コンピューティングプラットフォームのデプロイの作成 (コンソール)

このトピックでは、コンソールを使用して Amazon ECS サービスをデプロイする方法を示します。詳細については、「チュートリアル: Amazon ECS へアプリケーションをデプロイする」および「チュートリアル: 検証テストを使用して Amazon ECS サービスをデプロイする」を参照してください。

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. 次のいずれかを行います。

   - アプリケーションをデプロイする場合は、ナビゲーションペインで [デプロイ] を展開し、[アプリケーション] を選択します。デプロイするアプリケーションの名前を選択します。アプリケーションの [Compute platform (コンピューティングプラットフォーム)] 列が [Amazon ECS] になっていることを確認します。
   - デプロイを再デプロイする場合は、ナビゲーションペインで [デプロイ] を展開し、[デプロイ] を選択します。再デプロイするデプロイを選択し、[アプリケーション] 列で、アプリケーショ

ンの名前を選択します。デプロイの [Compute platform (コンピューティングプラットフォーム)] 列が [Amazon ECS] になっていることを確認します。

3. [デプロイ] タブで、[デプロイの作成] を選択します。

> **Note**
>
> アプリケーションをデプロイするには、アプリケーションにデプロイグループが必要です。アプリケーションにデプロイグループがない場合は、[デプロイグループ] タブで、[デプロイグループの作成] を選択します。詳細については、「CodeDeploy でデプロイグループを作成する」を参照してください。

4. [デプロイグループ] で、このデプロイで使用するデプロイグループを選択します。
5. [リビジョンの場所] の横で、リビジョンの場所を選択します。

   - [My application is stored in Amazon S3 (Amazon S3 に保存されたアプリケーション)] - 詳細に関しては、Amazon S3 バケットに格納されているリビジョンについての情報を指定します を参照し、ステップ 6 に戻ります。
   - [Use AppSpec editor (AppSpec エディタの使用)] — JSON または YAML を選択し、エディタに AppSpec ファイルを入力します。AppSpec ファイルを保存するには、[テキストファイルとして保存] を選択します。これらのステップの最後で [Deploy (デプロイ)] を選択すると、JSON または YAML が有効ではないというエラーが発生します。AppSpec ファイルの作成の詳細については、「CodeDeploy 用のアプリケーション仕様ファイルをリビジョンに追加」を参照してください。
6. (オプション) [デプロイの説明] に、このデプロイの説明を入力します。
7. (オプション) [Rollback configuration overrides] で、該当する場合は、デプロイグループに指定されているものとは別の自動ロールバックオプションをこのデプロイに指定できます。

   CodeDeploy のロールバックの詳細については、「デプロイと再デプロイのロールバック」および「CodeDeploy を使用した再デプロイおよびデプロイのロールバック」を参照してください。

   以下から選択します。

   - [Roll back when a deployment fails (デプロイが失敗したときにロールバックする)] — CodeDeploy は既知の正常なリビジョンを新しいデプロイとして再デプロイします。
   - [Roll back when alarm thresholds are met (アラームのしきい値が一致したときにロールバックする)] — デプロイグループにアラームが追加された場合、1 つ以上の指定したアラームがアクティブ化されたときに、CodeDeploy は既知の正常なリビジョンを再デプロイします。

- [ロールバックを無効にする] — このデプロイのロールバックを実行しません。

8. [デプロイの作成] を選択します。

   デプロイの状態を追跡するには、「CodeDeploy デプロイの詳細を表示する」を参照してください。

AWS Lambda コンピューティングプラットフォームのデプロイを作成する (コンソール)

このトピックでは、コンソールを使用して Lambda 関数をデプロイする方法を示します。

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. 次のいずれかを行います。

   - アプリケーションをデプロイする場合は、ナビゲーションペインで [デプロイ] を展開し、[アプリケーション] を選択します。デプロイするアプリケーションの名前を選択します。アプリケーションの [Compute platform (コンピューティングプラットフォーム)] 列が [AWS Lambda] になっていることを確認します。
   - デプロイを再デプロイする場合は、ナビゲーションペインで [デプロイ] を展開し、[デプロイ] を選択します。再デプロイするデプロイを選択し、[アプリケーション] 列で、アプリケーションの名前を選択します。デプロイ の [Compute platform (コンピューティングプラットフォーム)] 列が [AWS Lambda] になっていることを確認します。

3. [デプロイ] タブで、[デプロイの作成] を選択します。

   > **Note**
   >
   > アプリケーションをデプロイするには、アプリケーションにデプロイグループが必要です。アプリケーションにデプロイグループがない場合は、[デプロイグループ] タブで、[デプロイグループの作成] を選択します。詳細については、「CodeDeploy でデプロイグループを作成する」を参照してください。

4. [デプロイグループ] で、このデプロイで使用するデプロイグループを選択します。
5. [リビジョンの場所] の横で、リビジョンの場所を選択します。

   - [My application is stored in Amazon S3 (Amazon S3 に保存されたアプリケーション)] - 詳細に関しては、Amazon S3 バケットに格納されているリビジョンについての情報を指定します を参照し、ステップ 6 に戻ります。
   - [Use AppSpec editor (AppSpec エディタの使用)] — JSON または YAML を選択し、エディタに AppSpec ファイルを入力します。AppSpec ファイルを保存するには、[テキストファイルとして保存] を選択します。これらのステップの最後で [Deploy (デプロイ)] を選択すると、JSON または YAML が有効ではないというエラーが発生します。AppSpec ファイルの作成の詳細については、「CodeDeploy 用のアプリケーション仕様ファイルをリビジョンに追加」を参照してください。
6. (オプション) [デプロイの説明] に、このデプロイの説明を入力します。
7. (オプション) [Deployment group overrides (デプロイグループのオーバーライド)] を展開し、トラフィックを Lambda 関数バージョンにシフトする方法を制御するデプロイ設定を選択します。この場合、デプロイグループに指定したものとは異なるデプロイ設定を選択します。

   詳細については、「AWS Lambda コンピューティングプラットフォームでのデプロイ設定」を参照してください。
8. (オプション) [Rollback configuration overrides] で、該当する場合は、デプロイグループに指定されているものとは別の自動ロールバックオプションをこのデプロイに指定できます。

   CodeDeploy のロールバックの詳細については、「デプロイと再デプロイのロールバック」および「CodeDeploy を使用した再デプロイおよびデプロイのロールバック」を参照してください。

   以下から選択します。

   - [Roll back when a deployment fails (デプロイが失敗したときにロールバックする)] — CodeDeploy は既知の正常なリビジョンを新しいデプロイとして再デプロイします。
   - [Roll back when alarm thresholds are met (アラームのしきい値が一致したときにロールバックする)] — デプロイグループにアラームが追加された場合、1 つ以上の指定したアラームがアクティブ化されたときに、CodeDeploy は既知の正常なリビジョンを再デプロイします。
   - [ロールバックを無効にする] — このデプロイのロールバックを実行しません。
9. [デプロイの作成] を選択します。

   デプロイの状態を追跡するには、「CodeDeploy デプロイの詳細を表示する」を参照してください。

EC2/オンプレミスコンピューティングプラットフォームのデプロイ作成 (コンソール)

このトピックでは、コンソールを使用して Amazon EC2 またはオンプレミスサーバーにアプリケーションをデプロイする方法を示します。

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. 次のいずれかを行います。

   - アプリケーションをデプロイする場合は、ナビゲーションペインで [デプロイ] を展開し、[アプリケーション] を選択します。デプロイするアプリケーションの名前を選択します。アプリケーションの [コンピューティングプラットフォーム] 列が [EC2/オンプレミス] になっていることを確認します。
   - デプロイを再デプロイする場合は、ナビゲーションペインで [デプロイ] を展開し、[デプロイ] を選択します。再デプロイするデプロイを見つけ、[アプリケーション] 列で、アプリケーションの名前を選択します。デプロイ の [コンピューティングプラットフォーム] 列が [EC2/オンプレミス] になっていることを確認します。

3. [デプロイ] タブで、[デプロイの作成] を選択します。

   > **Note**
   >
   > アプリケーションをデプロイするには、アプリケーションにデプロイグループが必要です。アプリケーションにデプロイグループがない場合は、[デプロイグループ] タブで、[デプロイグループの作成] を選択します。詳細については、「CodeDeploy でデプロイグループを作成する」を参照してください。

4. [デプロイグループ] で、このデプロイで使用するデプロイグループを選択します。
5. [リポジトリタイプ] の横で、リビジョンが保存されているリポジトリタイプを選択します。

- [My application is stored in Amazon S3 (Amazon S3 に保存されたアプリケーション)] - 詳細に関しては、Amazon S3 バケットに格納されているリビジョンについての情報を指定します を参照し、ステップ 6 に戻ります。
- [My application is stored in GitHub (GitHub に保存されたアプリケーション)] — 詳細については、以下の「GitHub リポジトリに格納されているリビジョンについての情報を指定します」を参照し、ステップ 6 に戻ります。

6. (オプション) [デプロイの説明] に、このデプロイの説明を入力します。
7. (オプション) [Override deployment configuration (デプロイ設定の上書き)] を展開してデプロイ設定を選択し、デプロイグループに指定したものとは異なる Amazon EC2 またはオンプレミスサーバーにトラフィックをシフトする方法を制御します。

   詳細については、「CodeDeploy でデプロイ設定を使用する」を参照してください。
8. a. `ApplicationStop` ライフサイクルイベントが失敗した場合でもインスタンスへのデプロイを成功させる場合は、[Don't fail the deployment if the ApplicationStop lifecycle event fails (ApplicationStop ライフサイクルイベントの障害でデプロイを失敗させない)] を選択します。

   b. [Additional deployment behavior settings (追加のデプロイ動作設定)] を展開して、以前に成功したデプロイに含まれていなかったデプロイターゲットの場所にあるファイルを CodeDeploy で処理する方法を指定します。

      以下から選択します。

      - [Fail the deployment (デプロイの失敗)] — エラーが報告され、デプロイのステータスが `Failed` に変更されます。
      - [コンテンツの上書き] — デプロイ先に同じ名前のファイルが存在する場合は、アプリケーションリビジョンのバージョンによって置き換えられます。
      - [コンテンツの保持] — デプロイ先に同じ名前のファイルが存在する場合は、ファイルが保持され、アプリケーションリビジョンのバージョンはインスタンスにコピーされません。

      詳細については、「既存のコンテンツでのロールバック動作」を参照してください。
9. (オプション) [Rollback configuration overrides] で、該当する場合は、デプロイグループに指定されているものとは別の自動ロールバックオプションをこのデプロイに指定できます。

CodeDeploy のロールバックの詳細については、「デプロイと再デプロイのロールバック」および「CodeDeploy を使用した再デプロイおよびデプロイのロールバック」を参照してください。

以下から選択します。

- [Roll back when a deployment fails (デプロイが失敗したときにロールバックする)] — CodeDeploy は既知の正常なリビジョンを新しいデプロイとして再デプロイします。
- [Roll back when alarm thresholds are met (アラームのしきい値が一致したときにロールバックする)] — デプロイグループにアラームが追加された場合、1 つ以上の指定したアラームがアクティブ化されたときに、CodeDeploy は既知の正常なリビジョンをデプロイします。
- [ロールバックを無効にする] — このデプロイのロールバックを実行しません。

10. デプロイの開始 を選択します。

デプロイの状態を追跡するには、「CodeDeploy デプロイの詳細を表示する」を参照してください。

トピック

- Amazon S3 バケットに格納されているリビジョンについての情報を指定します
- GitHub リポジトリに格納されているリビジョンについての情報を指定します

Amazon S3 バケットに格納されているリビジョンについての情報を指定します

「EC2/オンプレミスコンピューティングプラットフォームのデプロイ作成 (コンソール)」の手順を使用している場合は、以下のステップに従って Amazon S3 バケットに保存されているアプリケーションリビジョンの詳細を追加してください。

1. リビジョンの Amazon S3 リンクを [Revision location (リビジョンの場所)] ボックスにコピーします。リンク値の確認

   a. 別のブラウザタブで

      にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/s3/ で Amazon S3 コンソールを開きます。

      リビジョンを参照して選択します。

   b. [Properties] ペインが表示されていない場合、[Properties] ボタンを選択します。

c. [Properties] ペインで、[Link] フィールドの値をCodeDeploy コンソールの [Revision location] ボックスにコピーします。

ETag (ファイルチェックサム) をリビジョンの場所の一部として指定するには。

- [リンク] フィールド値が **?versionId=*versionId*** で終わる場合、**&etag=** および ETag を [リンク] フィールド値の末尾に追加します。
- [リンク] フィールド値がバージョン ID を指定していない場合、**?etag=** および ETag を [リンク] フィールド値の末尾に追加します。

> **Note**
>
> [Link] フィールドの値をコピーするように簡単ではありませんが、次のいずれかの形式でリビジョンの場所を入力することもできます。
>
> **s3://*bucket-name*/*folders*/*objectName***
> **s3://*bucket-name*/*folders*/*objectName*?versionId=*versionId***
> **s3://*bucket-name*/*folders*/*objectName*?etag=*etag***
> **s3://*bucket-name*/*folders*/*objectName*?versionId=*versionId*&etag=*etag***
> ***bucket-name*.s3.amazonaws.com/*folders*/*objectName***

2. [File type] リストに、ファイル形式を検出できないというメッセージが表示された場合は、リビジョンのファイル形式を選択します。それ以外の場合は、検出されたファイル形式を使用します。

GitHub リポジトリに格納されているリビジョンについての情報を指定します

「EC2/オンプレミスコンピューティングプラットフォームのデプロイ作成 (コンソール)」の手順を使用している場合は、以下のステップに従って GitHub リポジトリに保存されているアプリケーションリビジョンの詳細を追加してください。

1. [Connect to GitHub] で、次のいずれかを実行します。

   - GitHub アカウントに対する CodeDeploy アプリケーションの接続を作成するには、ウェブブラウザの別のタブで GitHub からサインアウトします。[GitHub アカウント] に、この接続を識別する名前を入力し、[GitHub に接続] を選択します。アプリケーションの GitHub を操作す

ることを CodeDeploy に許可するよう求めるメッセージがウェブページに表示されます。ステップ 2 に進みます。

- 作成済みの接続を使用するには、その名前を [GitHub account] で選択してから [Connect to GitHub] を選択します。ステップ 4 に進みます。
- 別の GitHub アカウントへの接続を作成するには、ウェブブラウザの別のタブで GitHub からサインアウトします。[Connect to a different GitHub account] を選択し、[Connect to GitHub] を選択します。ステップ 2 に進みます。

2. GitHub にサインインするよう求められたら、[Sign in] ページの手順に従います。GitHub のユーザー名、または E メールとパスワードでサインインします。
3. [Authorize application] ページが表示された場合、[Authorize application] を選択します。
4. [Create deployment (デプロイの作成)] ページの、[Repository name (リポジトリの名前)] ボックスに、スラッシュ (/)、リビジョンを含むリポジトリの名前が後に続く、リビジョンを含む GitHub ユーザーまたは組織の名前を入力します。入力する値が不確実な場合:

   a. ウェブブラウザの別のタブで、GitHub dashboard にアクセスします。

   b. [Your repositories] で、ターゲットリポジトリの名前の上にマウスを置きます。GitHub ユーザーまたは組織の名前、スラッシュ (/)、リポジトリの名前の順序でツールヒントが表示されます。[Repository name (リポジトリの名前)] ボックスに、この表示された値を入力します。

   > **Note**
   >
   > [Your repositories] にターゲットリポジトリの名前が表示されない場合、[Search GitHub] ボックスを使用して、ターゲットリポジトリの名前と、GitHub ユーザーまたは組織の名前を検索します。

5. [Commit ID (コミットID)] で、リポジトリのリビジョンを参照するコミット ID を入力します。入力する値が不確実な場合:

   a. ウェブブラウザの別のタブで、GitHub dashboard にアクセスします。

   b. [Your repositories] で、ターゲットコミットを含むリポジトリの名前を選択します。

   c. コミットのリストで、リポジトリのリビジョンを参照するコミット ID を検索してコピーします。通常、この ID は 40 文字で、文字と数字の両方で構成されます (通常はコミット ID の長いバージョンの最初の 10 文字の、コミット ID の短いバージョンを使用しないでください)。

   d. [Commit ID] ボックスにコミット ID を貼り付けます。

Amazon ECS コンピューティングプラットフォームのデプロイの作成 (CLI)

アプリケーションとリビジョンの作成後 (Amazon ECS のデプロイでは、これは AppSpec ファイルです)。

create-deployment コマンドを呼び出し、指定します。

- アプリケーション名。アプリケーション名のリストを表示するには、list-applications コマンドを呼び出します。
- デプロイグループ名。デプロイグループ名のリストを表示するには、list-deployment-groupsコマンドを呼び出します。
- デプロイするリビジョンに関する情報。

  Amazon S3 に格納されているリビジョン。

  - Amazon S3 バケットの名前がリビジョンに含まれています。
  - アップロードされたリビジョンの名前。
  - (オプション) リビジョンの Amazon S3 バージョンのID。(バージョン ID を指定しない場合、CodeDeploy は最新バージョンを使用します。)
  - (オプション) リビジョンの ETag。(ETag が指定されていない場合、CodeDeploy はオブジェクトの検証をスキップします。)

  Amazon S3 にないファイルに保存されているリビジョンの場合、ファイル名とパスが必要です。リビジョンファイルは、YAML または JSON で書かれているため、ほとんどの場合、その拡張子は .json または .yaml です。
- (オプション) デプロイの説明。

リビジョンファイルは Amazon S3 バケットにアップロードされるファイルまたは文字列として指定できます。create-deployment コマンドの一部として使用する場合の各構文は次のようになります。

- Amazon S3 バケット:

  `version`および`eTag`はオプションです。

```
--s3-location bucket=string,key=string,bundleType=JSON|
YAML,version=string,eTag=string
```

- 文字列:

```
--revision '{"revisionType": "String", "string": {"content":"revision-as-string"}}'
```

> **Note**
>
> create-deployment コマンドはファイルからリビジョンをロードできます。詳細については、「ファイルからパラメータをロードする」を参照してください。

AWS Lambda デプロイリビジョンテンプレートについては、「」を参照してくださいAWS Lambda デプロイ用の AppSpec ファイルを追加する。リビジョンの例については、「Lambda AWS デプロイの AppSpec ファイルの例」を参照してください。

デプロイの状態を追跡するには、「CodeDeploy デプロイの詳細を表示する」を参照してください。

AWS Lambda コンピューティングプラットフォームのデプロイを作成する (CLI)

アプリケーションとリビジョンを作成した後 (AWS Lambda デプロイでは、これは AppSpec ファイルです) 。

create-deployment コマンドを呼び出し、指定します。

- アプリケーション名。アプリケーション名のリストを表示するには、list-applications コマンドを呼び出します。
- デプロイグループ名。デプロイグループ名のリストを表示するには、list-deployment-groupsコマンドを呼び出します。
- デプロイするリビジョンに関する情報。

  Amazon S3 に格納されているリビジョン。

  - Amazon S3 バケットの名前がリビジョンに含まれています。
  - アップロードされたリビジョンの名前。
  - (オプション) リビジョンの Amazon S3 バージョンのID。(バージョン ID を指定しない場合、CodeDeploy は最新バージョンを使用します。)

- (オプション) リビジョンの ETag。(ETag が指定されていない場合、CodeDeploy はオブジェクトの検証をスキップします。)

  Amazon S3 にないファイルに保存されているリビジョンの場合、ファイル名とパスが必要です。リビジョンファイルは、YAML または JSON で書かれているため、ほとんどの場合、その拡張子は .json または .yaml です。
- (オプション) 使用するデプロイ設定の名前。デプロイ設定のリストを表示するには、list-deployment-configs コマンドを呼び出します。(指定されない場合、CodeDeploy は、特定のデフォルトのデプロイ設定を使用します。)
- (オプション) デプロイの説明。

リビジョンファイルは Amazon S3 バケットにアップロードされるファイルまたは文字列として指定できます。create-deployment コマンドの一部として使用する場合の各構文は次のようになります。

- Amazon S3 バケット:

  `version`および`eTag`はオプションです。

```
--s3-location bucket=string,key=string,bundleType=JSON|
YAML,version=string,eTag=string
```

- 文字列:

```
--revision '{"revisionType": "String", "string": {"content":"revision-as-string"}}'
```

> **Note**
>
> create-deployment コマンドはファイルからリビジョンをロードできます。詳細については、「ファイルからパラメータをロードする」を参照してください。

AWS Lambda デプロイリビジョンテンプレートについては、「」を参照してくださいAWS Lambda デプロイ用の AppSpec ファイルを追加する。リビジョンの例については、「Lambda AWS デプロイの AppSpec ファイルの例」を参照してください。

デプロイの状態を追跡するには、「CodeDeploy デプロイの詳細を表示する」を参照してください。

EC2/ オンプレミスコンピューティングプラットフォームのデプロイ作成 (CLI)

を使用して EC2/オンプレミスコンピューティングプラットフォームにリビジョンを AWS CLI デプロイするには:

1. インスタンスを準備し、アプリケーションを作成して、リビジョンをプッシュした後、次のいずれかを実行します。
   - Amazon S3 バケットからリビジョンをデプロイする場合は、そのままステップ 2 に進みます。
   - GitHub リポジトリからリビジョンをデプロイする場合は、まず「CodeDeploy アプリケーションを GitHub リポジトリに接続します。」のステップを完了してからステップ 2 に戻ります。
2. create-deployment コマンドを呼び出し、指定します。
   - `--application-name`: アプリケーション名。アプリケーション名のリストを表示するには、list-applications コマンドを呼び出します。
   - `--deployment-group-name`: Amazon EC2 デプロイグループ名。デプロイグループ名のリストを表示するには、list-deployment-groupsコマンドを呼び出します。
   - `--revision`: デプロイするリビジョンに関する情報。

     Amazon S3 に格納されているリビジョン。
     - `s3Location`: リビジョンを含む Amazon S3 バケットの名前がリビジョンに含まれています。
     - `s3Location --> key`: アップロードされたリビジョンの名前。
     - `s3Location --> bundleType`: アップロードされたリビジョンの名前とファイル形式。

       > **Note**
       >
       > tar および圧縮 tar アーカイブファイル形式 (.tar および .tar.gz) は、Windows Server インスタンスではサポートされていません。

     - `s3Location --> version`: (オプション) リビジョンの Amazon S3 バージョン ID。(バージョン ID を指定しない場合、CodeDeploy は最新バージョンを使用します。)
     - `s3Location --> eTag`: (オプション) リビジョンの ETag。(ETag が指定されていない場合、CodeDeploy はオブジェクトの検証をスキップします。)

     GitHub で格納されたリビジョンの場合。

  - `gitHubLocation --> repository`: スラッシュ (/)、リポジトリの名前が後に続く、リポジトリに割り当てられたリビジョンを含む GitHub ユーザーまたは組織の名前。
  - `gitHubLocation --> commitId`: リビジョンのコミット ID。
- `--deployment-config-name`: (オプション) 使用するデプロイ設定の名前。デプロイ設定のリストを表示するには、list-deployment-configs コマンドを呼び出します。(指定されない場合、CodeDeploy は特定のデフォルトのデプロイ設定を使用します。)
- `--ignore-application-stop-failures | --no-ignore-application-stop-failures`: (オプション) `BeforeInstall` デプロイライフサイクルのイベントを続行してインスタンスへのデプロイを続行するかどうか。たとえ `ApplicationStop` デプロイライフサイクルイベントが失敗してもです。
- `--description`: (オプション) デプロイの説明。
- `--file-exists-behavior`: (オプション) デプロイプロセスの一環として、CodeDeploy エージェントは、前回のデプロイでインストールされたすべてのファイルを各インスタンスから削除します。以前のデプロイの一部ではなかったファイルがターゲットデプロイの場所に表示されるとどうなるかを選択します。
- `--target-instances`: Blue/Green デプロイの場合、1 つ以上の Amazon EC2 Auto Scaling グループの名前、または、Amazon EC2 インスタンスを識別するのに使用するタグフィルタキー、型、および値を含む、Blue/Green デプロイの置き換え先環境に属するインスタンスに関する情報です。

> **Note**
>
> Amazon S3 内のリビジョンに関する情報をコマンドラインで直接指定するには、create-deployment 呼び出しの一部としてこの構文を使用します。(`version` および `eTag` 設定はオプションです)。
>
> ```
> --s3-location bucket=string,key=string,bundleType=tar|tgz|
> zip,version=string,eTag=string
> ```
>
> コマンドラインで、GitHub のリビジョンについての情報を直接呼び出す create-deployment の一部としてこの構文を使用します。
>
> ```
> --github-location repository=string,commitId=string
> ```

すでにプッシュされているリビジョンについての情報を入手するには、list-application-revisions コマンドを呼び出します。

デプロイの状態を追跡するには、「CodeDeploy デプロイの詳細を表示する」を参照してください。

create-deployment コマンドリファレンス

コマンドでは、次のコマンド構造とオプションを使用できますcreate-deployment。詳細については、「AWS CLI コマンドリファレンス」の「create-deployment」を参照してください。

```
create-deployment
--application-name <value>
[--deployment-group-name <value>]
[--revision <value>]
[--deployment-config-name <value>]
[--description <value>]
[--ignore-application-stop-failures | --no-ignore-application-stop-failures]
[--target-instances <value>]
[--auto-rollback-configuration <value>]
[--update-outdated-instances-only | --no-update-outdated-instances-only]
[--file-exists-behavior <value>]
[--s3-location <value>]
[--github-location <value>]
[--cli-input-json <value>]
[--generate-cli-skeleton <value>]
```

CodeDeploy アプリケーションを GitHub リポジトリに接続します。

を使用して GitHub リポジトリからアプリケーションを初めてデプロイする前に AWS CLI、まず GitHub アカウントに代わって GitHub とやり取りするための CodeDeploy アクセス許可を付与する必要があります。CodeDeploy コンソールを使用して、このステップをアプリケーションごとに一度実行する必要があります。

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

> **Note**
>
> 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. [Applications] (アプリケーション) を選択します。
3. [アプリケーション] から、GitHub ユーザーアカウントにリンクするアプリケーションを選択し、[アプリケーションをデプロイする] を選択します。

> **Note**
>
> デプロイを作成していません。これは現在、GitHub ユーザーアカウントに代わって GitHub を操作するためにアクセス権限を CodeDeploy に付与する唯一の方法です。

4. [Repository type] の横の [My application revision is stored in GitHub] を選択します。
5. [Connect to GitHub] (GitHub に接続) を選択します。

> **Note**
>
> [Connect to a different GitHub account] リンクが表示されている場合:
> すでに CodeDeploy が GitHub とやりとりすることを、アプリケーションの別の GitHub アカウントに代わって許可している場合があります。
> CodeDeploy にリンクされているすべてのアプリケーションに対してサインインしている GitHub アカウントに代わって、GitHub を操作する CodeDeploy の承認を取り消した可能性があります。
> 詳細については、「CodeDeploy のアプリケーションを使用した GitHub の認証」を参照してください。

6. GitHub にまだサインインしていない場合は、[Sign in] ページの手順に従います。
7. [Authorize application] ページで、[Authorize application] を選択します。
8. CodeDeploy にアクセス許可が付与されたら、[キャンセル] を選択して、「EC2/ オンプレミスコンピューティングプラットフォームのデプロイ作成 (CLI)」の手順に進んでください。

を使用して Amazon ECS ブルー/グリーンデプロイを作成する CloudFormation

を使用して AWS CloudFormation 、CodeDeploy を通じて Amazon ECS ブルー/グリーンデプロイを管理できます。デプロイを生成するには、Green と Blue のリソースを定義し、 CloudFormationで使用するトラフィックルーティングと安定化の設定を指定します。このトピックでは、CodeDeploy によって管理される Amazon ECS Blue/Green デプロイと CloudFormationによって管理されるデプロイの違いについて説明します。

CloudFormation を使用して Amazon ECS ブルー/グリーンデプロイを管理するstep-by-stepについては、 AWS CloudFormation ユーザーガイドの「 を使用した CodeDeploy による ECS ブルー/グリーンデプロイの自動化 AWS CloudFormation」を参照してください。

> **Note**
>
> を使用した Amazon ECS ブルー/グリーンデプロイの管理 CloudFormation は、アジアパシフィック (大阪) リージョンでは利用できません。

CodeDeploy と を使用した Amazon ECS ブルー/グリーンデプロイの違い CloudFormation

CloudFormation スタックテンプレートは、Amazon ECS タスク関連のリソースとインフラストラクチャ、およびデプロイの設定オプションをモデル化します。したがって、標準の Amazon ECS ブルー/グリーンデプロイと、 を通じて作成されるブルー/グリーンデプロイには違いがあります CloudFormation。

標準の Amazon ECS Blue/Green デプロイとは異なり、以下のモデル作成や手動作成は行いません。

- デプロイする内容を一意に表す名前を指定しても、 AWS CodeDeploy アプリケーションは作成されません。
- AWS CodeDeploy デプロイグループを作成しません。
- アプリケーション仕様ファイル (AppSpec ファイル)を指定しない。通常、AppSpec ファイルで管理される情報 (加重設定オプションやライフサイクルイベントなど) は、`AWS::CodeDeploy::BlueGreen` フックによって管理されます。

この表は、デプロイタイプ間の高レベルのワークフローの違いをまとめたものです。

関数	標準 Blue/Green デプロイ	によるブルー/グリーンデプロイ CloudFormation
Amazon ECS クラスター、Amazon ECS サービス、Application Load Balancer またはNetwork Load Balancer、本稼働リスナー、テストリスナー、および 2 つのターゲットグループを指定します。	これらのリソースを指定する CodeDeploy デプロイグループを作成します。	これらのリソースをモデル化する CloudFormation テンプレートを作成します。
デプロイする変更を指定します。	CodeDeploy でアプリケーションを作成します。	コンテナイメージを指定する CloudFormation テンプレートを作成します。
Amazon ECS タスク定義、コンテナ名、コンテナポートを指定します。	これらのリソースを指定する AppSpec ファイルを作成します。	これらのリソースをモデル化する CloudFormation テンプレートを作成します。
デプロイトラフィックシフトオプションとライフサイクルイベントフックを指定します。	これらのオプションを指定する AppSpec ファイルを作成します。	`AWS::CodeDeploy::BlueGreen` フックパラメータを使用してこれらのオプションを指定する CloudFormation テンプレートを作成します。
CloudWatch アラーム。	ロールバックをトリガーする CloudWatch アラームを作成します。	ロールバックをトリガーする CloudWatch アラームを CloudFormation スタックレベルで設定します。
ロールバック/再デプロイ。	ロールバックおよび再デプロイのオプションを指定します。	スタックの更新をキャンセルします CloudFormation。

による Amazon ECS ブルー/グリーンデプロイのモニタリング CloudFormation

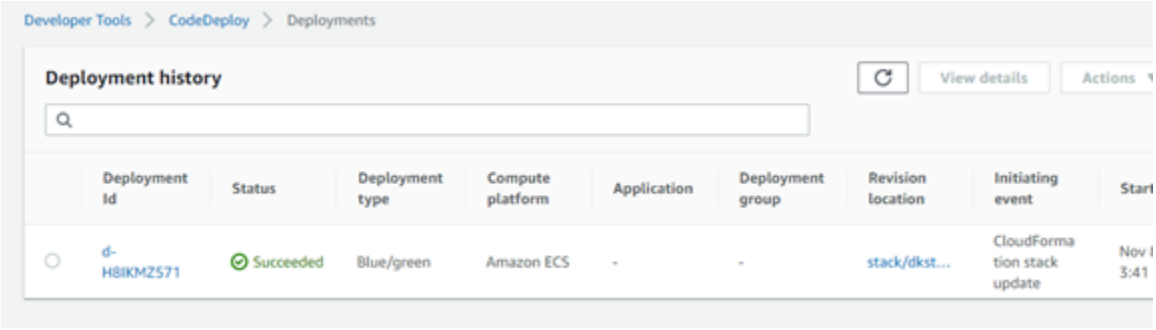
ブルー/グリーンデプロイは、 CloudFormation と CodeDeploy を使用してモニタリングできます。によるモニタリングの詳細については CloudFormation、「 AWS CloudFormation ユーザーガイド」の「 でのブルー/グリーンイベントのモニタリング CloudFormation」を参照してください。

CodeDeploy で Blue/Green デプロイのデプロイステータスを表示するには

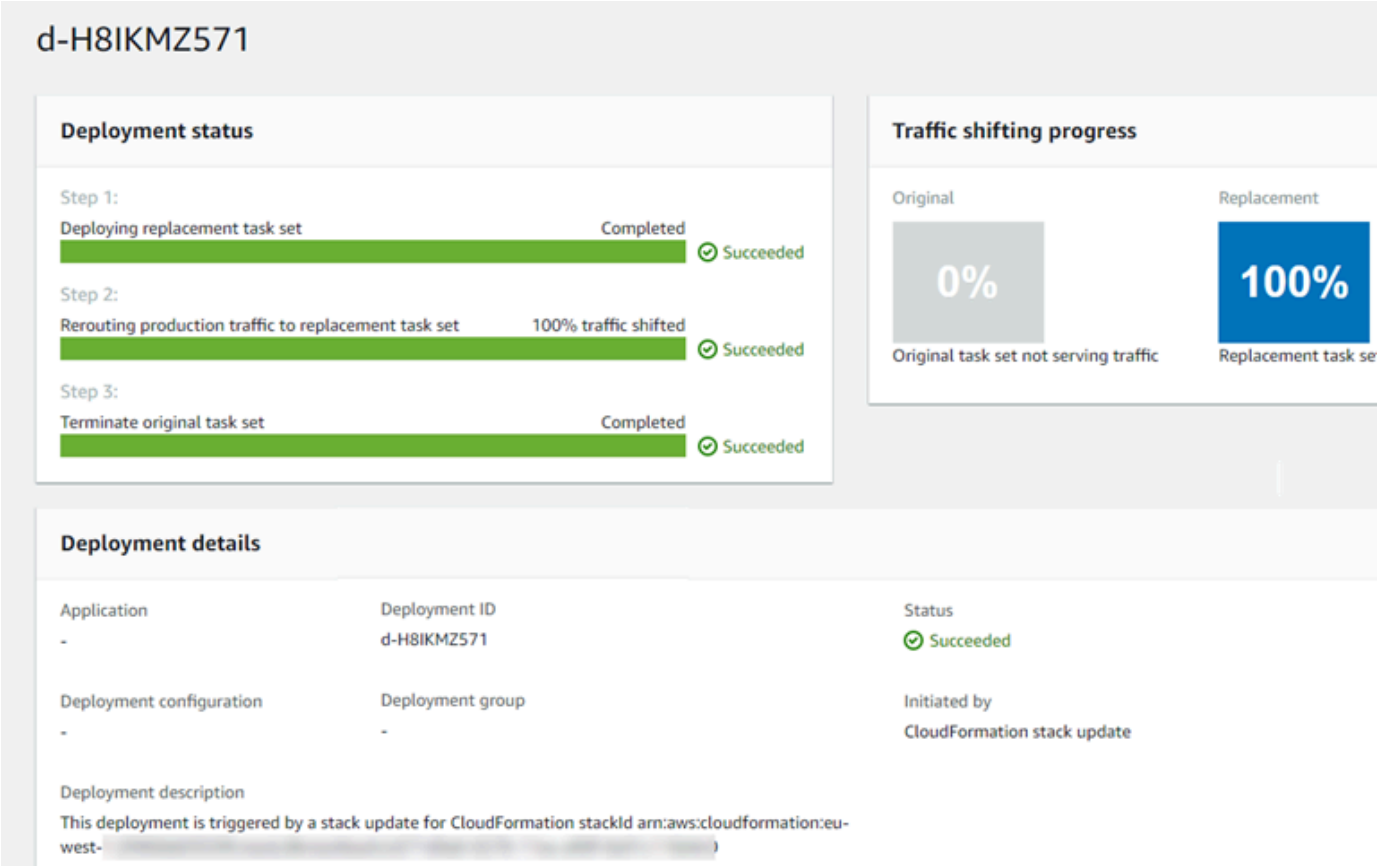
1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. デプロイでは、 CloudFormation スタックの更新によってトリガーされたデプロイが表示されます。デプロイを選択して、[デプロイ履歴] を表示します。



3. デプロイを選択して、トラフィックシフトステータスを表示します。アプリケーションおよびデプロイグループは作成されないことに注意してください。



4. デプロイのロールバックまたは停止には、次のことが適用されます。

   - 成功したデプロイは CodeDeploy に表示され、 CloudFormationによってデプロイが開始されたことが示されます。
   - デプロイを停止してロールバックする場合は、スタックの更新をキャンセルする必要があります CloudFormation。

CodeDeploy デプロイの詳細を表示する

CodeDeploy コンソール、 AWS CLI、または CodeDeploy APIs を使用して、 AWS アカウントに関連付けられたデプロイの詳細を表示できます。

> **Note**
>
> インスタンスの EC2　オンプレミスデプロイログは以下の場所で確認できます。
>
> - Amazon Linux、RHEL、Ubuntu Server:`/opt/codedeploy-agent/deployment-root/deployment-logs/codedeploy-agent-deployments.log`

- Windows Server の場合: C:\ProgramData\Amazon\CodeDeploy<DEPLOYMENT-GROUP-ID><DEPLOYMENT-ID>\logs\scripts.log

詳細については、「ログファイルの分析によるインスタンスでのデプロイの失敗の調査」を参照してください。

トピック

- デプロイの詳細の表示 (コンソール)
- デプロイの詳細の表示 (CLI)

デプロイの詳細の表示 (コンソール)

CodeDeploy コンソールを使用してデプロイの詳細を表示するには:

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで [デプロイ] を展開し、[アプリケーション] を選択します。

   > **Note**
   >
   > エントリが表示されない場合は、正しいリージョンが選択されていることを確認しましょう。ナビゲーションバーのリージョンセレクターで、「AWS 全般のリファレンス」の「リージョンとエンドポイント」に一覧表示されているいずれかのリージョンを選択します。CodeDeploy は、これらのリージョンでのみサポートされています。

3. 1 つのデプロイの詳細を表示するには、[デプロイ履歴] でデプロイ ID を選択するか、デプロイ ID の横にあるボタンを選択して、[表示] を選択します。

デプロイの詳細の表示 (CLI)

を使用してデプロイの詳細 AWS CLI を表示するには、 `get-deployment` コマンドまたは `batch-get-deployments` コマンドを呼び出します。`list-deployments` コマンドを呼び出して、`get-deployment` コマンドおよび `batch-get-deployments` コマンドへの入力として使用する一意のデプロイ ID のリストを取得できます。

1 つのデプロイの詳細を表示するには、[デプロイの取得] コマンドを呼び出し、一意のデプロイ ID を特定します。デプロイ ID を取得するには、[リストデプロイグループ] コマンドを呼び出します。

複数のデプロイの詳細を表示するには、[デプロイの取得バッチ] コマンドを呼び出し、複数の一意のデプロイ ID を特定します。デプロイ ID を取得するには、[リストデプロイ] コマンドを呼び出します。

デプロイ ID の一覧を表示するには、[リストデプロイ] コマンドを呼び出し、指定します。

- デプロイに関連付けられたアプリケーションの名前。アプリケーション名のリストを表示するには、[リストアプリケーション] コマンドを呼び出します。
- デプロイに関連付けられたデプロイグループの名前。デプロイグループ名のリストを表示するには、[リストデプロイグループ] コマンドを呼び出します。
- オプションで、デプロイの詳細をデプロイの状態別に含めるかどうか (指定しない場合、一致するすべてのデプロイが、デプロイの状態にかかわらず表示されます)。
- オプションで、デプロイの作成開始または終了時間別、あるいはその両方を基準にデプロイの詳細を含めるかどうか (指定しない場合、一致するすべてのデプロイが、作成時間にかかわらず表示されます)。

CodeDeploy EC2/オンプレミスデプロイのログデータの表示

CodeDeploy デプロイで作成されたログデータを表示するには、CloudWatch コンソールに集約データを表示するように Amazon CloudWatch エージェントを設定するか、個別のインスタンスにログインしてログファイルを確認します。

> **Note**
>
> ログは、 AWS Lambda または Amazon ECS デプロイではサポートされていません。EC2 オンプレミスデプロイのみに作成できます。

トピック

- Amazon CloudWatch コンソールでのログファイルのデータの表示
- インスタンスでのログファイルの表示

Amazon CloudWatch コンソールでのログファイルのデータの表示

インスタンスに Amazon CloudWatch エージェントがインストールされていると、そのインスタンスに対するすべてのデプロイのデータは、CloudWatch コンソールで表示できるようになります。わかりやすいように、インスタンス別に表示するのではなく、CloudWatch を使用してログファイルを集中的にモニタリングすることをお勧めします。詳細については、「CodeDeploy エージェントログを CloudWatch に送信する」を参照してください。

インスタンスでのログファイルの表示

個別のインスタンスのデプロイログデータを表示するには、インスタンスにサインインして、エラーや他のデプロイイベントに関する情報を参照します。

トピック

- Amazon Linux、RHEL、および Ubuntu サーバーインスタンスのデプロイログファイルを表示するには
- Windows Server インスタンスのデプロイログファイルを表示するには

Amazon Linux、RHEL、および Ubuntu サーバーインスタンスのデプロイログファイルを表示するには

Amazon Linux、RHEL、および Ubuntu サーバーの各インスタンスのデプロイログは、次の場所に保存されています。

`/opt/codedeploy-agent/deployment-root/deployment-logs/codedeploy-agent-deployments.log`

Amazon Linux、RHEL、および Ubuntu サーバーの各インスタンスのデプロイログを表示または分析するには、インスタンスにサインインし、次のコマンドを入力して CodeDeploy エージェントのログファイルを開きます。

```
less /var/log/aws/codedeploy-agent/codedeploy-agent.log
```

バージョン 2.0.x 以降では、エージェントログは毎日、またはファイルが 64 MB (設定可能) に達したときにローテーションされます。ローテーションされたアーカイブには名前が付けられ codedeploy-agent.YYYYMMDD.log (同じ日のローテーションには数値サフィックスが付きます) 、7 日後にプルーニングされます。スクリプトログ (scripts.log) のサイズは 64 MB で、最大 8 つのアーカイブファイル (設定可能) でローテーションされます。

エラーメッセージのログファイルを参照するには、次のコマンドを入力します。

コマンド	結果
& ERROR	ログファイルでエラーメッセージのみを表示します。**ERROR** という単語の前後に 1 つのスペースを使用します。
/ ERROR	次のエラーメッセージを検索します。[1]
? ERROR	前のエラーメッセージを検索します。[2] 単語の前後に 1 つのスペースを入力します **ERROR** 。
G	ログファイルの末尾に移動します。
g	ログファイルの先頭に移動します。
q	ログファイルを終了します。
h	追加のコマンドについて参照します。

[1] **/ ERROR** と入力してから、次のエラーメッセージを検索するには、**n** と入力します。前のエラーメッセージを検索するには、**N** と入力します。

[2] **? ERROR** と入力してから、次のエラーメッセージを検索するには **n** か前のエラーメッセージを検索するには **N** と入力します。

また、次のコマンドを入力して CodeDeploy スクリプトのログファイルを開くこともできます。

```
less /opt/codedeploy-agent/deployment-root/deployment-group-ID/deployment-ID/logs/
scripts.log
```

エラーメッセージのログファイルを参照するには、次のコマンドを入力します。

コマンド	結果
&stderr	ログファイルでエラーメッセージのみを表示します。
/stderr	次のエラーメッセージを検索します。[1]
?stderr	前のエラーメッセージを検索します。[2]
G	ログファイルの末尾に移動します。
g	ログファイルの先頭に移動します。
q	ログファイルを終了します。
h	追加のコマンドについて参照します。

[1] **/stderr** と入力してから、次のエラーメッセージを前方に検索するには **n** と入力します。前のエラーメッセージを後方に検索するには、 **N** と入力します。

[2] **?stderr** と入力してから、次のエラーメッセージを後方に検索するには、**n** と入力します。前のエラーメッセージを前方に検索するには、**N** と入力します。

Windows Server インスタンスのデプロイログファイルを表示するには

CodeDeploy エージェントのログファイル: Windows サーバーインスタンスで CodeDeploy エージェントのログファイルは次の場所に保存されています。

バージョン 2.0.x 以降の場合:

```
C:\ProgramData\Amazon\CodeDeploy\log\codedeploy-agent.log
```

バージョン 1.8.x 以前の場合:

```
C:\ProgramData\Amazon\CodeDeploy\log\codedeploy-agent-log.txt
```

Windows Serve インスタンスで CodeDeploy エージェントのログファイルを表示または分析するには、インスタンスにサインインし、次のコマンドを入力してファイルを開きます。

バージョン 2.0.x 以降の場合:

```
notepad C:\ProgramData\Amazon\CodeDeploy\log\codedeploy-agent.log
```

バージョン 1.8.x 以前の場合:

```
notepad C:\ProgramData\Amazon\CodeDeploy\log\codedeploy-agent-log.txt
```

ログファイルでエラーメッセージを参照するには、Ctrl+F キーを押し、**ERROR** と入力してから、Enter キーを押して最初のエラーを見つけます。

CodeDeploy スクリプトログファイル: Windows サーバーインスタンスのデプロイログは次の場所に保存されています。

C:\ProgramData\Amazon\CodeDeploy*deployment-group-id**deployment-id*\logs\scripts.log

説明は以下のとおりです。

- *deployment-group-id* は examplebf3a9c7a-7c19-4657-8684-b0c68d0cd3c4 などの文字列
- *deployment-id* は、d-12EXAMPLE などの識別子

また、次のコマンドを入力して CodeDeploy スクリプトのログファイルを開くこともできます。

```
notepad C:\ProgramData\Amazon\CodeDeploy\deployment-group-ID\deployment-ID\logs
\scripts.log
```

バージョン 2.0.x 以降では、エージェントログは毎日、またはファイルが 64 MB (設定可能) に達したときにローテーションされます。スクリプトログのサイズは 64 MB で、最大 8 つのアーカイブファイル (設定可能) でローテーションされます。

ログファイルでエラーメッセージを参照するには、Ctrl+F キーを押し、**stderr** と入力してから、Enter キーを押して最初のエラーを見つけます。

CodeDeploy でデプロイの停止

CodeDeploy コンソール、 AWS CLI、または CodeDeploy APIs を使用して、 AWS アカウントに関連付けられたデプロイを停止できます。

> **Warning**
>
> EC2 オンプレミス のデプロイを停止すると、デプロイグループのインスタンスの一部またはすべてを未確定のデプロイメントの状態のまま残すことができます。詳細については、「停止、失敗したデプロイ」を参照してください。

デプロイを停止することも、デプロイを停止してロールバックすることもできます。

- デプロイ (コンソール) の停止
- デプロイ (CLI) の停止

> **Note**
>
> デプロイが によるブルー/グリーンデプロイの場合 CloudFormation、CodeDeploy コンソールでこのタスクを実行することはできません。 CloudFormation コンソールに移動して、このタスクを実行します。

デプロイ (コンソール) の停止

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで [デプロイ] を展開し、[アプリケーション] を選択します。

   > **Note**
   >
   > エントリが表示されない場合は、正しいリージョンが選択されていることを確認しましょう。ナビゲーションバーのリージョンセレクターで、「AWS 全般のリファレンス」の「リージョンとエンドポイント」に一覧表示されているいずれかのリージョンを選択します。CodeDeploy は、これらのリージョンでのみサポートされています。

3. 停止するデプロイを選択し、以下のいずれかの操作を行います。

1. [デプロイの停止] を選択し、ロールバックなしでデプロイを停止します。
2. デプロイを停止してロールバックするには、[Stop and roll back deployment (デプロイを停止してロールバック)] を選択します。

詳細については、「CodeDeploy を使用した再デプロイおよびデプロイのロールバック」を参照してください。

> **Note**
>
> [デプロイの停止] および [Stop and roll back deployment (デプロイを停止してロールバック)] が使用不可の場合、そのデプロイは停止できないところまで進行しています。

デプロイ (CLI) の停止

デプロイ ID を指定して、[停止デプロイ] コマンドを呼び出します。デプロイ ID の一覧を表示するには、[リストデプロイ] コマンドを呼び出します。

CodeDeploy を使用した再デプロイおよびデプロイのロールバック

CodeDeploy は、以前にデプロイされたアプリケーションのリビジョンを新しいデプロイとして再デプロイすることによって、デプロイをロールバックします。これらのロールバックされたデプロイは、前のデプロイのバージョンを復元するのではなく、新しいデプロイ ID を使用する技術的に新しいデプロイです。

デプロイは、自動または手動でロールバックできます。

トピック

- 自動ロールバック
- 手動ロールバック
- ロールバックおよび再デプロイのワークフロー
- 既存のコンテンツでのロールバック動作

自動ロールバック

デプロイが失敗した場合、または指定した監視しきい値に達した場合、自動的にロールバックするように、デプロイグループまたはデプロイを設定できます。この場合、アプリケーションリビジョンの最後の既知の正常なバージョンがデプロイされます。自動ロールバックは、アプリケーションを作成するとき、またはデプロイグループを作成または更新するときに設定します。

新しいデプロイを作成するとき、デプロイグループに指定された自動ロールバック設定をオーバーライドすることもできます。

> **Note**
>
> デプロイが自動的にロールバックされるときには、Amazon Simple Notification Service を使用して通知を受け取ることができます。詳細については、「Monitoring Deployments with Amazon SNS Event Notifications」を参照してください。

自動ロールバックの設定の詳細については、「デプロイグループの詳細オプションの設定」を参照してください。

手動ロールバック

自動ロールバックをセットアップしていない場合は、以前にデプロイされたアプリケーションリビジョンを使用する新しいデプロイを作成し、リビジョンを再デプロイする手順に従うことによって、デプロイを手動でロールバックすることができます。アプリケーションが不明な状態になった場合、これを行う場合があります。トラブルシューティングに多くの時間を費やすのではなく、アプリケーションを既知の動作状態に再デプロイすることができます。詳細については、「CodeDeploy でデプロイを作成する」を参照してください。

> **Note**
>
> デプロイグループからインスタンスを削除する場合、CodeDeploy はそのインスタンスにすでにインストールされているものはアンインストールしません。

ロールバックおよび再デプロイのワークフロー

自動ロールバックが開始された場合、または再デプロイまたは手動ロールバックを手動で開始した場合、CodeDeploy は、まず、最後に正常にインストールされたすべてのファイルを各参加インスタン

スから削除しようとします。CodeDeploy は、クリーンアップファイルをチェックすることでこれを行います :

`/opt/codedeploy-agent/deployment-root/deployment-instructions/`*`deployment-group-ID`*`-cleanup` ファイル (Amazon Linux、Ubuntu Server、および RHEL インスタンス用)

`C:\ProgramData\Amazon\CodeDeploy\deployment-instructions\`*`deployment-group-ID`*`-cleanup` ファイル (Windows Server インスタンス用)

存在する場合、CodeDeploy は、クリーンアップファイルを使用して、新しいデプロイを開始する前にリストされたすべてのファイルをインスタンスから削除します。

例えば、最初の 2 つのテキストファイルおよび 2 つのスクリプトファイルは、Windows Server を実行している Amazon EC2 インスタンスにデプロイ済みであり、スクリプトによってデプロイライフサイクルイベント中にさらに 2 つのテキストファイルが作成されました :

```
c:\temp\a.txt (previously deployed by CodeDeploy)
c:\temp\b.txt (previously deployed by CodeDeploy)
c:\temp\c.bat (previously deployed by CodeDeploy)
c:\temp\d.bat (previously deployed by CodeDeploy)
c:\temp\e.txt (previously created by c.bat)
c:\temp\f.txt (previously created by d.bat)
```

クリーンアップファイルは、最初の 2 つのテキストファイルおよび 2 つのスクリプトファイルのみが表示されます。

```
c:\temp\a.txt
c:\temp\b.txt
c:\temp\c.bat
c:\temp\d.bat
```

新しいデプロイの前に、CodeDeploy は最初の 2 つのテキストファイルおよび 2 つのスクリプトファイルのみを削除し、最後の 2 つのテキストファイルはそのまま残します :

```
c:\temp\a.txt will be removed
c:\temp\b.txt will be removed
c:\temp\c.bat will be removed
c:\temp\d.bat will be removed
c:\temp\e.txt will remain
c:\temp\f.txt will remain
```

このプロセスの一部として、CodeDeploy は、手動または自動ロールバックであっても、その後の再デプロイの間に以前のデプロイのスクリプトによって実行されるアクションを元に戻したり、一致させたりしません。例えば、`c.bat` および `d.bat` のファイルに、`e.txt` および `f.txt` のファイルが既に存在している場合、再作成しないロジックを含んでいる場合は、`e.txt` および `f.txt` の古いバージョンはそのまま残り、それは CodeDeploy デプロイがその後のデプロイで `c.bat` および `d.bat` を実行しても残ったままです。`c.bat` および `d.bat` にロジックを追加して、新しいバージョンを作成する前に `e.txt` および `f.txt` の古いバージョンを常にチェックして削除することができます。

既存のコンテンツでのロールバック動作

デプロイプロセスの一環として、CodeDeploy エージェントは、最新のデプロイがインストールしたすべてのファイルを各インスタンスから削除します。以前のデプロイに含まれていないファイルがターゲットデプロイの場所に表示された場合は、次のデプロイ時に CodeDeploy がそれらに対して実行する内容を選択できます。

- [Fail the deployment] — エラーが報告され、デプロイのステータスが「Failed (失敗)」に変更されます。
- [コンテンツの上書き] — アプリケーションリビジョンのファイルのバージョンにより、インスタンスの既存のファイルのバージョンが置き換えられます。
- [コンテンツの保持] — デプロイ先のファイルは保持され、アプリケーションリビジョンのバージョンはインスタンスにコピーされません。

この動作は、デプロイの作成時に選択できます。コンソールでデプロイを作成する場合は、「EC2/オンプレミスコンピューティングプラットフォームのデプロイ作成 (コンソール)」を参照してください。を使用してデプロイを作成する場合は AWS CLI、「」を参照してくださいEC2/ オンプレミスコンピューティングプラットフォームのデプロイ作成 (CLI)。

ファイルを保持して次回のデプロイの一部とすることを選択すると、そのファイルはアプリケーションリビジョンパッケージに追加する必要がなくなります。たとえば、デプロイに必要ながアプリケーションリビジョンバンドルに追加されていないファイルをインスタンスに直接アップロードできます。または、アプリケーションが既に本番稼働環境にあっても CodeDeploy を初めて使用してデプロイする場合は、ファイルをインスタンスにアップロードできます。

ロールバックでは、デプロイの失敗が原因で前回の成功したデプロイのアプリケーションリビジョンが再デプロイされますが、その前回の成功したデプロイのコンテンツ処理オプションがロールバックデプロイに適用されます。

ただし、失敗したデプロイの設定がファイルを保持せずに上書きするようになっていた場合、ロールバックは予期しない結果になる可能性があります。特に、保持しようとしていたファイルが、失敗したデプロイによって削除される可能性があります。ロールバックデプロイを実行したときに、ファイルは、ロールバック時にインスタンス上ではなくデプロイが実行されます。

次の例では 3 つのデプロイがあります。失敗したデプロイ 2 で上書き (削除) されたファイルは、デプロイ 3 でアプリケーションリビジョン 1 が再度デプロイされたときには使用不能 (保持不能) です。

デプロイメント	アプリケーションリビジョン	コンテンツ上書きオプション	デプロイのステータス	動作と結果
デプロイ 1	アプリケーションリビジョン 1	保持	成功	CodeDeploy は、前回のデプロイでデプロイされなかったファイルをデプロイ先で検出します。これらのファイルは、現在のデプロイの一部とするために意図的に置かれている場合があります。これらのファイルは今回のデプロイパッケージの一部として保持および記録されます。
デプロイ 2	アプリケーションリビジョン 2	上書き	失敗	デプロイプロセス中に、CodeDeploy は前回の成功したデプロイに含

デプロイメント	アプリケーションリビジョン	コンテンツ上書きオプション	デプロイのステータス	動作と結果
				まれているすべてのファイルを削除します。これには、デプロイ 1 で保持されたファイルも含まれます。 ただし、デプロイが失敗する原因は、これとは無関係です。

デプロイメント	アプリケーションリビジョン	コンテンツ上書きオプション	デプロイのステータス	動作と結果
デプロイ 3	アプリケーションリビジョン 1	保持		デプロイまたはデプロイグループに対して自動ロールバックが有効になっているため、CodeDeploy は正常であることが判明している前回のアプリケーションリビジョン (アプリケーションリビジョン 1) をデプロイします。 ただし、デプロイ 1 で保持するファイルは、デプロイ 2 が失敗する前に削除され、 では取得できません AWS CodeDeploy。これらのファイルは、アプリケーションリビジョン 1 に必要であれば、自分でインスタンスに追加できます。または、新しいアプリケーション

デプロイメント	アプリケーションリビジョン	コンテンツ上書きオプション	デプロイのステータス	動作と結果
				リビジョンを作成できます。

別の AWS アカウントにアプリケーションをデプロイする

通常、組織には、さまざまな目的で使用する複数の AWS アカウントがあります (例えば、1 つはシステム管理タスク用、もう 1 つは開発、テスト、本番稼働用タスク用、もう 1 つは開発、テスト環境に関連付けられ、もう 1 つは本番稼働用環境に関連付けられています) 。

異なるアカウントで関連作業を実行できる場合でも、CodeDeploy デプロイグループとデプロイ先の Amazon EC2 インスタンスは、これらを作成したアカウントに厳密に関連付けられています。たとえば、1 つのアカウントで起動したインスタンスを別のデプロイグループに追加することはできません。

開発 AWS アカウントと本番稼働用アカウントの 2 つのアカウントがあるとします。主に開発用アカウントで作業しますが、認証情報のフルセットまたは開発用アカウントからのサインアウトや本番稼働用アカウントへのサインインなしで、本番稼働用アカウントでデプロイの開始を可能にします。

クロスアカウント設定ステップに従うと、組織の別のアカウントに属するデプロイを開始できます。その他のアカウントの完全な認証情報セットは必要ありません。これは、そのアカウントへの一時的アクセスを許可する AWS Security Token Service (AWS STS) が提供する機能を利用して一部行われます。

ステップ 1: いずれかのアカウントで S3 バケットを作成する

開発用アカウントまたは本番稼働用アカウントで以下を行います。

- まだそうしていない場合は、本稼働用アカウントのアプリケーションリビジョンが保存される Amazon S3 バケットを作成します。詳細については、「Amazon S3 のバケットの作成」を参照してください。同じファイルを開発用アカウントでテスト、確認した本稼働環境にデプロイして、同じバケットとアプリケーションリビジョンを両方のアカウントに使用することもできます。

ステップ 2: Amazon S3 バケットへのアクセス許可を本番稼働用アカウントの インスタンスプロファイルに付与する

ステップ 1 で作成した Amazon S3 バケットが本番稼働用アカウントにある場合、このステップは必要ありません。後で引き受けるロールは、本番稼働用アカウントにもあるため、このバケットへのアクセス権限をすでに持っています。

開発用アカウントで Amazon S3 バケットを作成する場合は、以下を実行します。

- 本番稼働用アカウントで、IAM インスタンスプロファイルを作成します。詳細については、「ステップ 4: Amazon EC2 インスタンス用の IAM インスタンスプロファイルを作成する」を参照してください。

  > **Note**
  >
  > この IAM インスタンスプロファイルの ARN を書き留めます。次に作成するクロスバケットポリシーにそれを追加する必要があります。

- 開発用アカウントで作成した Amazon S3 バケットへのアクセス権限を、本番稼働用アカウントで先ほど作成した IAM インスタンスプロファイルに付与します。詳細については、「例 2: バケット所有者がクロスアカウントのバケットのアクセス許可を付与する」を参照してください。

  クロスアカウントのバケットのアクセス許可を付与するプロセスを完了するにあたり、次の点に注意してください。

  - サンプルチュートリアルでは、アカウント A は開発用アカウントを表し、アカウント B は本番稼働用アカウントを表します。
  - アカウント A (開発用アカウント) タスクを実行する際、チュートリアルで提供されているサンプルポリシーを使用する代わりに、次のバケットポリシーを変更してクロスアカウントアクセス許可を付与します。

    JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Cross-account permissions",
            "Effect": "Allow",
            "Principal": {
```

```
                "AWS": "arn:aws:iam::111122223333:role/role-name"
            },
            "Action": [
                "s3:Get*",
                "s3:List*"
            ],
            "Resource": [
                "arn:aws:s3:::bucket-name/*"
            ]
        }
    ]
}
```

*account-id*は、IAM インスタンスプロファイルを作成した本番稼働用アカウントのアカウント番号を表します。

*role-name*は、作成した IAM インスタンスプロファイルの名前を表します。

bucket-name は、ステップ 1 で作成したバケットの名前を表します。バケット名の後に必ず /* が含まれるようにして、バケット内の各ファイルへのアクセスを提供します。

ステップ 3: 本番稼働用アカウントでリソースとクロスアカウントロールを作成する

本番稼働用アカウントで以下を行います。

- このガイドの手順を使用して、CodeDeploy リソース (アプリケーション、デプロイグループ、デプロイ設定、Amazon EC2 インスタンス、Amazon EC2 インスタンスプロファイル、サービスロールなど) を作成します。
- 開発用アカウントのユーザーが、この本番稼働用アカウントで CodeDeploy オペレーションを実行するために追加で引き受けるクロスアカウント IAM ロールを作成します。

  クロスAWS アカウントロールの作成に役立つガイドとして、チュートリアル: IAM ロールを使用してアカウント間のアクセスを委任します。チュートリアルのサンプルアクセス許可をポリシードキュメントに追加する代わりに、少なくとも次の 2 つの AWS 指定されたポリシーをロールにアタッチする必要があります。

  - AmazonS3FullAccess: S3 バケットが開発用アカウントにある場合にのみ必要です。引き受けた本番稼働用アカウントのロールに対して、リビジョンが保存されている、開発用アカウントの Amazon S3 サービスとリソースへのフルアクセスを提供します。

- AWSCodeDeployDeployerAccess: リビジョンを登録してデプロイすることをユーザーに許可します。

デプロイを開始するだけでなく、デプロイグループを作成および管理する場合、AWSCodeDeployFullAccess ポリシーの代わりに、AWSCodeDeployDeployerAccess ポリシーを追加します。IAM マネージドポリシーを使用して CodeDeploy タスクにアクセス許可を付与する方法の詳細については、「AWS CodeDeploy の マネージド (事前定義) ポリシー」を参照してください。

このクロスアカウントロールの使用時に、ほかの AWS サービスでタスクを実行する場合、追加のポリシーをアタッチできます。

> **Important**
>
> クロスアカウントの IAM ロールを作成する際に、本番稼働用アカウントへのアクセスを得るために必要な詳細を書き留めておきます。
> を使用してロール AWS マネジメントコンソール を切り替えるには、次のいずれかを指定する必要があります。
>
> - 引き受けたロールの認証情報を使用して本番稼働用アカウントにアクセスするための URL。URL は [確認] ページのクロスアカウント作成プロセスの最後に表示されます。
> - クロスアカウントロール名およびアカウント ID 番号またはエイリアス。
>
> を使用してロール AWS CLI を切り替えるには、以下を指定する必要があります。
>
> - 引き受けるクロスアカウントロールの ARN。

ステップ 4: Amazon S3 バケットにアプリケーションリビジョンをアップロードする

Amazon S3 バケットを作成したアカウントで

- Amazon S3 バケットにアプリケーションリビジョンをアップロードします。詳細については、「Amazon S3 に CodeDeploy のリビジョンをプッシュする (EC2/オンプレミスのデプロイのみ)」を参照してください。

ステップ 5: クロスアカウントロールを引き受け、アプリケーションをデプロイする

開発アカウントでは、 AWS CLI または AWS マネジメントコンソール を使用してクロスアカウントロールを引き受け、本番稼働用アカウントでデプロイを開始できます。

を使用してロール AWS マネジメントコンソール を切り替え、デプロイを開始する方法については、「ロールへの切り替え (AWS マネジメントコンソール)」および「」を参照してくださいEC2/オンプレミスコンピューティングプラットフォームのデプロイ作成 (コンソール)。

を使用してクロスアカウントロールを引き受け AWS CLI 、デプロイを開始する方法については、「IAM ロールへの切り替え (AWS Command Line Interface)」および「」を参照してくださいEC2/ オンプレミスコンピューティングプラットフォームのデプロイ作成 (CLI)。

を通じてロールを引き受ける方法の詳細については AWS STS、AWS Security Token Service「 ユーザーガイド」のAssumeRole」およびAWS CLI 「 コマンドリファレンス」の「 assume-role」を参照してください。

関連トピック:

- CodeDeploy: 開発用アカウントから本番稼働用アカウントへのデプロイ

CodeDeploy エージェントを使用してローカルマシン上のデプロイパッケージを検証する

CodeDeploy エージェントを使用して、ログインしているインスタンスのコンテンツをデプロイできます。これにより、デプロイ時に使用予定のアプリケーション仕様ファイル (AppSpec ファイル) とデプロイ予定のコンテンツの整合性を検証することができます。

アプリケーションおよびデプロイグループを作成する必要はありません。ローカルインスタンスに保存されているコンテンツをデプロイする場合は、 AWS アカウントも必要ありません。最も簡単な検証方法としては、AppSpec ファイル やデプロイ予定のコンテンツが含まれているディレクトリで、オプションを指定せずに codedeploy-agent deploy-local コマンドを実行します。このツールには、他のテストケースのオプションが含まれています。

ローカルマシン上のデプロイパッケージを検証することで、以下を行うことができます。

- アプリケーションリビジョンの整合性の検証。
- AppSpec ファイルのコンテンツの検証。

- 既存のアプリケーションコードを使用した CodeDeploy の初回実行。
- コンテンツの迅速なデプロイ (インスタンスにすでにログインしている場合)。

ローカルインスタンス、またはサポートされているリモートリポジトリタイプ (Amazon S3 バケットまたはパブリックの GitHub リポジトリ) 上に保存されているコンテンツをデプロイできます。

前提条件

ローカルのデプロイを開始する前に、以下の手順を行います。

- CodeDeploy エージェントでサポートされているインスタンスタイプを作成または使用します。詳細については、「CodeDeploy エージェントで対応するオペレーティングシステム」を参照してください。
- CodeDeploy エージェントのバージョン 2.0.0 以降をインストールします。`deploy-local` サブコマンドにはバージョン 2.0.0 以降が必要です。バージョン 1.8.x 以前では、代わりにスタンドアロン`codedeploy-local`コマンドを使用します。バージョンについては、「CodeDeploy エージェントのバージョン履歴」を参照してください。そのインストールについては、「CodeDeploy エージェントをインストールする」を参照してください。
- Amazon S3 バケットや GitHub リポジトリからコンテンツをデプロイする場合は、CodeDeploy で使用するユーザーをプロビジョニングします。詳細については、「ステップ 1: セットアップ」を参照してください。
- Amazon S3 バケットからアプリケーションリビジョンをデプロイする場合は、作業しているリージョンに Amazon S3 バケットを作成し、Amazon S3 バケットポリシーをバケットに適用します。このポリシーでは、アプリケーションリビジョンをダウンロードするために必要なアクセス許可をインスタンスに付与します。

  例えば、次の Amazon S3 バケットポリシーは、ARN `arn:aws:iam::444455556666:role/CodeDeployDemo` を含む IAM インスタンスプロファイルがアタッチされた Amazon EC2 インスタンスが、`amzn-s3-demo-bucket` という名前の Amazon S3 バケットの任意の場所からダウンロードすることを許可します。

```
{
    "Statement": [
        {
            "Action": [
                "s3:Get*",
                "s3:List*"
```

```
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::444455556666:role/CodeDeployDemo"
                ]
            }
        }
    ]
}
```

次の Amazon S3 バケットポリシーは、ARN `arn:aws:iam::444455556666:user/CodeDeployUser` を含む IAM ユーザーに関連付けられたオンプレミスインスタンスが、`amzn-s3-demo-bucket` という名前の Amazon S3 バケット内の任意の場所からダウンロードすることを許可します。

```
{
    "Statement": [
        {
            "Action": [
                "s3:Get*",
                "s3:List*"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::444455556666:user/CodeDeployUser"
                ]
            }
        }
    ]
}
```

Amazon S3 バケットポリシーを生成しアタッチする方法の詳細については、「バケットポリシーの例」を参照してください。

- Amazon S3 バケットまたは GitHub リポジトリからアプリケーションリビジョンをデプロイする場合は、IAM インスタンスプロファイルをセットアップして、そのプロファイルをインスタンスにアタッチします。詳細については、「ステップ 4: Amazon EC2 インスタンス用の IAM インスタンスプロファイルを作成する」、「CodeDeploy (AWS CLI または Amazon EC2 コンソール) 用の

Amazon EC2 インスタンスを作成する」、および「CodeDeploy 用の Amazon EC2 インスタンスを作成する (CloudFormation テンプレート)」を参照してください。

- GitHub からコンテンツをデプロイする場合は、GitHub アカウントおよびパブリックリポジトリを作成します。GitHub のアカウントを作成するには、「GitHub への参加」を参照してください。GitHub リポジトリを作成するには、「Repo の作成」を参照してください。

> **Note**
>
> プライベートリポジトリは、現在サポートされていません。プライベートの GitHub リポジトリにコンテンツが保存されている場合は、インスタンスにダウンロードし、`--bundle-location` オプションでローカルパスを指定します。

- インスタンスにデプロイするコンテンツ (AppSpec ファイルなど) を準備し、Amazon S3 バケットまたは GitHub リポジトリのローカルインスタンスに配置します。詳細については、「CodeDeploy のアプリケーションリビジョンの操作」を参照してください。
- 他の設定オプションでデフォルト以外の値を使用する場合は、設定ファイルを作成し、その設定ファイルをインスタンスに配置します (Amazon Linux、RHEL、または Ubuntu Server インスタンスの場合は `/etc/codedeploy-agent/conf/codedeployagent.yml`、Windows Server インスタンスの場合は `C:\ProgramData\Amazon\CodeDeploy\conf.yml`)。詳細については、「CodeDeploy エージェント設定リファレンス」を参照してください。

> **Note**
>
> Amazon Linux、RHEL、または Ubuntu Server インスタンスで設定ファイルを使用する場合は、以下のいずれかを行う必要があります。
>
> - `:root_dir:` 変数および `:log_dir:` 変数を使用して、デプロイルートおよびログディレクトリフォルダでデフォルト以外の場所を指定する。
> - `sudo` を使用して、CodeDeploy エージェントのコマンドを実行する。

ローカルのデプロイを作成する。

ローカルデプロイを作成するインスタンスで、ターミナルセッション (Amazon Linux、RHEL、または Ubuntu Server インスタンスの場合)、またはコマンドプロンプト (Windows Server の場合) を開き、ツールコマンドを実行します。

> **Note**
>
> codedeploy-agent deploy-local コマンドは、次の場所で `codedeploy-agent`バイナリを使用します。
>
> - Amazon Linux、RHEL、または Ubuntu Server: `/opt/codedeploy-agent/bin/codedeploy-agent`
> - Windows Server: `C:\ProgramData\Amazon\CodeDeploy\bin\codedeploy-agent.exe`
>
> 下位互換性のために、 codedeploy-local コマンドは同じ場所でも使用できます。
>
> - Amazon Linux、RHEL、または Ubuntu Server では、 コマンドは にあります`/opt/codedeploy-agent/bin/codedeploy-local`。
> - Windows Server では、 コマンドは にありま`C:\ProgramData\Amazon\CodeDeploy\bin\codedeploy-local.exe`す。
>
> codedeploy-local [options] 構文は引き続き機能し、 と同じオプションを受け入れますcodedeploy-agent deploy-local。

基本的なコマンド構文

```
codedeploy-agent deploy-local [options]
```

概要

```
codedeploy-agent deploy-local
[--bundle-location <value>]
[--type <value>]
[--file-exists-behavior <value>]
[--deployment-group <value>]
[--deployment-group-name <value>]
[--application-name <value>]
[--events <comma-separated values>]
[--agent-configuration-file <value>]
[--appspec-filename <value>]
```

オプション

-l、--bundle-location

アプリケーションリビジョンバンドルの場所。場所を指定しない場合は、現在作業中のディレクトリがデフォルトで使用されます。`--bundle-location` に値を指定する場合、`--type` の値も指定する必要があります。

バンドルの場所を表す形式の例を以下に示します。

- ローカルのAmazon Linux、RHEL、または Ubuntu Server インスタンス: `/path/to/local/bundle.tgz`
- ローカルの Windows Server インスタンス: `C:/path/to/local/bundle`
- Amazon S3 バケット: `s3://amzn-s3-demo-bucket/bundle.tar`
- GitHub リポジトリ: `https://github.com/account-name/repository-name/`

-t、--type

アプリケーションリビジョンバンドルの形式。サポートされるタイプには `tgz`、`tar`、`zip`、`directory` などがあります。タイプを指定しない場合は、デフォルトで `directory` が使用されます。`--type` に値を指定する場合、`--bundle-location` の値も指定する必要があります。

-b、--file-exists-behavior

デプロイのターゲット場所に存在しているが、デプロイが正常に完了していないファイルを処理する方法について説明します。オプションには DISALLOW、OVERWRITE、RETAIN などがあります。詳細については、「AWS CodeDeploy API リファレンス」の「fileExistsBehavior」を参照してください。

-g、--deployment-group

デプロイされるコンテンツのターゲット場所を示すフォルダへのパス。フォルダを指定しない場合は、default-local-deployment-group という名前のフォルダがデプロイルートディレクトリ内に作成されます。作成するローカルデプロイごとに、ツールは local-1234 (数値はプロセス ID) などの名前でこのフォルダ内にサブディレクトリを作成します。

-d、--deployment-group-name

DEPLOYMENT_GROUP_NAME 環境変数としてライフサイクルフックスクリプトに公開されるデプロイグループ名。デフォルトは LocalFleet です。

-a、--application-name

APPLICATION_NAME 環境変数としてライフサイクルフックスクリプトに公開されるアプリケーション名。デフォルトはバンドルの場所です。

-e、--events

上書きライフサイクルのイベントフックセット。AppSpec ファイルに表示されているイベントの代わりに、順番に実行されます。フックが複数ある場合は、カンマ区切りで指定できます。このオプションは以下の場合に使用できます。

- AppSpec ファイルをアップデートせずに、異なるイベントのセットを実行する場合。
- AppSpec ファイルの例外として、単一のイベントフックを実行する場合 (例: ApplicationStop)。

上書きリストで DownloadBundle イベントや Install イベントを指定しない場合は、指定したすべてのイベントフックより前に実行されます。DownloadBundle や Install を --events オプションのリストに含めた場合、これらのイベントに先行するイベントは、CodeDeploy デプロイでこれらのイベントの前に通常実行されるイベントに限ります。詳細については、「AppSpec の「hooks」セクション」を参照してください。

-c、--agent-configuration-file

デプロイに使用する設定ファイルの場所 (デフォルト以外の場所に設定ファイルを保存している場合)。設定ファイルは、デプロイ向けに別のデフォルト値および動作を指定します。

デフォルトでは、設定ファイルは /etc/codedeploy-agent/conf/codedeployagent.yml (Amazon Linux、RHEL、または Ubuntu Server インスタンス)、または C:/ProgramData/Amazon/CodeDeploy/conf.yml (Windows Server)。詳細については、「CodeDeploy エージェント設定リファレンス」を参照してください。

-A, --appspec-filename

AppSpec ファイルの名前。ローカルデプロイの場合、許容される値は appspec.yml と appspec.yaml です。デフォルトでは、AppSpec ファイルは appspec.yml と呼ばれます。

-h、--help

ヘルプコンテンツの概要を表示します。

-v、--version

ツールのバージョン番号を表示します。

例

有効なコマンド形式の例を以下に示します。

```
codedeploy-agent deploy-local
```

```
codedeploy-agent deploy-local --bundle-location /path/to/local/bundle/directory
```

```
codedeploy-agent deploy-local --bundle-location C:/path/to/local/bundle.zip --type zip
 --deployment-group my-deployment-group
```

```
codedeploy-agent deploy-local --bundle-location /path/to/local/directory --type
 directory --deployment-group my-deployment-group
```

Amazon S3 からバンドルをデプロイする。

```
codedeploy-agent deploy-local --bundle-location s3://amzn-s3-demo-bucket/bundle.tgz --
type tgz
```

```
codedeploy-agent deploy-local --bundle-location s3://amzn-s3-demo-bucket/bundle.zip?
versionId=1234&etag=47e8 --type zip --deployment-group my-deployment-group
```

パブリック GitHub リポジトリからバンドルをデプロイする:

```
codedeploy-agent deploy-local --bundle-location https://github.com/awslabs/aws-
codedeploy-sample-tomcat --type zip
```

```
codedeploy-agent deploy-local --bundle-location https://api.github.com/repos/awslabs/
aws-codedeploy-sample-tomcat/zipball/master --type zip
```

```
codedeploy-agent deploy-local --bundle-location https://api.github.com/repos/awslabs/
aws-codedeploy-sample-tomcat/zipball/HEAD --type zip
```

```
codedeploy-agent deploy-local --bundle-location https://api.github.com/repos/awslabs/
aws-codedeploy-sample-tomcat/zipball/1a2b3c4d --type zip
```

複数のライフサイクルイベントを指定するバンドルをデプロイする:

```
codedeploy-agent deploy-local --bundle-location /path/to/local/
bundle.tar --type tar --deployment-group my-deployment-group --events
 DownloadBundle,Install,ApplicationStart,HealthCheck
```

ApplicationStop ライフサイクルイベントを使用して、以前にデプロイされたアプリケーションを停止する:

```
codedeploy-agent deploy-local --bundle-location /path/to/local/bundle.tgz --type tgz --
deployment-group my-deployment-group --events ApplicationStop
```

特定のデプロイグループ ID を使用してデプロイする:

```
codedeploy-agent deploy-local --bundle-location C:/path/to/local/bundle/directory --
deployment-group 1234abcd-5dd1-4774-89c6-30b107ac5dca
```

```
codedeploy-agent deploy-local --bundle-location C:/path/to/local/bundle.zip --type zip
 --deployment-group 1234abcd-5dd1-4774-89c6-30b107ac5dca
```

CodeDeploy でのデプロイモニタリング

モニタリングは、CodeDeploy と AWS ソリューションの信頼性、可用性、パフォーマンスを維持する上で重要な部分です。マルチポイント障害が発生した場合は、その障害をより簡単にデバッグできるように、 AWS ソリューションのすべての部分からモニタリングデータを収集する必要があります。ただし、CodeDeploy のモニタリングを開始する前に、以下の質問に対する回答を反映したモニタリング計画を作成する必要があります。

- モニタリングの目的は何ですか?
- どのリソースをモニタリングしますか?
- どのくらいの頻度でこれらのリソースをモニタリングしますか?
- どのモニタリングツールを利用しますか?
- 誰がモニタリングタスクを実行しますか?
- 問題が発生したときに誰が通知を受け取りますか?

次のステップでは、さまざまなタイミングと負荷条件で CodeDeploy パフォーマンスを測定することにより、お客様の環境で通常の のパフォーマンスのベースラインを確定します。CodeDeploy のモニタリングでは、過去のモニタリングデータを保存し、現在のパフォーマンスデータと比較することで、パフォーマンスの通常パターンと異常パターンを特定し、問題に対処する方法を考案できます。

例えば、CodeDeploy を使用すると、デプロイおよびターゲットインスタンスのステータスをモニタリングできます。デプロイまたはインスタンスが失敗すると、アプリケーション仕様ファイルの再設定、CodeDeploy エージェントの再インストールまたは更新、アプリケーションまたはデプロイグループの設定の更新、インスタンスの設定または AppSpec ファイルの変更が必要が生じることがあります。

ベースラインを確立するには、少なくとも次の項目をモニタリングする必要があります。

- デプロイイベントとステータス
- インスタンスイベントとステータス

自動モニタリングツール

AWS には、CodeDeploy のモニタリングに使用できるさまざまなツールが用意されています。これらのツールの一部はモニタリングを行うように設定できますが、一部のツールは手動による介入が必要です。モニタリングタスクをできるだけ自動化することをお勧めします。

以下の自動化されたモニタリングツールを使用して、CodeDeploy を監視し、問題が発生したときにレポートできます。

- Amazon CloudWatch アラーム - 指定した期間にわたって単一のメトリクスをモニタリングし、複数の期間にわたる特定のしきい値に対するメトリクスの値に基づいて 1 つ以上のアクションを実行します。アクションは、Amazon Simple Notification Service (Amazon SNS) のトピックまたは Amazon EC2 Auto Scaling のポリシーに送信される通知です。CloudWatch アラームは、特定の状態にあるという理由だけでアクションを呼び出すことはありません。状態が変更され、指定された期間維持されている必要があります。詳細については、「Monitoring Deployments with Amazon CloudWatch Tools」を参照してください。

  CloudWatch アラームモニタリングを使用するサービスロールの更新に関する詳細については、「CloudWatch アクセス権限を CodeDeploy サービスロールに付与する」を参照してください。CloudWatch アラームモニタリングの CodeDeploy オペレーションへの追加の詳細については、「CodeDeploy でアプリケーションを作成する」、「CodeDeploy でデプロイグループを作成する」または「CodeDeploy を使用して、デプロイグループの設定を変更します。」を参照してください。

- Amazon CloudWatch Logs - AWS CloudTrail またはその他の出典のログファイルのモニタリング、保存、アクセスを行います。詳細については、「Amazon CloudWatch ユーザーガイド」の「ログファイルのモニタリング」を参照してください。

  CloudWatch コンソールを使用して CodeDeploy ログを表示する方法については、「CloudWatch Logs コンソールで CodeDeploy ログを表示する」を参照してください。。

- Amazon CloudWatch Events - イベントに一致したものを 1 つ以上のターゲットの関数またはストリームに渡して、変更、状態の情報の収集、是正措置を行います。詳細については、「Amazon CloudWatch ユーザーガイド」の「Amazon CloudWatch Events とは」を参照してください。

  CodeDeploy オペレーションで CloudWatch Events の使用の詳細については、「Amazon CloudWatch Events を使用したデプロイのモニタリング」を参照してください。

- AWS CloudTrail ログモニタリング – アカウント間でログファイルを共有し、CloudWatch Logs に送信CloudWatch CloudTrail ログファイルをリアルタイムでモニタリングし、Java でログ処理アプ

リケーションを書き込み、CloudTrail による配信後にログファイルが変更されていないことを確認します。詳細については、「AWS CloudTrail ユーザーガイド」の「CloudTrail ログファイルの使用」を参照してください。

CodeDeploy での CloudTrail の使用についての詳細は、「Monitoring Deployments」を参照してください。

- Amazon Simple 通知サービス - イベント駆動型のトリガーを設定して、成功または失敗など、デプロイおよびインスタンスイベントについての SMS や電子メール通知を受信します。詳細については、「トピックの作成 と Amazon Simple Notification Service とは」を参照してください。

  アカウントレベルの Amazon SNS の CodeDeploy 通知をセットアップする詳しい方法については、「Monitoring Deployments with Amazon SNS Event Notifications」を参照してください。

手動モニタリングツール

CodeDeploy のモニタリングでもう 1 つ重要な点は、CloudWatch のアラームの対象外の項目を手動でモニタリングすることです。CodeDeploy、CloudWatch、およびその他の AWS コンソールダッシュボードには、 AWS 環境の状態がat-a-glanceビューが表示されます。CodeDeploy デプロイのログファイルを確認することもお勧めします。

- CodeDeploy コンソールは以下で表示されます。
  - デプロイのステータス
  - リビジョンのデプロイを最後に試みた日時と、最後に成功した日時
  - デプロイの成功、失敗、スキップ、進行中のインスタンス数
  - オンプレミスインスタンスのステータス
  - オンプレミスインスタンスが登録、または登録解除された日時
- CloudWatch ホームページには、次の内容が表示されます。
  - 現在のアラームとステータス
  - アラームとリソースのグラフ
  - サービスのヘルスステータス

  また、CloudWatch を使用して以下のことを行えます。
  - 重視するサービスをモニタリングするためのカスタマイズしたダッシュボードを作成します
  - メトリクスデータをグラフ化して、問題のトラブルシューティングを行い、傾向を確認する
  - すべての AWS リソースメトリクスを検索して参照する

- 問題があることを通知するアラームを作成/編集する

トピック

- Monitoring Deployments with Amazon CloudWatch Tools
- Monitoring Deployments
- Monitoring Deployments with Amazon SNS Event Notifications

Amazon CloudWatch ツールを使用したデプロイのモニタリング

CodeDeploy デプロイは、Amazon CloudWatch Events、CloudWatch アラーム、およびAmazon CloudWatch Logs の CloudWatch ツールを使用して、モニタリングできます。

CodeDeploy エージェントおよびデプロイによって作成されたログを確認することにより、デプロイの障害の原因のトラブルシューティングを行うのに役立ちます。各インスタンスで個別に CodeDeploy ログを確認する代わりに、CloudWatch Logs を使用してすべてのログを一元的にモニタリングできます。

CloudWatch アラームと CloudWatch Events を使用して CodeDeploy デプロイをモニタリングする方法については、以下のトピックを参照してください。

トピック

- CodeDeploy での CloudWatch アラームを使用したデプロイのモニタリング
- Amazon CloudWatch Events を使用したデプロイのモニタリング

CodeDeploy での CloudWatch アラームを使用したデプロイのモニタリング

CodeDeploy オペレーションで使用するインスタンスや Amazon EC2 Auto Scaling グループの CloudWatch アラームを作成する方法を説明します。アラームは、指定期間にわたって単一のメトリクスを監視し、その値と複数期間に対するしきい値との比較結果に基づいて 1 つ以上のアクションを実行します。CloudWatch アラームは、状態が変化したときにアクションを呼び出します (例えば、OK から ALARM)。

ネイティブ CloudWatch アラーム機能では、デプロイで使用しているインスタンスが失敗したときに Amazon SNS 通知を送信したり、インスタンスを停止、終了、再起動、復旧したりするな

ど、CloudWatch でサポートされているアクションを指定できます。CodeDeploy オペレーションでは、デプロイグループに関連付けた CloudWatch アラームがアクティブ化されるたびにデプロイを停止するようにデプロイグループを設定できます。

最大 10 個の CloudWatch アラームを CodeDeploy デプロイグループに関連付けることができます。指定したアラームがアクティブ化した場合、デプロイは停止し、ステータスは [Stopped] に更新されます。このオプションを使用するには、CodeDeploy サービスロールに対して CloudWatch へのアクセス許可を付与する必要があります。

CloudWatch コンソール での CloudWatch アラームの設定に関する詳細については、Amazon CloudWatch アラームの作成 は Amazon CloudWatch ユーザーガイド を参照してください。

CloudWatch アラームを CodeDeploy でデプロイグループに関連付ける方法については、「CodeDeploy でデプロイグループを作成する」および「CodeDeploy を使用して、デプロイグループの設定を変更します。」を参照してください。

トピック

- CloudWatch アクセス権限を CodeDeploy サービスロールに付与する

CloudWatch アクセス権限を CodeDeploy サービスロールに付与する

デプロイで CloudWatch アラームモニタリングを使用する前に、CodeDeploy オペレーションで使用するサービスロールに対して CloudWatch リソースへのアクセス許可を付与する必要があります。

CloudWatch アクセス権限を CloudWatch サービスロールに付与する

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. IAM コンソールのナビゲーションペインで [ロール] を選択します。
3. AWS CodeDeploy オペレーションで使用するサービスロールの名前を選択します。
4. [アクセス許可] タブの [インラインポリシー] エリアで、[ロールポリシーの作成] を選択します。

   -または-

   [Create Role Policy] ボタンを使用できない場合は、[Inline Policies] エリアを拡張して、[click here] を選択します。
5. [Set Permissions] ページで、[Custom Policy] を選択し、次に [Select] を選択します。

6. [Review Policy] ページで、[Policy Name] フィールドに、このポリシーを識別するための名前 [CWAlarms] などを入力します。
7. [Policy Document] フィールドに以下を貼り付けます。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "cloudwatch:DescribeAlarms",
            "Resource": "*"
        }
    ]
}
```

8. [ポリシーを適用] を選びます。

Amazon CloudWatch Events を使用したデプロイのモニタリング

Amazon CloudWatch Events を使用して、インスタンスの状態の変更、または CodeDeploy オペレーションのデプロイ (「イベント」として) を検出して対処できます。次に、作成したルールに基づいて CloudWatch Events はデプロイまたはインスタンスが、ルールで指定した状態になると、1 つ以上のターゲットアクションを呼び出します。状態変更のタイプに応じて、通知を送信、状態情報を取得し、修正作業またはその他のアクションを取ることができます。CodeDeploy オペレーションの一部として CloudWatch Events を使用している場合は、以下のターゲットのタイプを選択できます。

- AWS Lambda 関数
- Kinesis Streams
- Amazon SQS キュー
- 組み込みターゲット (EC2 CreateSnapshot API call、EC2 RebootInstances API call、EC2 StopInstances API call および EC2 TerminateInstances API call) 。
- Amazon SNS トピック

次にユースケースをいくつか示します。

- Lambda 機能を使用して、デプロイが失敗するたびに Slack チャネルに通知を配信します。
- Kinesis ストリームにデプロイまたはインスタンスのデータをプッシュして、包括的でリアルタイムの状態モニタリングをサポートします。
- CloudWatch アラームアクションを使用して、指定したデプロイやインスタンスイベントが発生したときに、Amazon EC2 インスタンスを自動的に停止、終了、再起動、復旧します。

このトピックの残りの部分では、CodeDeploy の CloudWatch Events ルールを作成するための基本的な手順について説明します。ただし CodeDeploy オペレーションで使用するイベントルールを作成する前に、以下のことを実行する必要があります。

- CloudWatch Events の前提条件を完了します。詳細については、「Amazon CloudWatch Events 前提条件」を参照してください。
- CloudWatch Events のイベント、ルール、およびターゲットをしっかりと理解しておきます。詳細については、Amazon CloudWatch Events とは」を参照してください。 および新しい CloudWatch Events – AWS リソースの変更を追跡して対応します。
- イベントのルールで使用するターゲットを作成します。

CodeDeploy の CloudWatch Events ルールを作成するには

1. CloudWatch コンソールの https://console.aws.amazon.com/cloudwatch/ を開いてください。
2. ナビゲーションペインの Events] を選択します。
3. [ルールの作成] を選択してから、[イベントの選択] で [AWS CodeDeploy] を選択します。
4. 詳細タイプを指定します。
   - インスタンスとデプロイの両方のすべての状態変更に適用されるルールを作成するには、[Any detail type] を選択してから、ステップ 6 に進んでください。
   - インスタンスのみに適用するルールを作成するためには [Specific detail type] を選択してから、[CodeDeploy インスタンスの状態変更通知] を選択します。
   - デプロイのみに適用するルールを作成するためには [Specific detail type] を選択してから、[CodeDeploy デプロイの状態変更通知] を選択します。
5. ルールを適用する状態変更を指定します。
   - すべての状態変更に適用されるルールを作成するには、[Any state] を選択します。

- いくつかの状態変更のみに適用されるルールを作成するためには、Specific state(s) を選択してから、リストから 1 つ以上のステータス値を選択します。次の表は、使用できるステータス値を一覧表示します。

デプロイのステータス値	インスタンスのステータス値
FAILURE	FAILURE
開始	開始
STOP	準備完了
QUEUED	SUCCESS
準備完了	
SUCCESS	

6. ルールが適用される CodeDeploy アプリケーションを指定します。
   - すべてのアプリケーションに適用するルールを作成するためには、[Any application] を選択し、ステップ 8 に進んでください。
   - 1 つのアプリケーションのみに適用するルールを作成するためには、[Specific application] を選択してから、リストからアプリケーション名を選択します。
7. ルールが適用されるデプロイを指定します。
   - 選択したアプリケーションと関連付けられたすべてのデプロイグループに適用されるルールを作成するためには、[Any deployment group] を選択します。
   - 選択したアプリケーションに関連付けられる 1 つのデプロイグループのみに適用されるルールを作成するためには、[Specific deployment group(s)] を選択してから、リストからデプロイグループ名を選択します。
8. ルール設定を確認して、イベントモニタリング要件を満たしていることを確認します。
9. [Targets] エリアで、[Add target*] を選択します。
10. Select target type リストで、このルールを使用するために準備したターゲットのタイプを選択してから、そのタイプに必要な追加オプションを設定します。
11. 設定の詳細 を選択します。

12. ルールの詳細の設定ページで、ルールの名前と説明を入力し、ステートボックスを選択してルールを有効にします。
13. ルールが適切であることを確認したら、[Create rule] を選択します。

を使用したデプロイのモニタリング AWS CloudTrail

CodeDeploy は CloudTrail と統合されています。CloudTrail は、 AWS アカウントで CodeDeploy によって、または CodeDeploy に代わって行われた API コールをキャプチャし、指定した Amazon S3 バケットにログファイルを配信するサービスです。CloudTrail は、CodeDeploy コンソール、 を介した CodeDeploy コマンド AWS CLI、または CodeDeploy APIs。CloudTrail によって収集された情報を使用して、CodeDeploy に対してどのようなリクエストが行われたかを判断することができます。リクエストの作成元のソース IP アドレス、リクエストの実行者、リクエストの実行日時などです。設定や有効化の方法など、CloudTrail の詳細については、「AWS CloudTrail ユーザーガイド」を参照してください。

CloudTrail での CodeDeploy 情報

AWS アカウントで CloudTrail ログ記録が有効になっている場合、CodeDeploy アクションに対して行われた API コールはログファイルに記録されます。CodeDeploy レコードは、他の AWS サービスレコードと一緒にログファイルに書き込まれます。CloudTrail は、期間とファイルサイズに基づいて、新しいファイルをいつ作成して書き込むかを決定します。

CodeDeploy のアクションはすべてログに記録されます。このアクションについては、「AWS CodeDeploy コマンドラインリファレンス」および「AWS CodeDeploy API リファレンス」に記載されています。たとえば、デプロイを作成し、アプリケーションを削除し、アプリケーションリビジョンを登録する呼び出しは、CloudTrail ログファイルにエントリを生成します。

各ログエントリには、リクエストの生成者に関する情報が含まれます。ログのユーザー ID 情報は、リクエストがルート認証情報またはユーザー認証情報で行われたか、ロールまたはフェデレーティッドユーザーの一時的なセキュリティ認証情報で行われたか、または別の AWS サービスによって行われたかを判断するのに役立ちます。詳細については、CloudTrail ログイベントリファレンスの userIdentity フィールドを参照してください。

必要な場合はログファイルを自身のバケットに保管できますが、ログファイルを自動的にアーカイブまたは削除するように Amazon S3 ライフサイクルルールを定義することもできます。デフォルトで、Amazon S3 のサーバー側の暗号化 (SSE) を使用して、ログファイルが暗号化されます。

新しいログファイルが配信されるときに、CloudTrail が Amazon SNS の通知を発行するようにできます。詳細については、CloudTrail 用の Amazon SNS 通知の設定を参照してください。

複数の AWS リージョンと複数の AWS アカウントからの CodeDeploy ログファイルを 1 つの Amazon S3 バケットに集約することもできます。詳細については、「複数のリージョンからの CloudTrail ログファイルの受信」を参照してください。

CodeDeploy ログファイルエントリの理解

CloudTrail ログファイルには、複数の JSON 形式イベントで構成されるひとつ以上のログエントリを記録できます。ログエントリは任意の送信元からの単一のリクエストを表し、リクエストされたアクション、任意のパラメータ、アクションの日時などに関する情報を含みます。ログエントリは、特定の順序になるように生成されるわけではありません。つまり、パブリック API コールの順序付けられたスタックトレースではありません。

以下の例は、CodeDeploy がデプロイグループのアクションを作成することを示す CloudTrail ログエントリを示しています。

```
{
 "Records": [{
  "eventVersion": "1.02",
  "userIdentity": {
   "type": "AssumedRole",
   "principalId": "AKIAI44QH8DHBEXAMPLE:203.0.113.11",
   "arn": "arn:aws:sts::123456789012:assumed-role/example-role/203.0.113.11",
   "accountId": "123456789012",
   "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
   "sessionContext": {
    "attributes": {
     "mfaAuthenticated": "false",
     "creationDate": "2014-11-27T03:57:36Z"
    },
    "sessionIssuer": {
     "type": "Role",
     "principalId": "AKIAI44QH8DHBEXAMPLE",
     "arn": "arn:aws:iam::123456789012:role/example-role",
     "accountId": "123456789012",
     "userName": "example-role"
    }
   }
  },
  "eventTime": "2014-11-27T03:57:36Z",
```

```
  "eventSource": "codedeploy.amazonaws.com",
  "eventName": "CreateDeploymentGroup",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "203.0.113.11",
  "userAgent": "example-user-agent-string",
  "requestParameters": {
   "applicationName": "ExampleApplication",
   "serviceRoleArn": "arn:aws:iam::123456789012:role/example-instance-group-role",
   "deploymentGroupName": "ExampleDeploymentGroup",
   "ec2TagFilters": [{
               "value": "CodeDeployDemo",
    "type": "KEY_AND_VALUE",
    "key": "Name"
           }],
           "deploymentConfigName": "CodeDeployDefault.HalfAtATime"
  },
  "responseElements": {
   "deploymentGroupId": "7d64e680-e6f4-4c07-b10a-9e117EXAMPLE"
  },
  "requestID": "86168559-75e9-11e4-8cf8-75d18EXAMPLE",
  "eventID": "832b82d5-d474-44e8-a51d-093ccEXAMPLE",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
 },
    ... additional entries ...
    ]
}
```

Amazon SNS イベント通知を使用したデプロイのモニタリング

トリガーを CodeDeploy デプロイグループに追加すると、そのデプロイグループのデプロイまたはインスタンスに関連するイベントに関する通知を受信できます。これらの通知は、トリガーのアクションの一部にした Amazon SNS トピックをサブスクライブする受信者に送信されます。

SMS メッセージまたは電子メールメッセージで、CodeDeploy イベントの通知を受信できます。また、Amazon SQS キューへのメッセージの送信、または AWS Lambdaでの関数の呼び出しなど、指定されたイベントが他の方法で発生したときに作成される JSON データを使用することもできます。デプロイおよびインスタンストリガーに提供される JSON データの構造については、「CodeDeploy トリガーの JSON データ形式」を参照してください。

次の場合に、トリガーを使用して通知を受け取ることもできます。

- トラブルシューティングできるように、デプロイが失敗または停止したときに知る必要がある開発者の場合。
- Amazon EC2 フリートの状態を監視するために、いくつのインスタンスが失敗したかを知る必要があるシステム管理者の場合。
- デスクトップの電子メールクライアントのフォルダに様々なタイプの通知をルーティングするフィルタリングルールを使用して、デプロイおよびインスタンスのイベントの数を一目で把握したいマネージャーの場合。

次のいずれかのイベントタイプに対して、CodeDeploy デプロイグループごとに最大 10 のトリガーを作成できます。

デプロイイベント	インスタンスイベント
• Success • 失敗 • 起動済み • 停止 • ロールバック • 準備完了[1] • すべてのデプロイイベント	• Success • 失敗 • 起動済み • 準備完了[1] • すべてのインスタンスイベント

[1] Blue/Green デプロイにのみ適用されます。最新のアプリケーションのリビジョンが、置き換え先環境でインスタンスにインストールされており、元の環境からのトラフィックをロードバランサーの背後で再ルーティングすることができることを示します。詳細については、「CodeDeploy でのデプロイグループの使用」を参照してください。

トピック

- Amazon SNS アクセス許可を CodeDeploy サービスロールに付与する
- CodeDeploy イベントのトリガーを作成
- CodeDeploy デプロイグループのトリガーの編集
- CodeDeploy デプロイグループからトリガーを削除
- CodeDeploy トリガーの JSON データ形式

Amazon SNS アクセス許可を CodeDeploy サービスロールに付与する

トリガーが通知を生成する前に、CodeDeploy オペレーションで使用するサービスロールには、Amazon SNS リソースへのアクセス許可を付与する必要があります。

サービスロールへの Amazon SNS アクセス許可の付与

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. IAM コンソールのナビゲーションペインで [ロール] を選択します。
3. AWS CodeDeploy オペレーションで使用するサービスロールの名前を選択します。
4. [アクセス許可] タブの [インラインポリシー] エリアで、[ロールポリシーの作成] を選択します。

   -または-

   [Create Role Policy] ボタンを使用できない場合は、[Inline Policies] エリアを拡張して、[click here] を選択します。
5. [Set Permissions] ページで、[Custom Policy] を選択し、次に [Select] を選択します。
6. [ポリシーの確認] ページで、[ポリシー名] フィールドにポリシーを識別する名前 (`SNSPublish` など) を入力します。
7. [Policy Document] フィールドに以下を貼り付けます。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sns:Publish",
            "Resource": "*"
        }
    ]
}
```

8. [ポリシーを適用] を選びます。

CodeDeploy イベントのトリガーを作成

デプロイまたはインスタンスイベントの Amazon Simple Notification Service (Amazon SNS) トピックを発行 AWS CodeDeploy するトリガーを作成できます。次に、そのイベントが発生した場合、関連付けられたトピックのすべてのサブスクライバーは、SMS メッセージまたは E メールメッセージなどの、トピックで指定されたエンドポイントを経由して通知を受信します。Amazon SNS では、トピックをサブスクライブするための複数の方法が用意されています。

トリガーを作成する前に、トリガーの参照先となる Amazon SNS トピックを設定する必要があります。詳細については、「トピックの作成」を参照してください。トピックを作成する際には、`Topic-group-us-west-3-deploy-fail` または `Topic-group-project-2-instance-stop` などの形式で、目的をわかりやすくする名前を付けることをお勧めします。

トリガーのために通知を送信する前に、Amazon SNS サービスロールに CodeDeploy アクセス許可を付与する必要もあります。詳細については、「Amazon SNS アクセス許可を CodeDeploy サービスロールに付与する」を参照してください。

トピックを作成した後、サブスクライバーを追加できます。トピックの作成、管理、およびサブスクライブについての詳細は、「Amazon Simple Notification Service とは」を参照してください。

CodeDeploy イベントの通知を送信するトリガーの作成 (コンソール)

CodeDeploy コンソールを使用して、CodeDeploy イベント用のトリガーを作成できます。セットアッププロセスの最後に、アクセス許可およびトリガーの詳細の両方が正しくセットアップされていることを確認するためにテスト通知メッセージが送信されます。

CodeDeploy イベントのトリガーを作成

1. で AWS マネジメントコンソール、 AWS CodeDeploy コンソールを開きます。
2. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

3. ナビゲーションペインで [デプロイ] を展開し、[アプリケーション] を選択します。
4. [アプリケーション] ページで、トリガーを追加するデプロイグループに関連付けられているアプリケーションの名前を選択します。

5. [アプリケーションの詳細] ページで、トリガーを追加するデプロイグループを選択します。
6. [編集] を選択します。
7. [Advanced - optional (詳細 - オプション)] を展開します。
8. [トリガー] エリアで、[Create trigger (トリガーの作成)] を選択します。
9. [Create deployment trigger (デプロイトリガーの作成)] ペインで、次の操作を行います。

   a. [トリガー名] に、用途をわかりやすく示すトリガーの名前を入力します。`Trigger-group-us-west-3-deploy-fail` または `Trigger-group-eu-central-instance-stop` などの形式をお勧めします。

   b. [イベント] で、通知を送信するために Amazon SNS トピックをトリガーするイベントタイプまたはタイプを選択します。

   c. [Amazon SNS トピック] で、このトリガーの通知を送信するために作成したトピックの名前を選択します。

   d. [Create trigger (トリガーの作成)] を選択します。CodeDeploy は、CodeDeploy および Amazon SNS トピックとの間のアクセスを正しく設定したことを確認するためのテスト通知を送信します。トピックに対して選択したエンドポイントタイプに応じて、トピックをサブスクライブしている場合は、SMS メッセージまたは E メールメッセージで確認を受信します。

10. [Save changes] (変更の保存) をクリックします。

CodeDeploy イベントの通知を送信するトリガーの作成 (CLI)

CLI を使用して、デプロイグループを作成するときにトリガーを含めることも、既存のデプロイグループにトリガーを追加することもできます。

新しいデプロイグループの通知を送信するためのトリガーを作成するには

JSON ファイルを作成してデプロイグループを設定し、デプロイメントグループの作成 コマンドの使用 `--cli-input-json` オプションを実行します。

JSON ファイルを作成する最も簡単な方法は、`--generate-cli-skeleton` オプションを使用して JSON 形式のコピーを取得し、プレーンテキストエディターで必要な値を指定することです。

1. 次のコマンドを実行し、結果をプレーンテキストエディターにコピーします。

```
aws deploy create-deployment-group --generate-cli-skeleton
```

2. 既存の CodeDeploy アプリケーションの名前を追加します。

```
{
    "applicationName": "TestApp-us-east-2",
    "deploymentGroupName": "",
    "deploymentConfigName": "",
    "ec2TagFilters": [
        {
            "Key": "",
            "Value": "",
            "Type": ""
        }
    ],
    "onPremisesInstanceTagFilters": [
        {
            "Key": "",
            "Value": "",
            "Type": ""
        }
    ],
    "autoScalingGroups": [
        ""
    ],
    "serviceRoleArn": "",
    "triggerConfigurations": [
        {
            "triggerName": "",
            "triggerTargetArn": "",
            "triggerEvents": [
                ""
            ]
        }
    ]
}
```

3. 設定するパラメータの値を指定します。

   create-deployment-group コマンドを使用する場合は、少なくとも次のパラメータ値を指定する必要があります。

   - `applicationName`: アカウントで既に作成されたアプリケーションの名前。
   - `deploymentGroupName`: 作成するデプロイグループの名前。

- serviceRoleArn: アカウント内の CodeDeploy 用にセットアップする既存のサービスロールの ARN。詳細については、「ステップ 2: CodeDeployのサービスのロールを作成する」を参照してください。

triggerConfigurations セクションで、次のパラメーターの値を指定します。

- triggerName: 簡単に識別できるように、トリガーに付与した名前。Trigger-group-us-west-3-deploy-fail または Trigger-group-eu-central-instance-stop などの形式をお勧めします。
- triggerTargetArn: トリガーに関連付けるために作成した Amazon SNS トピックの ARN で以下の形式: arn:aws:sns:us-east-2:444455556666:NewTestTopic。
- triggerEvents: 通知をトリガーするイベントタイプまたはイベント。1 つ以上のイベントタイプを指定し、複数のイベントタイプ名をカンマで区切ることができます (たとえば、"triggerEvents": ["DeploymentSuccess","DeploymentFailure","InstanceFailure"])。1 つ以上のイベントタイプを追加すると、これらのすべてのタイプの通知は、それぞれの異なるトピックではなく、指定したトピックに送信されます。次のイベントタイプから選択できます。
  - DeploymentStart
  - DeploymentSuccess
  - DeploymentFailure
  - DeploymentStop
  - DeploymentRollback
  - DeploymentReady (Blue/Green デプロイで置き換え先インスタンスにのみ適用します)
  - InstanceStart
  - InstanceSuccess
  - InstanceFailure
  - InstanceReady (Blue/Green デプロイで置き換え先インスタンスにのみ適用します)

以下の設定例では、dep-group-ghi-789-2 という名前のアプリケーション用の TestApp-us-east-2 という名前のデプロイグループを作成し、デプロイの開始、成功、または失敗のたびに通知の送信を促すトリガーを作成します。

```
{
    "applicationName": "TestApp-us-east-2",
```

```
    "deploymentConfigName": "CodeDeployDefault.OneAtATime",
    "deploymentGroupName": "dep-group-ghi-789-2",
    "ec2TagFilters": [
        {
            "Key": "Name",
            "Value": "Project-ABC",
            "Type": "KEY_AND_VALUE"
        }
    ],
    "serviceRoleArn": "arn:aws:iam::444455556666:role/AnyCompany-service-role",
    "triggerConfigurations": [
        {
            "triggerName": "Trigger-group-us-east-2",
            "triggerTargetArn": "arn:aws:sns:us-east-2:444455556666:us-east-
deployments",
            "triggerEvents": [
                "DeploymentStart",
                "DeploymentSuccess",
                "DeploymentFailure"
            ]
        }
    ]
}
```

4. 更新を JSON ファイルとして保存し、create-deployment-group コマンドを実行するときに `--cli-input-json` オプションを使用してそのファイルを呼び出します。

> **Important**
>
> ファイル名の前に必ず `file://` を含めてください。このコマンドでは必須です。

```
aws deploy create-deployment-group --cli-input-json file://filename.json
```

作成プロセスの最後に、アクセス許可およびトリガーの詳細の両方が正しく設定されていることを示すテスト通知メッセージが届きます。

既存のデプロイグループの通知を送信するためのトリガーを作成するには

を使用して CodeDeploy イベントのトリガーを既存のデプロイグループ AWS CLI に追加するには、JSON ファイルを作成してデプロイグループを更新し、 `--cli-input-json`オプションを使用して update-deployment-group コマンドを実行します。

JSON ファイルを作成する最も簡単な方法は、get-deployment-group コマンドを実行し、JSON 形式でデプロイグループの設定をコピーして、プレーンテキストエディターでパラメーター値を更新することです。

1. 次のコマンドを実行し、結果をプレーンテキストエディターにコピーします。

```
aws deploy get-deployment-group --application-name application --deployment-group-
name deployment-group
```

2. 出力から次のものを削除します。

   - 出力の先頭の [{ "deploymentGroupInfo":] を削除します。
   - 出力の末尾の [}] を削除します。
   - [deploymentGroupId] を含む行を削除します。
   - [deploymentGroupName] を含む行を削除します。

   テキストファイルのコンテンツは、次のようになります。

```
{
    "applicationName": "TestApp-us-east-2",
    "deploymentConfigName": "CodeDeployDefault.OneAtATime",
    "autoScalingGroups": [],
    "ec2TagFilters": [
        {
            "Type": "KEY_AND_VALUE",
            "Value": "Project-ABC",
            "Key": "Name"
        }
    ],
    "triggerConfigurations": [],
    "serviceRoleArn": "arn:aws:iam::444455556666:role/AnyCompany-service-role",
    "onPremisesInstanceTagFilters": []
}
```

3. [triggerConfigurations] セクションで、[triggerEvents]、[triggerTargetArn]、および [triggerName] パラメーターのデータを追加します。トリガーの設定パラメータについての詳細は、「TriggerConfig」を参照してください。

   テキストファイルのコンテンツは、次のようになります。このコードでは、デプロイの開始、成功、または失敗のたびに通知を送信するように求められます。

```
{
    "applicationName": "TestApp-us-east-2",
    "deploymentConfigName": "CodeDeployDefault.OneAtATime",
    "autoScalingGroups": [],
    "ec2TagFilters": [
        {
            "Type": "KEY_AND_VALUE",
            "Value": "Project-ABC",
            "Key": "Name"
        }
    ],
    "triggerConfigurations": [
        {
            "triggerEvents": [
                "DeploymentStart",
                "DeploymentSuccess",
                "DeploymentFailure"
            ],
            "triggerTargetArn": "arn:aws:sns:us-east-2:444455556666:us-east-
deployments",
            "triggerName": "Trigger-group-us-east-2"
        }
    ],
    "serviceRoleArn": "arn:aws:iam::444455556666:role/AnyCompany-service-role",
    "onPremisesInstanceTagFilters": []
}
```

4. アップデートを JSON ファイルとして保存し、--cli-input-json オプションを使用して、update-deployment-group コマンドを実行します。必ず --current-deployment-group-name オプションを含めて、*filename* を JSON ファイルの名前に置き換えてください。

> **Important**
>
> ファイル名の前に必ず `file://` を含めてください。このコマンドでは必須です。

```
aws deploy update-deployment-group --current-deployment-group-name deployment-group-name --cli-input-json file://filename.json
```

作成プロセスの最後に、アクセス許可およびトリガーの詳細の両方が正しく設定されていることを示すテスト通知メッセージが届きます。

CodeDeploy デプロイグループのトリガーの編集

通知の要件が変更された場合は、新しいトリガーを作成するのではなく、トリガーを変更することができます。

CodeDeploy トリガーの変更 (CLI)

を使用してデプロイグループを更新するときに CodeDeploy イベントのトリガーの詳細 AWS CLI を変更するには、JSON ファイルを作成してデプロイグループのプロパティの変更を定義し、 `--cli-input-json`オプションを指定して update-deployment-group コマンドを実行します。

JSON ファイルを作成する最も簡単な方法は、get-deployment-group コマンドを実行して現在のデプロイグループの詳細を JSON 形式で取得し、プレーンテキストエディターで必要な値を編集することです。

1. 以下のコマンドを実行します (アプリケーションおよびデプロイグループの名前を *application* および *deployment-group* に置き換えます)。

   ```
   aws deploy get-deployment-group --application-name application --deployment-group-name deployment-group
   ```

2. コマンド結果をプレーンテキストエディターにコピーし、次のものを削除します。

   - 出力の先頭の [{ "deploymentGroupInfo":] を削除します。
   - 出力の末尾の [}] を削除します。
   - [deploymentGroupId] を含む行を削除します。

- [deploymentGroupName] を含む行を削除します。

テキストファイルのコンテンツは、次のようになります。

```
{
    "applicationName": "TestApp-us-east-2",
    "deploymentConfigName": "CodeDeployDefault.OneAtATime",
    "autoScalingGroups": [],
    "ec2TagFilters": [
        {
            "Type": "KEY_AND_VALUE",
            "Value": "East-1-Instances",
            "Key": "Name"
        }
    ],
    "triggerConfigurations": [
        {
            "triggerEvents": [
                "DeploymentStart",
                "DeploymentSuccess",
                "DeploymentFailure",
                "DeploymentStop"
            ],
            "triggerTargetArn": "arn:aws:sns:us-east-2:111222333444:Trigger-group-
us-east-2",
            "triggerName": "Trigger-group-us-east-2"
        }
    ],
    "serviceRoleArn": "arn:aws:iam::444455556666:role/AnyCompany-service-role",
    "onPremisesInstanceTagFilters": []
}
```

3. 必要に応じてパラメーターを変更します。トリガーの設定パラメータについての詳細は、「TriggerConfig」を参照してください。
4. アップデートを JSON ファイルとして保存し、--cli-input-json オプションを使用して、update-deployment-group コマンドを実行します。必ず --current-deployment-group-name オプションを含めて、*filename* を JSON ファイルの名前に置き換えてください。

> **Important**
>
> ファイル名の前に必ず `file://` を含めてください。このコマンドでは必須です。

```
aws deploy update-deployment-group --current-deployment-group-name deployment-group-name --cli-input-json file://filename.json
```

作成プロセスの最後に、アクセス許可およびトリガーの詳細の両方が正しく設定されていることを示すテスト通知メッセージが届きます。

CodeDeploy デプロイグループからトリガーを削除

デプロイグループごとに 10 個のトリガーの制限があるため、使用されなくなったトリガーを削除することをお勧めします。トリガーの削除は元に戻すことはできませんが、1 つを再作成することはできます。

デプロイグループ (コンソール) からトリガーを削除

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。

   > **Note**
   >
   > 「CodeDeploy の開始方法」で設定したのと同じユーザーでサインインします。

2. ナビゲーションペインで [デプロイ] を展開し、[アプリケーション] を選択します。
3. [アプリケーション] ページで、トリガーを削除するデプロイグループに関連付けられているアプリケーションの名前を選択します。
4. [アプリケーションの詳細] ページで、トリガーを削除するデプロイグループを選択します。
5. [編集] を選択します。
6. [Advanced - optional (詳細 - オプション)] を展開します。
7. [トリガー] 領域で、削除するトリガーを選択し、[Delete trigger (トリガーを削除)] を選択します。
8. [Save changes] (変更の保存) をクリックします。

デプロイグループ (CLI) からトリガーを削除

CLI を使用してトリガーを削除するには、空のトリガー設定パラメータを使用して、update-deployment-group コマンドを呼び出し、次のように指定します。

- デプロイグループに関連付けられたアプリケーションの名前。アプリケーション名のリストを表示するには、[list-applications] コマンドを呼び出します。
- アプリケーションに関連付けられたデプロイグループの名前。デプロイグループ名のリストを表示するには、[list-deployment-groups] コマンドを呼び出します。

例:

```
aws deploy update-deployment-group --application-name application-name --current-
deployment-group-name deployment-group-name --trigger-configurations
```

CodeDeploy トリガーの JSON データ形式

デプロイまたはインスタンスのトリガーが、Amazon SQS キューへのメッセージの送信、または AWS Lambdaでの関数の呼び出しなどのカスタム通知ワークフローでアクティブ化されたときに作成される JSON 出力を使用できます。

> **Note**
>
> このガイドでは、JSON を使用して通知を設定する方法については説明していません。Amazon SNS を使用して Amazon SQS キューにメッセージを送信する方法の詳細については、Amazon SNS を Amazon SQS キューへメッセージ送信 を参照してください。Amazon SNS を使用して Lambda 関数を呼び出す方法の詳細については、Amazon SNS デベロッパーガイドの Lambda 関数の呼び出しを参照してください。

次の例は、CodeDeploy トリガーで使用可能な JSON 出力の構造を示しています。

インスタンスベースのトリガー用 JSON 出力のサンプル

```
{
    "region": "us-east-2",
    "accountId": "111222333444",
    "eventTriggerName": "trigger-group-us-east-instance-succeeded",
    "deploymentId": "d-75I7MBT7C",
```

```
    "instanceId": "arn:aws:ec2:us-east-2:444455556666:instance/i-496589f7",
    "lastUpdatedAt": "1446744207.564",
    "instanceStatus": "Succeeded",
    "lifecycleEvents": [
        {
            "LifecycleEvent": "ApplicationStop",
            "LifecycleEventStatus": "Succeeded",
            "StartTime": "1446744188.595",
            "EndTime": "1446744188.711"
        },
        {
            "LifecycleEvent": "BeforeInstall",
            "LifecycleEventStatus": "Succeeded",
            "StartTime": "1446744189.827",
            "EndTime": "1446744190.402"
        }
//More lifecycle events might be listed here
    ]
}
```

デプロイベースのトリガー用 JSON 出力のサンプル

```
{
    "region": "us-west-1",
    "accountId": "111222333444",
    "eventTriggerName": "Trigger-group-us-west-3-deploy-failed",
    "applicationName": "ProductionApp-us-west-3",
    "deploymentId": "d-75I7MBT7C",
    "deploymentGroupName": "dep-group-def-456",
    "createTime": "1446744188.595",
    "completeTime": "1446744190.402",
    "deploymentOverview": {
        "Failed": "10",
        "InProgress": "0",
        "Pending": "0",
        "Skipped": "0",
        "Succeeded": "0"
    },
    "status": "Failed",
    "errorInformation": {
        "ErrorCode": "IAM_ROLE_MISSING",
        "ErrorMessage": "IAM Role is missing for deployment group: dep-group-def-456"
    }
```

```
}
```

セキュリティ in AWS CodeDeploy

のクラウドセキュリティが最優先事項 AWS です。お客様は AWS 、セキュリティを最も重視する組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャを活用できます。

セキュリティは、 AWS お客様とお客様の間の責任共有です。責任共有モデルでは、これをクラウドのセキュリティおよびクラウド内のセキュリティとして説明しています。

- クラウドのセキュリティ – AWS クラウドで AWS サービスを実行するインフラストラクチャを保護する AWS 責任があります。 AWS また、 では、安全に使用できるサービスも提供しています。「AWS 」 コンプライアンスプログラムの一環として、サードパーティーの監査が定期的にセキュリティの有効性をテストおよび検証しています。 AWS CodeDeploy に適用されるコンプライアンスプログラムの詳細については、AWS「コンプライアンスプログラムによる対象範囲内のサービス」を参照してください。
- クラウド内のセキュリティ – お客様の責任は、使用する AWS サービスによって決まります。また、お客様は、お客様のデータの機密性、企業の要件、および適用可能な法律および規制などの他の要因についても責任を担います。

このドキュメントは、CodeDeploy を使用する際の責任共有モデルの適用方法を理解するのに役立ちます。以下のトピックでは、セキュリティおよびコンプライアンスの目的を達成するように CodeDeploy を設定する方法について説明します。また、CodeDeploy リソースのモニタリングや保護に役立つ他の AWS サービスの使用方法についても説明します。

トピック

- AWS CodeDeploy でのデータ保護
- の ID とアクセスの管理 AWS CodeDeploy
- CodeDeploy でのロギングとモニタリング
- AWS CodeDeploy のコンプライアンス検証
- AWS CodeDeploy の耐障害性
- インフラストラクチャセキュリティ in AWS CodeDeploy

AWS CodeDeploy でのデータ保護

責任 AWS 共有モデル、 AWS CodeDeploy でのデータ保護に適用されます。このモデルで説明されているように、 AWS はすべての を実行するグローバルインフラストラクチャを保護する責任

があります AWS クラウド。ユーザーは、このインフラストラクチャでホストされるコンテンツに対する管理を維持する責任があります。また、使用する AWS のサービス のセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、「Data Privacy FAQChina」を参照してください。 欧州におけるデータ保護に関する情報については、General Data Protection Regulation (GDPR) Center を参照してください。

データ保護の目的で、認証情報を保護し AWS アカウント 、 AWS IAM アイデンティティセンターまたは AWS Identity and Access Management (IAM) を使用して個々のユーザーを設定することをお勧めします。この方法により、各自のジョブを遂行するために必要なアクセス許可のみが各ユーザーに付与されます。また、次の方法でデータを保護することもお勧めします:

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。TLS 1.2 は必須ですが、TLS 1.3 を推奨します。
- で API とユーザーアクティビティのログ記録を設定します AWS CloudTrail。CloudTrail 証跡を使用して AWS アクティビティをキャプチャする方法については、「 AWS CloudTrail ユーザーガイド」のCloudTrail 証跡の使用」を参照してください。
- AWS 暗号化ソリューションと、 内のすべてのデフォルトのセキュリティコントロールを使用します AWS のサービス。
- Amazon Macie などの高度な管理されたセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは API AWS を介して にアクセスするときに FIPS 140-3 検証済み暗号化モジュールが必要な場合は、FIPS エンドポイントを使用します。利用可能な FIPS エンドポイントの詳細については、「連邦情報処理規格 (FIPS) 140-3」を参照してください。

お客様の E メールアドレスなどの極秘または機密情報を、タグ、または [名前] フィールドなどの自由形式のテキストフィールドに含めないことを強くお勧めします。これは、コンソール、API、または SDK を使用して CodeDeploy AWS CLIまたは他の AWS のサービス を使用する場合も同様です。AWS SDKs タグ、または名前に使用される自由記述のテキストフィールドに入力したデータは、請求または診断ログに使用される場合があります。外部サーバーに URL を提供する場合、そのサーバーへのリクエストを検証できるように、認証情報を URL に含めないことを強くお勧めします。

ネットワーク間のトラフィックのプライバシー

CodeDeploy は、EC2 インスタンス、 Lambda 関数、Amazon ECS、オンプレミスサーバーをサポートするフルマネージド型のデプロイサービスです。EC2 インスタンスとオンプレミスサーバーに関しては、ホストベースのエージェントは TLS を使用して CodeDeploy と通信します。

現在、エージェントからサービスへの通信にはアウトバウンドインターネット接続が必要で、それによってエージェントは、公開された CodeDeploy および Amazon S3 サービスエンドポイントと通信できるようになります。仮想プライベートクラウドでは、インターネットゲートウェイ、企業ネットワークへのサイト間 VPN 接続、または直接接続を使用してこれを実現できます。

CodeDeploy エージェントは HTTP プロキシをサポートします。

Amazon VPC エンドポイントは AWS PrivateLink、特定のリージョンで CodeDeploy で使用できます。詳細については、「Amazon Virtual Private Cloud で CodeDeploy を使用」を参照してください。

> **Note**
>
> CodeDeploy エージェントは、Amazon EC2 オンプレミスのコンピューティングプラットフォームにデプロイする場合にのみ必要です。エージェントは、Amazon ECS または AWS Lambda コンピューティングプラットフォームを使用するデプロイには必要ありません。

保管中の暗号化

カスタマーコードは CodeDeploy に保存されません 。デプロイサービスとして、CodeDeploy は EC2 インスタンスまたはオンプレミスサーバーで稼働している CodeDeploy エージェントにコマンドを送信します。次に、CodeDeploy エージェントは TLS を使用してコマンドを実行します。デプロイ、デプロイ設定、デプロイグループ、アプリケーション、およびアプリケーションリビジョンのサービスモデルデータは Amazon DynamoDB に保存され AWS 所有のキー、CodeDeploy が所有および管理する を使用して保管時に暗号化されます。詳細については、AWS 所有のキー を参照してください。

転送中の暗号化

CodeDeploy エージェントは、CodeDeploy とのすべての通信をポート 443 を介して開始します。エージェントは CodeDeploy をポーリングしてコマンドをリッスンします。CodeDeploy エージェントはオープンソースです。すべてのサービスからサービスへの通信およびクライアントからサービスへの通信は、TLS を使用して転送中に暗号化されます。これにより、CodeDeploy と Amazon S3 などの他のサービス間で転送されるお客様データが保護されます。

暗号化キーの管理

お客様が管理する必要のある暗号化キーはありません。CodeDeploy サービスモデルデータは AWS 所有のキー、CodeDeploy が所有および管理する を使用して暗号化されます。詳細については、AWS 所有のキー を参照してください。

の ID とアクセスの管理 AWS CodeDeploy

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御 AWS のサービス するのに役立つ です。IAM 管理者は、誰を認証 (サインイン) し、誰に CodeDeploy リソースの使用を承認する (アクセス許可を付与する) かを管理します。IAM は、追加料金なしで使用できる AWS のサービス です。

トピック

- オーディエンス
- アイデンティティを使用した認証
- ポリシーを使用したアクセス管理
- AWS CodeDeploy が IAM と連携する方法
- AWS CodeDeploy の マネージド (事前定義) ポリシー
- AWS マネージドポリシーへの CodeDeploy の更新
- AWS CodeDeploy アイデンティティベースのポリシーの例
- AWS CodeDeploy ID とアクセスのトラブルシューティング
- CodeDeploy アクセス許可 リファレンス
- サービス間での不分別な代理処理の防止

オーディエンス

AWS Identity and Access Management (IAM) の使用方法は、ロールによって異なります。

- サービスユーザー - 機能にアクセスできない場合は、管理者にアクセス許可をリクエストします (「AWS CodeDeploy ID とアクセスのトラブルシューティング」を参照)。
- サービス管理者 - ユーザーアクセスを決定し、アクセス許可リクエストを送信します (「AWS CodeDeploy が IAM と連携する方法」を参照)

- IAM 管理者 - アクセスを管理するためのポリシーを作成します (「AWS CodeDeploy アイデンティティベースのポリシーの例」を参照)

アイデンティティを使用した認証

認証とは、ID 認証情報 AWS を使用して にサインインする方法です。、IAM ユーザー AWS アカウントのルートユーザー、または IAM ロールを引き受けることで認証される必要があります。

AWS IAM アイデンティティセンター (IAM Identity Center)、シングルサインオン認証、Google/Facebook 認証情報などの ID ソースからの認証情報を使用して、フェデレーティッド ID としてサインインできます。サインインの詳細については、「AWS サインイン ユーザーガイド」の「AWS アカウントにサインインする方法」を参照してください。

プログラムによるアクセスの場合、 は SDK と CLI AWS を提供してリクエストを暗号化して署名します。詳細については、「IAM ユーザーガイド」の「API リクエストに対するAWS 署名バージョン 4」を参照してください。

AWS アカウント ルートユーザー

を作成するときは AWS アカウント、まず、すべての AWS のサービス および リソースへの完全なアクセス権を持つ AWS アカウント root ユーザーと呼ばれる 1 つのサインイン ID から始めます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザー認証情報を必要とするタスクについては、「IAM ユーザーガイド」の「ルートユーザー認証情報が必要なタスク」を参照してください。

ユーザーとグループ

IAM ユーザーは、1 人のユーザーまたは 1 つのアプリケーションに対して特定のアクセス許可を持つ ID です。長期認証情報を持つ IAM ユーザーの代わりに一時的な認証情報を使用することをお勧めします。詳細については、IAM ユーザーガイドの「ID プロバイダーとのフェデレーションを使用してにアクセスする必要がある AWS」を参照してください。

IAM グループは、IAM ユーザーの集合を指定し、大量のユーザーに対するアクセス許可の管理を容易にします。詳細については、「IAM ユーザーガイド」の「IAM ユーザーに関するユースケース」を参照してください。

IAM ロール

IAM ロールは、特定のアクセス許可を持つアイデンティであり、一時的な認証情報を提供します。ユーザーから IAM ロール (コンソール) に切り替えるか、 または API オペレーションを呼び出すこ

とで、ロールを引き受けることができます。 AWS CLI AWS 詳細については、「IAM ユーザーガイド」の「ロールを引き受けるための各種方法」を参照してください。

IAM ロールは、フェデレーションユーザーアクセス、一時的な IAM ユーザーのアクセス許可、クロスアカウントアクセス、クロスサービスアクセス、および Amazon EC2 で実行するアプリケーションに役立ちます。詳細については、IAM ユーザーガイド の IAM でのクロスアカウントリソースアクセス を参照してください。

ポリシーを使用したアクセス管理

でアクセスを制御する AWS には、ポリシーを作成し、ID AWS またはリソースにアタッチします。ポリシーは、アイデンティティまたはリソースに関連付けられたときにアクセス許可を定義します。は、プリンシパルがリクエストを行うときにこれらのポリシー AWS を評価します。ほとんどのポリシーは JSON ドキュメント AWS として に保存されます。JSON ポリシードキュメントの詳細については、「IAM ユーザーガイド」の「JSON ポリシー概要」を参照してください。

管理者は、ポリシーを使用して、どのプリンシパルがどのリソースに対して、どのような条件でアクションを実行できるかを定義することで、誰が何にアクセスできるかを指定します。

デフォルトでは、ユーザーやロールにアクセス許可はありません。IAM 管理者は IAM ポリシーを作成してロールに追加し、このロールをユーザーが引き受けられるようにします。IAM ポリシーは、オペレーションの実行方法を問わず、アクセス許可を定義します。

アイデンティティベースのポリシー

アイデンティティベースのポリシーは、アイデンティティ (ユーザー、グループ、またはロール) にアタッチできる JSON アクセス許可ポリシードキュメントです。これらのポリシーは、アイデンティティがどのリソースに対してどのような条件下でどのようなアクションを実行できるかを制御します。アイデンティティベースポリシーの作成方法については、IAM ユーザーガイド の カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する を参照してください。

アイデンティティベースのポリシーは、インラインポリシー (単一の ID に直接埋め込む) または管理ポリシー (複数の ID にアタッチされたスタンドアロンポリシー) にすることができます。マネージドポリシーとインラインポリシーのいずれかを選択する方法については、「IAM ユーザーガイド」の「管理ポリシーとインラインポリシーのいずれかを選択する」を参照してください。

その他のポリシータイプ

AWS は、より一般的なポリシータイプによって付与されるアクセス許可の最大数を設定できる追加のポリシータイプをサポートしています。

- アクセス許可の境界 – アイデンティティベースのポリシーで IAM エンティティに付与することのできるアクセス許可の数の上限を設定します。詳細については、「IAM ユーザーガイド」の「IAM エンティティのアクセス許可境界」を参照してください。
- サービスコントロールポリシー (SCP) - AWS Organizations内の組織または組織単位の最大のアクセス許可を指定します。詳細については、「AWS Organizations ユーザーガイド」の「サービスコントロールポリシー」を参照してください。
- リソースコントロールポリシー (RCP) – は、アカウント内のリソースで利用できる最大数のアクセス許可を定義します。詳細については、「AWS Organizations ユーザーガイド」の「リソースコントロールポリシー (RCP)」を参照してください。
- セッションポリシー – ロールまたはフェデレーションユーザーの一時セッションを作成する際にパラメータとして渡される高度なポリシーです。詳細については、「IAM ユーザーガイド」の「セッションポリシー」を参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成されるアクセス許可を理解するのがさらに難しくなります。が複数のポリシータイプが関与する場合にリクエストを許可するかどうか AWS を決定する方法については、「IAM ユーザーガイド」の「ポリシー評価ロジック」を参照してください。

AWS CodeDeploy が IAM と連携する方法

IAM を使用して CodeDeploy へのアクセスを管理する前に、CodeDeploy で使用できる IAM 機能について理解しておきましょう。詳細については、IAM ユーザーガイド の「IAM と連携するAWS サービス」を参照してください。

トピック

- CodeDeploy アイデンティティベースのポリシー
- CodeDeploy のリソースベースのポリシー
- CodeDeploy タグに基づく認可
- CodeDeploy IAM ロール

CodeDeploy アイデンティティベースのポリシー

IAM のアイデンティティベースポリシーでは、許可または拒否するアクションとリソース、およびアクションが許可または拒否される条件を指定できます。CodeDeploy はアクション、リソース、お

よび条件キーをサポートします。JSON ポリシーで使用する要素については、IAM ユーザーガイド 内の「IAM JSON ポリシーの要素のリファレンス」を参照してください。

アクション

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

JSON ポリシーの `Action` 要素は、ポリシー内でアクセスを許可または拒否するために使用できるアクションを記述します。関連付けられたオペレーションを実行するためのアクセス許可を付与するには、ポリシーにアクションを含めます。

CodeDeploy のポリシーアクションは、アクションの前に `codedeploy:` プレフィックスを使用します。例えば、`codedeploy:GetApplication` 許可は、`GetApplication` オペレーションを実行するためのアクセス許可をユーザーに付与します。ポリシーステートメントには`Action` または `NotAction` 要素を含める必要があります。CodeDeploy は、このサービスで実行できるタスクを記述する独自のアクションのセットを定義します。

単一のステートメントに複数のアクションを指定するには次のようにコンマで区切ります。

```
"Action": [
      "codedeploy:action1",
      "codedeploy:action2"
```

ワイルドカード (*) を使用して複数アクションを指定できます。例えば、`Describe` という単語で始まるすべてのアクションを指定するには、次のアクションを含めます。

```
"Action": "ec2:Describe*"
```

CodeDeploy アクションのリストを表示するには、IAM ユーザーガイド の 「AWS CodeDeployによって定義されたアクション」を参照してください。

すべての CodeDeploy API アクションとそれらが適用されるリソースの表については、CodeDeploy アクセス許可 リファレンス を参照してください。

リソース

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

Resource JSON ポリシー要素はアクションを適用するオブジェクトを指定します。ベストプラクティスとして、Amazon リソースネーム (ARN) を使用してリソースを指定します。リソースレベルのアクセス許可をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

```
"Resource": "*"
```

例えば、以下のように ARN を使用して、ステートメント内でデプロイグループ (*myDeploymentGroup*) を指定できます。

```
"Resource": "arn:aws:codedeploy:us-
west-2:123456789012:deploymentgroup:myApplication/myDeploymentGroup"
```

以下のようにワイルドカード文字 (*) を使用して、特定のアカウントに属するすべてのデプロイグループを指定することもできます。

```
"Resource": "arn:aws:codedeploy:us-west-2:123456789012:deploymentgroup:*"
```

すべてのリソースを指定する場合、または API アクションが ARN をサポートしていない場合は、以下の要領で、Resource エレメント内でワイルドカード文字 (*) を使用します。

```
"Resource": "*"
```

一部の CodeDeploy API アクションは複数のリソースを受け入れます (例: BatchGetDeploymentGroups)。単一のステートメントに複数のリソースを指定するには、以下のようにコンマで ARN を区切ります。

```
"Resource": ["arn1", "arn2"]
```

CodeDeploy には、CodeDeploy リソースを操作するための一連のオペレーションが用意されています。使用可能なオペレーションのリストについては、「CodeDeploy アクセス許可 リファレンス」を参照してください。

CodeDeploy リソースタイプとその ARN のリストを表示するには、IAM ユーザーガイド の AWS CodeDeploy「で定義されるリソース」 を参照してください。各リソースの ARN を指定できるアクションの詳細については、AWS CodeDeployで定義されたアクション を参照してください。

CodeDeploy リソースとオペレーション

CodeDeploy では、プライマリリソースはデプロイグループです。ポリシーで Amazon リソースネーム (ARN) を使用して、ポリシーを適用するリソースを識別します。CodeDeploy は、アプリケーション、デプロイ設定、インスタンスなど、デプロイグループで使用できる他のリソースをサポートします。これらは サブリソースと呼ばれます。これらのリソースとサブリソースには、一意の ARN が関連付けられています。詳細については、Amazon Web Services 全般のリファレンス の「Amazon リソースネーム (ARN)」を参照してください。

リソースタイプ	ARN 形式
デプロイグループ	`arn:aws:codedeploy:` *region*`:`*account-id*` :deploymentgroup:` *application-name* `/`*deployment-group-name*
アプリケーション	`arn:aws:codedeploy:` *region*`:`*account-id*` :application:` *application-name*
Deployment configuration	`arn:aws:codedeploy:` *region*`:`*account-id*` :deploymentconfig:` *deployment-configuration-name*
インスタンス	`arn:aws:codedeploy:` *region*`:`*account-id*` :instance/` *instance-ID*
すべての CodeDeploy リソース	`arn:aws:codedeploy:*`
特定のリージョンの特定アカウントが所有するすべての CodeDeploy リソース	`arn:aws:codedeploy:` *region*`:`*account-id*` :*`

> **Note**
>
> のほとんどのサービスは、ARN でコロン (:) またはスラッシュ (/) を同じ文字として AWS 扱います。 ARNs ただし、CodeDeploy では、リソースパターンとルールで完全一致が使用されます。イベントパターンを作成するときは、リソースの ARN 構文と一致するように正しい ARN 文字を使用してください。

条件キー

CodeDeploy にはサービス固有条件キーがありませんが、いくつかのグローバル条件キーの使用がサポートされています。条件キーの詳細については、IAM ユーザーガイド の「AWS グローバル条件コンテキストキー」 を参照してください。

例

CodeDeploy アイデンティティベースのポリシーの例については、AWS CodeDeploy アイデンティティベースのポリシーの例 を参照してください。

CodeDeploy のリソースベースのポリシー

CodeDeploy では、リソースベースのポリシーはサポートされていません。詳細なリソースベースのポリシーページの例を表示するには、「リソースベースのポリシーの使用 AWS Lambda」を参照してください。

CodeDeploy タグに基づく認可

CodeDeploy は、リソースのタグ付けやタグに基づいたアクセスの制御をサポートしていません。

CodeDeploy IAM ロール

IAM ロールは、特定のアクセス許可を持つ AWS アカウントのエンティティです。

CodeDeploy での一時的な認証情報の使用

一時的な認証情報を使用して、フェデレーションでサインインする、IAM 役割を引き受ける、またはクロスアカウント役割を引き受けることができます。一時的なセキュリティ認証情報を取得するには、AssumeRole や GetFederationToken などの AWS STS API オペレーションを呼び出します。

CodeDeploy は、一時的な認証情報の使用をサポートしています。

サービスリンクロール

CodeDeploy は、サービスにリンクされたロールをサポートしていません。

サービス役割

この機能により、ユーザーに代わってサービスがサービスロールを引き受けることが許可されます。この役割により、サービスがお客様に代わって他のサービスのリソースにアクセスし、アクションを完了することが許可されます。サービスロールは AWS アカウントに表示され、アカウントによって所有されます。つまり、ユーザーは、このロールのアクセス許可を変更できます。ただし、それにより、サービスの機能が損なわれる場合があります。

CodeDeploy はサービスロールをサポートします。

CodeDeploy での IAM ロールの選択

CodeDeploy でデプロイグループリソースを作成する場合、 CodeDeploy がユーザーの代わりに自動的に Amazon EC2 にアクセスすることを許可するロールを選択する必要があります。サービスロールあるいはサービスにリンクされたロールを以前に作成している場合、CodeDeploy は選択できるロールのリストを表示します。EC2 インスタンスの起動と停止へのアクセスを許可するロールを選択することが重要です。

AWS CodeDeploy の マネージド (事前定義) ポリシー

AWS は、 によって作成および管理されるスタンドアロン IAM ポリシーを提供することで、多くの一般的なユースケースに対処します AWS。これらの AWS管理ポリシーは、一般的なユースケースのアクセス許可を付与するため、どのアクセス許可が必要かを調査する必要がなくなります。詳細については、「IAM ユーザーガイド」の「AWS マネージドポリシー」を参照してください。

トピック

- CodeDeploy の AWS マネージドポリシーのリスト
- CodeDeploy のマネージドポリシーと通知

CodeDeploy の AWS マネージドポリシーのリスト

アカウントのユーザーにアタッチできる以下の AWS 管理ポリシーは、CodeDeploy に固有です。

- `AWSCodeDeployFullAccess`: CodeDeploy への全面的なアクセス権を付与します。

> **Note**
>
> AWSCodeDeployFullAccess は、Amazon EC2 や Amazon S3 などのアプリケーションをデプロイするために必要な他のサービスの操作にアクセス権限を提供しません。CodeDeploy に固有の操作にのみアクセス権限を提供します。

- `AWSCodeDeployDeployerAccess`: リビジョンを登録してデプロイする権限を許可します。

- `AWSCodeDeployReadOnlyAccess`: CodeDeploy への読み取り専用アクセス権を付与します。

- `AWSCodeDeployRole`: CodeDeploy に以下を許可します
  - インスタンスのタグを読み取る、または Amazon EC2 Auto Scaling グループ名により Amazon EC2 インスタンスを識別する
  - Amazon EC2 Auto Scaling グループ、ライフサイクルフック、スケーリングポリシー、ウォームプールの機能の読み取り、作成、更新、削除を行う
  - Amazon SNS トピックに情報を公開
  - Amazon CloudWatch アラームに関する基本的な情報を取得
  - Elastic Load Balancing サービスでのリソースの読み取りと更新

  ポリシーには、次の規定が含まれます。

  JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "autoscaling:CompleteLifecycleAction",
        "autoscaling:DeleteLifecycleHook",
        "autoscaling:DescribeAutoScalingGroups",
        "autoscaling:DescribeLifecycleHooks",
        "autoscaling:PutLifecycleHook",
        "autoscaling:RecordLifecycleActionHeartbeat",
        "autoscaling:CreateAutoScalingGroup",
```

```
        "autoscaling:CreateOrUpdateTags",
        "autoscaling:UpdateAutoScalingGroup",
        "autoscaling:EnableMetricsCollection",
        "autoscaling:DescribePolicies",
        "autoscaling:DescribeScheduledActions",
        "autoscaling:DescribeNotificationConfigurations",
        "autoscaling:SuspendProcesses",
        "autoscaling:ResumeProcesses",
        "autoscaling:AttachLoadBalancers",
        "autoscaling:AttachLoadBalancerTargetGroups",
        "autoscaling:PutScalingPolicy",
        "autoscaling:PutScheduledUpdateGroupAction",
        "autoscaling:PutNotificationConfiguration",
        "autoscaling:DescribeScalingActivities",
        "autoscaling:DeleteAutoScalingGroup",
        "autoscaling:PutWarmPool",
        "ec2:DescribeInstances",
        "ec2:DescribeInstanceStatus",
        "ec2:TerminateInstances",
        "tag:GetResources",
        "sns:Publish",
        "cloudwatch:DescribeAlarms",
        "cloudwatch:PutMetricAlarm",
        "elasticloadbalancing:DescribeLoadBalancers",
        "elasticloadbalancing:DescribeLoadBalancerAttributes",
        "elasticloadbalancing:DescribeInstanceHealth",
        "elasticloadbalancing:RegisterInstancesWithLoadBalancer",
        "elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
        "elasticloadbalancing:DescribeTargetGroups",
        "elasticloadbalancing:DescribeTargetGroupAttributes",
        "elasticloadbalancing:DescribeTargetHealth",
        "elasticloadbalancing:RegisterTargets",
        "elasticloadbalancing:DeregisterTargets"
      ],
      "Resource": "*"
    }
  ]
}
```

- AWSCodeDeployRoleForLambda: CodeDeploy に、デプロイに必要な AWS Lambda および他のリソースへのアクセス許可を付与します。

- `AWSCodeDeployRoleForECS`: CodeDeploy に Amazon ECS およびデプロイに必要なその他のリソースへのアクセス許可を付与します。

- `AWSCodeDeployRoleForECSLimited`: CodeDeploy に Amazon ECS およびデプロイに必要なその他のリソースへのアクセス許可を付与します。ただし、下記を例外とします。
  - AppSpec ファイルの `hooks` セクションでは、名前が `CodeDeployHook_` で始まる Lambda 関数のみを使用できます。詳細については、「Amazon ECS のデプロイ向けの AppSpec の「hooks」セクション」を参照してください。
  - S3 バケットへのアクセスは、`UseWithCodeDeploy` の値の登録タグを持つ `true` で S3 バケットに限定されます。詳細については、「オブジェクトのタグ付け」を参照してください。
- `AmazonEC2RoleforAWSCodeDeployLimited`: CodeDeploy Amazon S3 バケット内のオブジェクトを取得および一覧表示のアクセス権限を CodeDeploy に付与します。ポリシーには、次の規定が含まれます。

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion",
                "s3:ListBucket"
            ],
            "Resource": "arn:aws:s3:::*/CodeDeploy/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
```

```
                    "s3:ExistingObjectTag/UseWithCodeDeploy": "true"
                }
            }
        }
    ]
}
```

デプロイプロセスの一部に対するアクセス許可は、CodeDeploy を代行する他の 2 つのロールタイプに付与されます。

- IAM インスタンスプロファイルは、Amazon EC2 インスタンス にアタッチする IAM ロールです。このプロファイルには、アプリケーションが保存される Amazon S3 バケットまたは GitHub リポジトリへのアクセスに必要な権限が含まれます。詳細については、「ステップ 4: Amazon EC2 インスタンス用の IAM インスタンスプロファイルを作成する」を参照してください。
- サービスロールは、 AWS リソースにアクセスできるように AWS サービスにアクセス許可を付与する IAM ロールです。サービスロールにアタッチするポリシーによって、サービスがアクセスできる AWS リソースと、それらのリソースで実行できるアクションが決まります。CodeDeploy では、サービスロールは、以下の目的で使用されます。
  - インスタンスに適用されているタグやインスタンスに関連付けられている Amazon EC2 Auto Scaling グループ名を読み取ることができます。これにより、CodeDeploy はアプリケーションをデプロイできるインスタンスを識別できます。
  - インスタンス、Amazon EC2 Auto Scaling グループ、および Elastic Load Balancing ロードバランサーでオペレーションを実行するには。
  - 指定したデプロイまたはインスタンスイベントが発生したときに通知を送信できるように、Amazon SNS トピックに情報を公開すること。
  - CloudWatch アラームに関する情報を取得して、デプロイのアラームモニタリングを設定します。

  その他の詳細については、「ステップ 2: CodeDeployのサービスのロールを作成する」を参照してください。

カスタム IAM ポリシーを作成して、CodeDeploy アクションとリソースのアクセス許可を付与することもできます。こうしたカスタムポリシーを IAM ロールにアタッチし、ロールをアクセス許可が必要なユーザーまたはグループに割り当てます。

CodeDeploy のマネージドポリシーと通知

CodeDeploy は、デプロイへの重要な変更をユーザーに知らせるための通知をサポートしています。CodeDeploy のマネージドポリシーには、通知機能のポリシーステートメントが含まれます。詳細については、通知とは を参照してください。

フルアクセスマネージドポリシー内の通知へのアクセス許可

AWSCodeDeployFullAccess マネージドポリシーには、通知へのフルアクセスを許可する次のステートメントが含まれています。このマネージドポリシーが適用されたユーザーは、Amazon SNS 通知関連トピックの作成と管理、トピックへのユーザーの登録および登録解除、そして通知ルールのターゲットとして選択するトピックの一覧表示を行うことができます。

```
    {
        "Sid": "CodeStarNotificationsReadWriteAccess",
        "Effect": "Allow",
        "Action": [
            "codestar-notifications:CreateNotificationRule",
            "codestar-notifications:DescribeNotificationRule",
            "codestar-notifications:UpdateNotificationRule",
            "codestar-notifications:DeleteNotificationRule",
            "codestar-notifications:Subscribe",
            "codestar-notifications:Unsubscribe"
        ],
        "Resource": "*",
        "Condition" : {
            "ArnLike" : {"codestar-notifications:NotificationsForResource" :
 "arn:aws:codedeploy:*:*:application:*"}
        }
    },
    {
        "Sid": "CodeStarNotificationsListAccess",
        "Effect": "Allow",
        "Action": [
            "codestar-notifications:ListNotificationRules",
            "codestar-notifications:ListTargets",
            "codestar-notifications:ListTagsforResource"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CodeStarNotificationsSNSTopicCreateAccess",
        "Effect": "Allow",
```

```
        "Action": [
            "sns:CreateTopic",
            "sns:SetTopicAttributes"
        ],
        "Resource": "arn:aws:sns:*:*:codestar-notifications*"
    },
    {
      "Sid" : "CodeStarNotificationsChatbotAccess",
      "Effect" : "Allow",
      "Action" : [
        "chatbot:DescribeSlackChannelConfigurations"
      ],
      "Resource" : "*"
    },
    {
        "Sid": "SNSTopicListAccess",
        "Effect": "Allow",
        "Action": [
            "sns:ListTopics"
        ],
        "Resource": "*"
    }
```

読み取り専用マネージドポリシー内の通知へのアクセス許可

AWSCodeDeployReadOnlyAccess マネージドポリシーには、通知への読み取り専用アクセスを許可する以下のステートメントが含まれています。このマネージドポリシーが適用されたユーザーは、リソースの通知を表示することはできますが、リソースの作成や管理、リソースへのサブスクライブを行うことはできません。

```
   {
        "Sid": "CodeStarNotificationsPowerUserAccess",
        "Effect": "Allow",
        "Action": [
            "codestar-notifications:DescribeNotificationRule"
        ],
        "Resource": "*",
        "Condition" : {
            "ArnLike" : {"codestar-notifications:NotificationsForResource" :
 "arn:aws:codedeploy:*:*:application:*"}
        }
    },
    {
```

```
        "Sid": "CodeStarNotificationsListAccess",
        "Effect": "Allow",
        "Action": [
            "codestar-notifications:ListNotificationRules"
        ],
        "Resource": "*"
    }
```

その他のマネージドポリシーの通知に関連するアクセス許可

AWSCodeDeployDeployerAccess マネージドポリシーには、ユーザーがリソースの通知を作成、更新、サブスクライブ、表示することを許可する以下のステートメントが含まれていますが、削除することはできません。このマネージドポリシーが適用されたユーザーは、通知の Amazon SNS トピックを作成および管理することもできます。

このポリシーには以下を実行するための許可が含まれます。

- codestar-notifications:CreateNotificationRule – プリンシパルが通知を作成できるようにします。
- codestar-notifications:DescribeNotificationRule — プリンシパルが通知に関する情報を取得できるようにします。
- codestar-notifications:UpdateNotificationRule – プリンシパルが通知を更新できるようにします。
- codestar-notifications:Subscribe – プリンシパルが通知更新をサブスクライブできるようにします。
- codestar-notifications:Unsubscribe – プリンシパルが通知更新のサブスクリプションを解除できるようにします。
- codestar-notifications:ListNotificationRules – プリンシパルが通知ルールのリストを取得できるようにします。
- codestar-notifications:ListTargets – プリンシパルがターゲットのリストを取得できるようにします。
- codestar-notifications:ListTagsforResource – プリンシパルがタグのリストを取得できるようにします。
- codestar-notifications:ListEventTypes – プリンシパルがイベントタイプのリストを取得できるようにします。
- chatbot:DescribeSlackChannelConfiguration - プリンシパルがスタックチャネル設定に関する情報を取得できるようにします。

- sns:ListTopics – プリンシパルが通知用の Amazon SNS トピックのリストを取得できるようにします。

```
  {
     "Sid" : "CodeStarNotificationsReadWriteAccess",
     "Effect" : "Allow",
     "Action" : [
       "codestar-notifications:CreateNotificationRule",
       "codestar-notifications:DescribeNotificationRule",
       "codestar-notifications:UpdateNotificationRule",
       "codestar-notifications:Subscribe",
       "codestar-notifications:Unsubscribe"
     ],
     "Resource" : "*",
     "Condition" : {
       "ArnLike" : {
         "codestar-notifications:NotificationsForResource" :
"arn:aws:codedeploy:*:*:application:*"
       }
     }
   },
   {
     "Sid" : "CodeStarNotificationsListAccess",
     "Effect" : "Allow",
     "Action" : [
       "codestar-notifications:ListNotificationRules",
       "codestar-notifications:ListTargets",
       "codestar-notifications:ListTagsforResource",
       "codestar-notifications:ListEventTypes"
     ],
     "Resource" : "*"
   },
   {
     "Sid" : "CodeStarNotificationsChatbotAccess",
     "Effect" : "Allow",
     "Action" : [
       "chatbot:DescribeSlackChannelConfigurations"
     ],
     "Resource" : "*"
   },
   {
     "Sid" : "SNSTopicListAccess",
```

```
      "Effect" : "Allow",
      "Action" : [
        "sns:ListTopics"
      ],
      "Resource" : "*"
    }
```

詳細については、「AWS CodeStar Notifications の Identity and Access Management」を参照してください。

AWS マネージドポリシーへの CodeDeploy の更新

このサービスがこれらの変更の追跡を開始してからの CodeDeploy の AWS 管理ポリシーの更新に関する詳細を表示します。このページへの変更に関する自動アラートを受けるには、CodeDeploy ドキュメント履歴 の RSS フィードに登録してください。

変更	説明	日付
AWSCodeDeployDeployerAccess マネージドポリシ - 既存のポリシーへの更新	IAM ポリシー検証の変更をサポートするように codestar-notifications:NotificationsForResource アクションを更新しました。元のリソース arn:aws:codedeploy:* は arn:aws:codedeploy:*:*:application:* に更新されました。 このポリシーの詳細については、その他のマネージドポリシーの通知に関連するアクセス許可 を参照してください。	2024 年 12 月 16 日
AWSCodeDeployReadOnlyAccess マネージドポリシ - 既存のポリシーへの更新	IAM ポリシー検証の変更をサポートするように codestar-notifications:NotificationsForResource アクションを更新	2024 年 12 月 16 日

変更	説明	日付
	しました。元のリソース arn:aws:codedeploy:* は arn:aws:codedeploy:*:*:application:* に更新されました。 このポリシーの詳細については、読み取り専用マネージドポリシー内の通知へのアクセス許可 を参照してください。	
AWSCodeDeployFullAccess マネージドポリシー - 既存のポリシーへの更新	IAM ポリシー検証の変更をサポートするように codestar-notifications:NotificationsForResource アクションを更新しました。元のリソース arn:aws:codedeploy:* は arn:aws:codedeploy:*:*:application:* に更新されました。 このポリシーの詳細については、フルアクセスマネージドポリシー内の通知へのアクセス許可 を参照してください。	2024 年 12 月 16 日

変更	説明	日付
AWSCodeDeployRole マネージドポリシ - 既存のポリシーへの更新	Elastic Load Balancing の変更をサポートする elasticloadbalancing:DescribeLoadBalancerAttributes および elasticloadbalancing:DescribeTargetGroupAttributes アクションをポリシーステートメントに追加しました。 このポリシーの詳細については、AWSCodeDeployRole を参照してください。	2023 年 8 月 16 日
AWSCodeDeployFullAccess マネージドポリシ - 既存のポリシーへの更新	通知の変更をサポートする chatbot:ListMicrosoftTeamsChannelConfigurations アクションをポリシーステートメントに追加しました。 このポリシーの詳細については、AWSCodeDeployRole を参照してください。	2023 年 5 月 11 日

変更	説明	日付
AWSCodeDeployRole マネージドポリシー - 既存のポリシーへの更新	Amazon EC2 Auto Scaling の認可変更をサポートするために、 autoscaling:CreateOrUpdateTags アクションをポリシーステートメントに追加しました。 このポリシーの詳細については、AWSCodeDeployRole を参照してください。	2023 年 2 月 3 日
AmazonEC2RoleforAWSCodeDeployLimited マネージドポリシー - 既存のポリシーへの更新	s3:ListBucket アクションを s3:ExistingObjectTag/UseWithCodeDeploy 条件を含むポリシーステートメントから削除しました。 このポリシーの詳細については、AmazonEC2RoleforAWSCodeDeployLimited を参照してください。	2021 年 11 月 22 日
AWSCodeDeployRole マネージドポリシー - 既存のポリシーへの更新	ブルー/グリーンデプロイ に関して Amazon EC2 Auto Scaling グループへのウォームプールの追加をサポートする autoscaling:PutWarmPool アクションが追加されました。 不要な重複アクションを削除しました。	2021 年 5 月 18 日

変更	説明	日付
CodeDeploy が変更の追跡をスタート	CodeDeploy は AWS 、管理ポリシーの変更の追跡を開始しました。	2021 年 5 月 18 日

AWS CodeDeploy アイデンティティベースのポリシーの例

デフォルトでは、ユーザーには、CodeDeploy リソースを作成または変更するアクセス許可はありません。また、 AWS マネジメントコンソール、 AWS CLI、または AWS API を使用してタスクを実行することはできません。必要な指定されたリソースに対して API オペレーションを実行するためのアクセス許可を IAM ロールに付与する IAM ポリシーを作成する必要があります。続いて、それらのアクセス許可が必要なユーザーまたはグループにその IAM ロールをアタッチします。

これらの JSON ポリシードキュメント例を使用して IAM のアイデンティティベースのポリシーを作成する方法については、IAM ユーザーガイドの「JSON タブでのポリシーの作成」を参照してください。

CodeDeploy 内では、アイデンティティベースのポリシーは、デプロイプロセスに関連するさまざまなリソースに対するアクセス権限を管理するために使用されます。次のリソースタイプへのアクセスをコントロールできます。

- アプリケーションおよびアプリケーションリビジョン。
- デプロイ。
- デプロイ設定。
- インスタンスとオンプレミスインスタンス。

リソースポリシーによって制御される機能は、次の表にあるように、リソースタイプによって異なります。

リソースタイプ:	機能
すべて	リソースの詳細の閲覧および一覧表示
アプリケーション	リソースの作成
デプロイ設定	リソースの削除

リソースタイプ:	機能
デプロイグループ	
デプロイ	デプロイの作成 デプロイの停止
アプリケーションリビジョン	アプリケーションリビジョンの登録
アプリケーション デプロイグループ	更新されたリソース
オンプレミスインスタンス	インスタンスにタグを追加する インスタンスからタグを削除する インスタンスを登録する インスタンスを登録解除する

次の例では、**us-west-2** リージョンの **WordPress_App** という名前のアプリケーションに関連付けられている **WordPress_DepGroup** という名前のデプロイグループを削除することをユーザーに許可するアクセス許可ポリシーを示しています。

JSON

```
{
  "Version":"2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "codedeploy:DeleteDeploymentGroup"
      ],
      "Resource" : [
        "arn:aws:codedeploy:us-west-2:111122223333:deploymentgroup:WordPress_App/
WordPress_DepGroup"
      ]
    }
```

```
  ]
}
```

トピック

- カスタマーマネージドポリシーの例
- ポリシーに関するベストプラクティス
- CodeDeploy コンソールの使用
- 自分の権限の表示をユーザーに許可する

カスタマーマネージドポリシーの例

このセクションでは、さまざまな CodeDeploy アクションのアクセス許可を付与するポリシー例を示しています。これらのポリシーは、CodeDeploy API、 AWS SDK、または AWS CLIを使用しているときに機能します。コンソールで実行するアクションには、追加のアクセス許可を付与する必要があります。コンソールアクセス許可の付与の詳細については、「CodeDeploy コンソールの使用」を参照してください。

> **Note**
>
> 例はすべて、米国西部 (オレゴン) リージョン (us-west-2) を使用し、架空のアカウント ID を使用しています。

例

- 例 1: 単一の リージョンで CodeDeploy オペレーションを実行することを許可する
- 例 2: 単一のアプリケーションへのリビジョンの登録を許可する
- 例 3: 単一のデプロイグループへのデプロイの作成を許可する

例 1: 単一の リージョンで CodeDeploy オペレーションを実行することを許可する

次の例では、CodeDeploy オペレーションを **us-west-2** リージョンでのみ実行するアクセス許可を付与します。

JSON

```
{
  "Version":"2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "codedeploy:*"
      ],
      "Resource" : [
        "arn:aws:codedeploy:us-west-2:111122223333:*"
      ]
    }
  ]
}
```

例 2: 単一のアプリケーションへのリビジョンの登録を許可する

次の例では、**us-west-2** リージョンの **Test** で始まるすべてのアプリケーションのアプリケーションリビジョンを登録するアクセス許可を付与します。

JSON

```
{
  "Version":"2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "codedeploy:RegisterApplicationRevision"
      ],
      "Resource" : [
        "arn:aws:codedeploy:us-west-2:111122223333:application:Test*"
      ]
    }
  ]
}
```

例 3: 単一のデプロイグループへのデプロイの作成を許可する

次の例では、**WordPress_App** という名前のアプリケーションに関連付けられた **WordPress_DepGroup** という名前のデプロイグループ用、**ThreeQuartersHealthy** という名前のカスタムデプロイ構成用、および **WordPress_App** という名前のアプリケーションに関連付けられたあらゆるアプリケーションリビジョン用のデプロイを作成することを許可します。これらのリソースはすべて **us-west-2** リージョンにあります。

JSON

```
{
  "Version":"2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "codedeploy:CreateDeployment"
      ],
      "Resource" : [
        "arn:aws:codedeploy:us-west-2:111122223333:deploymentgroup:WordPress_App/
WordPress_DepGroup"
      ]
    },
    {
      "Effect" : "Allow",
      "Action" : [
        "codedeploy:GetDeploymentConfig"
      ],
      "Resource" : [
        "arn:aws:codedeploy:us-
west-2:111122223333:deploymentconfig:ThreeQuartersHealthy"
      ]
    },
    {
      "Effect" : "Allow",
      "Action" : [
        "codedeploy:GetApplicationRevision"
      ],
      "Resource" : [
        "arn:aws:codedeploy:us-west-2:111122223333:application:WordPress_App"
      ]
    }
```

```
  ]
}
```

ポリシーに関するベストプラクティス

ID ベースのポリシーは、ユーザーのアカウント内の CodeDeploy リソースを誰かが作成、アクセス、または削除できるどうかを決定します。これらのアクションでは、 AWS アカウントに費用が発生する場合があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS 管理ポリシーを開始し、最小特権のアクセス許可に移行する – ユーザーとワークロードにアクセス許可の付与を開始するには、多くの一般的なユースケースにアクセス許可を付与するAWS 管理ポリシーを使用します。これらは で使用できます AWS アカウント。ユースケースに固有の AWS カスタマー管理ポリシーを定義することで、アクセス許可をさらに減らすことをお勧めします。詳細については、IAM ユーザーガイド の AWS マネージドポリシー または ジョブ機能のAWS マネージドポリシー を参照してください。
- 最小特権を適用する – IAM ポリシーでアクセス許可を設定する場合は、タスクの実行に必要な許可のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権アクセス許可とも呼ばれています。IAM を使用して許可を適用する方法の詳細については、IAM ユーザーガイド の IAM でのポリシーとアクセス許可 を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する – ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。例えば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。条件を使用して、サービスアクションが などの特定の を通じて使用されている場合に AWS のサービス、サービスアクションへのアクセスを許可することもできます CloudFormation。詳細については、IAM ユーザーガイド の IAM JSON ポリシー要素:条件 を参照してください。
- IAM アクセスアナライザーを使用して IAM ポリシーを検証し、安全で機能的なアクセス許可を確保する – IAM アクセスアナライザー は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠していることを確認します。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、IAM ユーザーガイド の IAM Access Analyzer でポリシーを検証する を参照してください。
- 多要素認証 (MFA) を要求する – で IAM ユーザーまたはルートユーザーを必要とするシナリオがある場合は AWS アカウント、MFA をオンにしてセキュリティを強化します。API オペレーション

が呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、IAM ユーザーガイド の MFA を使用した安全な API アクセス を参照してください。

IAM でのベストプラクティスの詳細については、「IAM ユーザーガイド」の「IAM でのセキュリティのベストプラクティス」を参照してください。

CodeDeploy コンソールの使用

CodeDeploy コンソールを使用する場合は、 AWS アカウントの他の AWS リソースを記述できる最小限のアクセス許可のセットが必要です。CodeDeploy コンソール内で CodeDeploy を利用するには、次のサービスのアクセス許可が必要です。

- Amazon EC2 Auto Scaling
- AWS CodeDeploy
- Amazon Elastic Compute Cloud
- Elastic Load Balancing
- AWS Identity and Access Management
- Amazon Simple Storage Service
- Amazon Simple Notification Service
- Amazon CloudWatch

これらの最小限必要なアクセス許可よりも制限された IAM ポリシーを作成している場合、その IAM ポリシーを使用するユーザーに対してコンソールは意図したとおりには機能しません。このようなユーザーが CodeDeploy コンソールを使用できることを保証するには、ユーザーに割り当てられたロール対する `AWSCodeDeployReadOnlyAccess` マネージドポリシーのアタッチも行ってください。こちらは「AWS CodeDeploy の マネージド (事前定義) ポリシー」に記述があります。

AWS CLI または CodeDeploy API のみを呼び出すユーザーには、最小限のコンソールアクセス許可を付与する必要はありません。

自分の権限の表示をユーザーに許可する

この例では、ユーザーアイデンティティにアタッチされたインラインおよびマネージドポリシーの表示を IAM ユーザーに許可するポリシーの作成方法を示します。このポリシーには、コンソールで、または AWS CLI または AWS API を使用してプログラムでこのアクションを実行するアクセス許可が含まれています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS CodeDeploy ID とアクセスのトラブルシューティング

次の情報は、CodeDeploy と IAM の使用に伴って発生する可能性がある一般的な問題の診断や修復に役立ちます。

トピック

- iam:PassRole を実行する権限がありません
- 自分の AWS アカウント以外のユーザーに CodeDeploy リソースへのアクセスを許可したい

iam:PassRole を実行する権限がありません

`iam:PassRole` アクションを実行する権限がないというエラーが表示された場合は、ポリシーを更新して CodeDeploy にロールを渡せるようにする必要があります。

一部の AWS のサービス では、新しいサービスロールまたはサービスにリンクされたロールを作成する代わりに、そのサービスに既存のロールを渡すことができます。そのためには、サービスにロールを渡すアクセス許可が必要です。

以下のエラー例は、`marymajor` という IAM ユーザーがコンソールを使用して CodeDeploy でアクションを実行しようとした場合に発生したものです。ただし、このアクションをサービスが実行するには、サービスロールから付与されたアクセス許可が必要です。Mary には、ロールをサービスに渡すアクセス許可がありません。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

この場合、Mary のポリシーを更新して `iam:PassRole` アクションの実行を彼女に許可する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン資格情報を提供した担当者が管理者です。

自分の AWS アカウント以外のユーザーに CodeDeploy リソースへのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセスコントロールリスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください:

- CodeDeploy がこれらの機能をサポートするかどうかについては、「AWS CodeDeploy が IAM と連携する方法」を参照してください。
- 所有 AWS アカウント している のリソースへのアクセスを提供する方法については、IAM ユーザーガイドの「所有 AWS アカウント している別の の IAM ユーザーへのアクセスを提供する」を参照してください。

- リソースへのアクセスをサードパーティーに提供する方法については AWS アカウント、IAM ユーザーガイドの「サードパーティー AWS アカウント が所有する へのアクセスを提供する」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、IAM ユーザーガイド の 外部で認証されたユーザー (ID フェデレーション) へのアクセスの許可 を参照してください。
- クロスアカウントアクセスにおけるロールとリソースベースのポリシーの使用方法の違いについては、「IAM ユーザーガイド」の「IAM でのクロスアカウントのリソースへのアクセス」を参照してください。

CodeDeploy アクセス許可 リファレンス

アクセスをセットアップし、IAM アイデンティティ (アイデンティティベースのポリシー) にアタッチできるアクセス権限ポリシーを作成する際、以下の表をリファレンスとして使用できます。この表には、各 CodeDeploy API オペレーション、アクションを実行するためのアクセス許可を付与できるアクション、およびアクセス許可を付与するために使用するリソース ARN の形式が示されています。アクションは、ポリシーの `Action` フィールドで指定します。ポリシーの `Resource` フィールドでリソース値として、ワイルドカード文字 (*) を使用して、または使用せずに ARN を指定します。

CodeDeploy ポリシーで AWS全体の条件キーを使用して、条件を表現できます。 AWS全体のキーの完全なリストについては、IAM ユーザーガイドの「使用可能なキー」を参照してください。

アクションを指定するには、API オペレーション名 (例えば、`codedeploy:` や `codedeploy:GetApplication`) の前に `codedeploy:CreateApplication` プレフィックスを使用します。単一のステートメントに複数のアクションを指定するには、コンマで区切ります (例えば、`"Action": ["codedeploy:action1", "codedeploy:action2"]`)。

ワイルドカード文字の使用

ARN でワイルドカード文字 (*) を使用して、複数のアクションまたはリソースを指定できます。例えば、`codedeploy:*` は、すべての CodeDeploy アクションを指定し、`codedeploy:Get*` は、`Get` という単語で始まるすべての CodeDeploy アクションを指定します。次の例では、`West` で始まり、名前が `Test` で始まるアプリケーションに関連付けられている名前を持つすべてのデプロイグループにアクセス権限を付与します。

```
arn:aws:codedeploy:us-west-2:444455556666:deploymentgroup:Test*/West*
```

表に表示されている次のリソースでワイルドカードを使用できます。

- *application-name*
- *deployment-group-name*
- *deployment-configuration-name*
- *instance-ID*

ワイルドカードは *region* または *account-id* では使用できません。ワイルドカードの詳細については、IAM ユーザーガイドの IAM ID を参照してください。

> **Note**
>
> 各アクションの ARN ではリソースの後にコロン (:) が続きます。また、リソースの後にスラッシュ (/) を使用できます。詳細については、「CodeDeploy の ARN の例」を参照してください。

CodeDeploy API オペレーションおよびコミットされたコードのアクションで必要なアクセス権限

AddTagsToOnPremisesInstances

アクション:codedeploy:AddTagsToOnPremisesInstances

1 つ以上のオンプレミスインスタンスにタグを追加するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:instance/*instance-ID*

BatchGetApplicationRevisions

アクション:codedeploy:BatchGetApplicationRevisions

ユーザーに関連付けられた複数のアプリケーションリビジョンに関する情報を取得するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:application:*application-name*

BatchGetApplications

アクション:codedeploy:BatchGetApplications

ユーザーに関連付けられた複数のアプリケーションに関する情報を取得するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:application:*

BatchGetDeploymentGroups

アクション:codedeploy:BatchGetDeploymentGroups

ユーザーに関連付けられた複数のデプロイグループに関する情報を取得するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:deploymentgroup:*application-name*/*deployment-group-name*

BatchGetDeploymentInstances

アクション:codedeploy:BatchGetDeploymentInstances

デプロイグループの一部である 1 つ以上のインスタンスに関する情報を取得するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:deploymentgroup:*application-name*/*deployment-group-name*

BatchGetDeployments

アクション:codedeploy:BatchGetDeployments

ユーザーに関連付けられた複数のデプロイに関する情報を取得するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:deploymentgroup:*application-name*/*deployment-group-name*

BatchGetOnPremisesInstances

アクション:codedeploy:BatchGetOnPremisesInstances

1 つ以上のオンプレミスインスタンスに関する情報を取得するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:*

ContinueDeployment

アクション:codedeploy:ContinueDeployment

ブルー/グリーンデプロイ中、Elastic Load Balancing ロードバランサーを使用して、置き換え先環境でのインスタンス登録を開始するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:deploymentgroup:*application-name*/*deployment-group-name*

CreateApplication

アクション:codedeploy:CreateApplication

ユーザーに関連付けられたアプリケーションを作成するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:application:*application-name*

CreateDeployment [1]

アクション:codedeploy:CreateDeployment

ユーザーに関連付けられたアプリケーションのデプロイを作成するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:deploymentgroup:*application-name*/*deployment-group-name*

CreateDeploymentConfig

アクション:codedeploy:CreateDeploymentConfig

ユーザーに関連付けられたカスタムデプロイ設定を作成するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:deploymentconfig/*deployment-configuration-name*

CreateDeploymentGroup

アクション:codedeploy:CreateDeploymentGroup

ユーザーに関連付けられたアプリケーションのデプロイグループを作成するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:deploymentgroup:*application-name*/*deployment-group-name*

DeleteApplication

アクション:codedeploy:DeleteApplication

ユーザーに関連付けられたアプリケーションを削除するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:application:*application-name*

DeleteDeploymentConfig

アクション:codedeploy:DeleteDeploymentConfig

ユーザーに関連付けられたカスタムデプロイ設定を削除するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:deploymentconfig/*deployment-configuration-name*

DeleteDeploymentGroup

アクション:codedeploy:DeleteDeploymentGroup

ユーザーに関連付けられたアプリケーションのデプロイグループを削除するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:deploymentgroup:*application-name*/*deployment-group-name*

DeregisterOnPremisesInstance

アクション:codedeploy:DeregisterOnPremisesInstance

オンプレミスインスタンスを登録解除するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:instance/*instance-ID*

GetApplication

アクション:codedeploy:GetApplication

ユーザーに関連付けられた単一のアプリケーションに関する情報を取得するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:application:*application-name*

GetApplicationRevision

アクション:codedeploy:GetApplicationRevision

ユーザーに関連付けられたアプリケーションの単一のアプリケーションのリビジョンに関する情報を取得するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:application:*application-name*

GetDeployment

アクション:codedeploy:GetDeployment

ユーザーに関連付けられたアプリケーションのデプロイグループへの単一のデプロイに関する情報を取得するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:deploymentgroup:*application-name*/*deployment-group-name*

GetDeploymentConfig

アクション:codedeploy:GetDeploymentConfig

ユーザーに関連付けられた単一のデプロイ設定に関する情報を取得するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:deploymentconfig/*deployment-configuration-name*

GetDeploymentGroup

アクション:codedeploy:GetDeploymentGroup

ユーザーに関連付けられたアプリケーションの単一のデプロイグループに関する情報を取得するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:deploymentgroup:*application-name*/*deployment-group-name*

GetDeploymentInstance

アクション:codedeploy:GetDeploymentInstance

ユーザーに関連付けられたデプロイの単一のインスタンスに関する情報を取得するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:deploymentgroup:*application-name*/*deployment-group-name*

GetOnPremisesInstance

アクション:codedeploy:GetOnPremisesInstance

単一のオンプレミスインスタンスに関する情報を取得するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:instance/*instance-ID*

ListApplicationRevisions

アクション:codedeploy:ListApplicationRevisions

ユーザーに関連付けられたアプリケーションのすべてのアプリケーションリビジョンに関する情報を取得するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:application:*

ListApplications

アクション:codedeploy:ListApplications

ユーザーに関連付けられたすべてのアプリケーションに関する情報を取得するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:application:*

ListDeploymentConfigs

アクション:codedeploy:ListDeploymentConfigs

ユーザーに関連付けられたすべてのデプロイ設定に関する情報を取得するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:deploymentconfig/*

ListDeploymentGroups

アクション:codedeploy:ListDeploymentGroups

ユーザーに関連付けられたアプリケーションのすべてのデプロイグループに関する情報を取得するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:deploymentgroup:*application-name*/*

ListDeploymentInstances

アクション:codedeploy:ListDeploymentInstances

ユーザーに関連付けられたデプロイのすべてのインスタンスに関する情報を取得するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:deploymentgroup:*application-name*/*deployment-group-name*

ListDeployments

アクション:codedeploy:ListDeployments

ユーザーに関連付けられたデプロイグループへのすべてのデプロイに関する情報、またはユーザーに関連付けられたすべてのデプロイに関する情報を取得するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:deploymentgroup:*application-name*/*deployment-group-name*

ListGitHubAccountTokenNames

アクション:codedeploy:ListGitHubAccountTokenNames

GitHub アカウントへの保存された接続の名前を一覧表示するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:*

ListOnPremisesInstances

アクション:codedeploy:ListOnPremisesInstances

1 つ以上のオンプレミスインスタンス名のリストを取得するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:*

RegisterApplicationRevision

アクション:codedeploy:RegisterApplicationRevision

ユーザーに関連付けられたアプリケーションのアプリケーションリビジョンに関する情報を登録するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:application:*application-name*

RegisterOnPremisesInstance

アクション:codedeploy:RegisterOnPremisesInstance

オンプレミスインスタンスを CodeDeploy で登録するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:instance/*instance-ID*

RemoveTagsFromOnPremisesInstances

アクション:codedeploy:RemoveTagsFromOnPremisesInstances

1 つ以上のオンプレミスインスタンスからタグを削除するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:instance/*instance-ID*

SkipWaitTimeForInstanceTermination

アクション:codedeploy:SkipWaitTimeForInstanceTermination

指定された待機時間をオーバーライドし、ブルー/グリーンデプロイでトラフィックが正常にルーティングした直後に元の環境でインスタンスの削除を開始するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:instance/*instance-ID*

StopDeployment

アクション:codedeploy:StopDeployment

ユーザーに関連付けられたアプリケーションのデプロイグループへの進行中のデプロイを停止するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:deploymentgroup:*application-name*/*deployment-group-name*

UpdateApplication [3]

アクション:codedeploy:UpdateApplication

ユーザーに関連付けられたアプリケーションに関する情報を変更するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:application:*application-name*

UpdateDeploymentGroup [3]

アクション:codedeploy:UpdateDeploymentGroup

ユーザーに関連付けられたアプリケーションで単一のデプロイグループに関する情報を変更するために必要です。

リソース: arn:aws:codedeploy:*region*:*account-id*:deploymentgroup:*application-name*/*deployment-group-name*

CreateDeployment アクセス許可を指定する場合は、デプロイ設定および GetDeploymentConfig の GetApplicationRevision アクセス許可、またはアプリケーションリビジョンへの RegisterApplicationRevision アクセス許可も指定する必要があります。

ListDeployments 特定のデプロイグループを指定する場合には有効ですが、ユーザーに関連付けられたすべてのデプロイを一覧表示する場合には有効ではありません。

UpdateApplication については、古いアプリケーション名と新しいアプリケーション名の両方に対する UpdateApplication アクセス許可が必要です。デプロイグループの名前の変更を伴う

UpdateDeploymentGroup アクションの場合、古いデプロイグループ名と新しいデプロイグループ名の両方に対する UpdateDeploymentGroup アクセス許可が必要です。

サービス間での不分別な代理処理の防止

混乱した代理問題は、アクションを実行するためのアクセス許可を持たないエンティティが、より特権のあるエンティティにアクションの実行を強制できてしまう場合に生じる、セキュリティ上の問題です。では AWS、サービス間のなりすましにより、混乱した代理問題が発生する可能性があります。サービス間でのなりすましは、1 つのサービス (呼び出し元サービス) が、別のサービス (呼び出し対象サービス) を呼び出すときに発生する可能性があります。呼び出し元サービスは、本来ならアクセスすることが許可されるべきではない方法でその許可を使用して、別のお客様のリソースに対する処理を実行するように操作される場合があります。これを防ぐために、 は、アカウント内のリソースへのアクセス権が付与されたサービスプリンシパルを持つすべてのサービスのデータを保護するのに役立つツール AWS を提供します。

リソースポリシーで aws:SourceArn および aws:SourceAccount のグローバル条件コンテキストキーを使用して、CodeDeploy が別のサービスに付与する許可をそのリソースに制限することをお勧めします。同じポリシーステートメントでこれらのグローバル条件コンテキストキーの両方を使用し、アカウント ID にaws:SourceArn の値が含まれていない場合、aws:SourceAccount 値と aws:SourceArn 値の中のアカウントには、同じアカウント ID を使用する必要があります。クロスサービスのアクセスにリソースを 1 つだけ関連付けたい場合は、aws:SourceArn を使用します。クロスサービスが使用できるように、アカウント内の任意のリソースを関連付けたい場合は、aws:SourceAccount を使用します。

EC2/オンプレミス、 AWS Lambda、および通常の Amazon ECS デプロイの場合、 の値には CodeDeploy が IAM ロールを引き受けることができる CodeDeploy デプロイグループ ARN を含めるaws:SourceArn必要があります。

を使用して作成された Amazon ECS ブルー/グリーンデプロイ CloudFormationの場合、 の値には CodeDeploy が IAM ロールを引き受けることができる CloudFormation スタック ARN を含めるaws:SourceArn必要があります。

混乱した代理問題から保護するための最も効果的な方法は、リソースの完全な ARN を指定して aws:SourceArn キーを使用することです。リソースの完全な ARN が不明な場合や、複数のリソースを指定する場合には、未知部分を示すためにワイルドカード文字 (*) を使用します。

たとえば、EC2/オンプレミス、 AWS Lambda、または通常の Amazon ECS デプロイで次の信頼ポリシーを使用できます。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": "codedeploy.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:codedeploy:us-
east-1:111122223333:deploymentgroup:myApplication/*"
                }
            }
        }
    ]
}
```

を使用して作成された Amazon ECS ブルー/グリーンデプロイ CloudFormationでは、以下を使用できます。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": "codedeploy.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
```

```
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:cloudformation:us-
east-1:111122223333:stack/MyCloudFormationStackName/*"
                }
            }
        }
    ]
}
```

CodeDeploy でのロギングとモニタリング

このセクションでは、CodeDeploy でのモニタリング、ログ記録、およびインシデントへの対応の概要を説明します。

CodeDeploy とのすべてのインタラクションの監査

CodeDeploy は AWS CloudTrail、 AWS アカウントで CodeDeploy によって行われた、または CodeDeploy に代わって行われた API コールをキャプチャし、指定した S3 バケットにログファイルを配信するサービスである と統合されています。CloudTrail は、CodeDeploy コンソール、 AWS CLIを通した CodeDeploy コマンド、または CodeDeploy API からの API コールを直接キャプチャします。CloudTrail で収集された情報を使用して、CodeDeploy に対するリクエスト、リクエスト元の出典 IP アドレス、リクエスト送信者、リクエスト日時などの詳細を確認できます。CloudTrail のより詳しい情報については、AWS CloudTrail ユーザーガイド 内の 「CloudTrail ログファイルの操作」を参照してください。

CodeDeploy のデプロイで作成されたログデータを表示するには、Amazon CloudWatch エージェントを設定して CloudWatch コンソール内の集計データを表示するか、インスタンスにサインインしてログファイルを確認します。詳細については、「CodeDeploy エージェントログを CloudWatch に送信する」を参照してください。

アラートと障害管理

Amazon CloudWatch Events を使用して、CodeDeploy オペレーション内のインスタンスまたはデプロイ (「イベント」) のステートにおける変化を検出して対処できます。この場合、デプロイまたは

インスタンスがルール内で指定した状態になると、作成したルールに基づいて、CloudWatch Events が 1 つ以上のターゲットアクションを呼び出します。状態変更のタイプに応じて、通知を送信、状態情報を取得し、修正作業またはその他のアクションを取ることができます。CloudWatch Events を CodeDeploy オペレーションの一部としてを使用する場合、次のタイプのターゲットを選択できます。

- AWS Lambda 関数
- Kinesis Streams
- Amazon SQS SQS キュー
- 組み込みターゲット (CloudWatch アラームアクション)
- Amazon SNS トピック

次にユースケースをいくつか示します。

- Lambda 機能を使用して、デプロイが失敗するたびに Slack チャネルに通知を配信します。
- Kinesis ストリームにデプロイまたはインスタンスのデータをプッシュして、包括的でリアルタイムの状態モニタリングをサポートします。
- CloudWatch アラームアクションを使用することで、指定したデプロイやインスタンスイベントが発生したときに、EC2 インスタンスを自動的に停止、終了、再起動、もしくは復旧することができます。

詳細については、「Amazon CloudWatch ユーザーガイド」の「Amazon CloudWatch Events とは」を参照してください。

AWS CodeDeploy のコンプライアンス検証

新しい AWS サインアップエクスペリエンスは、規制されたワークロード向けには設計されていません。新しい AWS サインアップエクスペリエンスを使用しているが、規制対象のワークロード AWS に を使用する場合は、 AWS (アドバンスド) にサインアップするか、 AWS 環境の高度な機能をアクティブ化できます。

AWS のサービス が特定のコンプライアンスプログラムの範囲内にあるかどうかを確認するには、AWS のサービス「コンプライアンスプログラムによるスコープ」の「コンプライアンス」を参照して、関心のあるコンプライアンスプログラムを選択します。一般的な情報については、AWS「コンプライアンスプログラム」を参照してください。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「Downloading Reports in AWS Artifact」を参照してください。

を使用する際のお客様のコンプライアンス責任 AWS のサービス は、お客様のデータの機密性、貴社のコンプライアンス目的、適用される法律および規制によって決まります。を使用する際のコンプライアンス責任の詳細については AWS のサービス、AWS 「 セキュリティドキュメント」を参照してください。

AWS CodeDeploy の耐障害性

AWS グローバルインフラストラクチャは、 AWS リージョンとアベイラビリティーゾーンを中心に構築されています。 AWS リージョンは、低レイテンシー、高スループット、および高度に冗長なネットワークで接続された、物理的に分離および分離された複数のアベイラビリティーゾーンを提供します。アベイラビリティーゾーンでは、ゾーン間で中断することなく自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性、フォールトトレランス、および拡張性が優れています。

AWS リージョンとアベイラビリティーゾーンの詳細については、AWS 「 グローバルインフラストラクチャ」を参照してください。

インフラストラクチャセキュリティ in AWS CodeDeploy

マネージドサービスである AWS CodeDeploy は、ホワイトペーパー「Amazon Web Services: セキュリティプロセスの概要」に記載されている AWS グローバルネットワークセキュリティ手順で保護されています。

AWS 公開された API コールを使用して、ネットワーク経由で CodeDeploy にアクセスします。クライアントは、Transport Layer Security (TLS) 1.2 以降をサポートする必要があります。TLS 1.3 以降が推奨されます。また、Ephemeral Diffie-Hellman (DHE) や Elliptic Curve Ephemeral Diffie-Hellman (ECDHE) などの Perfect Forward Secrecy (PFS) を使用した暗号スイートもクライアントでサポートされている必要があります。これらのモードは Java 7 以降など、ほとんどの最新システムでサポートされています。

リクエストは、アクセスキー ID と、IAM プリンシパルに関連付けられているシークレットのアクセスキーを使用して署名する必要があります。または、AWS Security Token Service (AWS STS) を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

参照資料

リファレンス

トピック

- CodeDeploy AppSpec ファイルのリファレンス
- CodeDeploy エージェント設定リファレンス
- CloudFormation CodeDeploy リファレンスの テンプレート
- Amazon Virtual Private Cloud で CodeDeploy を使用
- CodeDeploy リソースキットリファレンス
- CodeDeploy のクォータ

CodeDeploy AppSpec ファイルのリファレンス

このセクションは参照のみを目的としています。AppSpec ファイルの概念の概要については、Application Specification Files を参照してください。

アプリケーション仕様ファイル (AppSpec ファイル) は、CodeDeploy がデプロイを管理するために使用する YAML 形式または JSON 形式のファイルです。

> **Note**
>
> EC2/オンプレミスデプロイの AppSpec ファイルは、ローカルデプロイを実行しない限り、`appspec.yml` という名前にする必要があります。詳細については、「ローカルのデプロイを作成する。」を参照してください。

トピック

- Amazon ECS コンピューティングプラットフォームの AppSpec ファイル
- AWS Lambda コンピューティングプラットフォーム上の AppSpec ファイル
- EC2/オンプレミスコンピューティングプラットフォームの AppSpec ファイル
- AppSpec ファイル構造
- AppSpec ファイルの例
- AppSpec ファイルの間隔

- AppSpec ファイルの検証とファイルの場所

Amazon ECS コンピューティングプラットフォームの AppSpec ファイル

Amazon ECS コンピューティングプラットフォームアプリケーションでは、AppSpec ファイルは CodeDeploy が判断するために使用されます。

- Amazon ECS タスク定義ファイル これは、AppSpec ファイルの `TaskDefinition` 命令の ARN で指定されます。
- 置き換えタスクのコンテナとポートは、Application Load Balancer または Network Load Balancer がデプロイ中にトラフィックを再ルーティングする場所を設定します。これは、AppSpec ファイルの `LoadBalancerInfo` 命令で指定されます。
- サービスが実行されるプラットフォームのバージョン、そのサブネット、およびそのセキュリティグループなどの、Amazon ECS サービスに関するオプション情報。
- Amazon ECS のデプロイ中、ライフサイクルイベントに応じたフック中に実行される、オプションの Lambda 関数。詳細については、「Amazon ECS のデプロイ向けの AppSpec の「hooks」セクション」を参照してください。

AWS Lambda コンピューティングプラットフォーム上の AppSpec ファイル

AWS Lambda コンピューティングプラットフォームアプリケーションの場合、AppSpec ファイルは CodeDeploy が以下を判断するために使用されます。

- デプロイする Lambda 関数のバージョン。
- 検証テストとして使用する Lambda 関数。

AppSpec ファイルは、YAML 形式または JSON 形式とすることができます。デプロイの作成時に、AppSpec ファイルの内容を直接 CodeDeploy コンソールに入力することもできます。

EC2/オンプレミスコンピューティングプラットフォームの AppSpec ファイル

アプリケーションで EC2/オンプレミスコンピューティングプラットフォームを使用する場合、AppSpec ファイルは、`appspec.yml` という名前の YAML 形式のファイルで、アプリケーショ

ンのソースコードのディレクトリ構造のルートに配置する必要があります。それ以外の場合、デプロイは失敗します。CodeDeploy を決定するために使用されます。

- Amazon S3 または GitHub のアプリケーションリビジョンからインスタンスにインストールする必要があるもの。
- デプロイライフサイクルイベントに応じて実行するライフサイクルイベントフック。

AppSpec ファイルを完了したら、デプロイするコンテンツとともに、アーカイブファイル (zip、tar、または圧縮 tar) にバンドルします。詳細については、「CodeDeploy のアプリケーションリビジョンの操作」を参照してください。

> **Note**
>
> tar および圧縮 tar アーカイブファイル形式 (.tar および .tar.gz) は、Windows Server インスタンスではサポートされていません。

アーカイブファイル (CodeDeploy では リビジョン と呼ばれます) をバンドルしたら、Amazon S3 バケットまたは Git レポジトリにアップロードします。次に、CodeDeploy を使用してリビジョンをデプロイします。手順については、「CodeDeploy でデプロイを作成する」を参照してください。

EC2/オンプレミスコンピューティングプラットフォームのデプロイの appspec.yml は、リビジョンのルートディレクトリに保存されます。詳細については、「EC2/オンプレミスデプロイに AppSpec ファイルを追加する」および「CodeDeploy のリビジョンを計画する」を参照してください。

AppSpec ファイル構造

次に示すのは、 AWS Lambda および EC2/オンプレミスコンピューティングプラットフォームへのデプロイに使用される AppSpec ファイルのハイレベルな構造です。

文字列である YAML 形式の AppSpec ファイルの値は、別途指定されている場合を除き、引用符 ("") で囲んではいけません。

Amazon ECS デプロイ向けの AppSpec ファイル構造

> **Note**
>
> この AppSpec ファイルは YAML で記述されますが、同じ構造を使用して JSON で記述することもできます。JSON 形式の AppSpec ファイルの文字列は、常に引用符 ("") で囲みます。

```
version: 0.0
resources:
  ecs-service-specifications
hooks:
  deployment-lifecycle-event-mappings
```

この構造の説明

バージョン

このセクションでは、AppSpec ファイルのバージョンを指定します。この値を変更しないでください。これは必須です。現在許容されている値は、**0.0** のみです。将来の利用のために CodeDeploy で予約されます。

文字列で version を指定します。

リソース

このセクションでは、デプロイする Amazon ECS アプリケーションに関する情報を指定します。

詳細については、「Amazon ECS デプロイ用の AppSpec の「resources」セクション」を参照してください。

hooks

このセクションでは、デプロイ検証のために特定のデプロイライフサイクルイベントフックで実行する、Lambda 関数を指定します。

詳細については、「Amazon ECS のデプロイ向けのライフサイクルイベントフックのリスト」を参照してください。

Lambda AWS デプロイの AppSpec ファイル構造

> **Note**
>
> この AppSpec ファイルは、YAML で記述されますが、同じ構造を使用して JSON で Lambda デプロイ用の AppSpec ファイルを記述することもできます。JSON 形式の AppSpec ファイルの文字列は、常に引用符 ("") で囲みます。

```
version: 0.0
resources:
  lambda-function-specifications
hooks:
  deployment-lifecycle-event-mappings
```

この構造の説明

バージョン

このセクションでは、AppSpec ファイルのバージョンを指定します。この値を変更しないでください。これは必須です。現在許容されている値は、**0.0** のみです。将来の利用のために CodeDeploy で予約されます。

文字列で version を指定します。

リソース

このセクションでは、デプロイする Lambda 関数に関する情報を指定します。

詳細については、「AppSpec の「リソース」セクション (Amazon ECS と AWS Lambda デプロイのみ)」を参照してください。

hooks

このセクションでは、デプロイを検証するために特定のデプロイライフサイクルイベントで実行する Lambda 関数を指定します。

詳細については、「AppSpec の「hooks」セクション」を参照してください。

EC2/オンプレミスデプロイの AppSpec ファイル構造

```
version: 0.0
os: operating-system-name
files:
  source-destination-files-mappings
permissions:
  permissions-specifications
hooks:
  deployment-lifecycle-event-mappings
```

この構造の説明

バージョン

このセクションでは、AppSpec ファイルのバージョンを指定します。この値を変更しないでください。これは必須です。現在許容されている値は、**0.0** のみです。将来の利用のために CodeDeploy で予約されます。

文字列で version を指定します。

os

このセクションでは、デプロイ先であるインスタンスのオペレーティングシステムの値を指定します。これは必須です。次の値を指定できます。

- linux – インスタンスは Amazon Linux、Ubuntu Server、RHEL、または macOS インスタンスです。
- windows— インスタンスは Windows Server インスタンスです。

文字列で os を指定します。

ファイル

このセクションでは、デプロイの Install イベント中に、インスタンスにコピーするファイル名を指定します。

詳細については、「AppSpec の「ファイル」セクション (EC2/オンプレミスデプロイのみ)」を参照してください。

アクセス権限

このセクションでは、インスタンスへのコピー中に特別なアクセス権限をファイルの `files` セクションに適用する方法を指定します。このセクションは、Amazon Linux、Ubuntu Server、Red Hat Enterprise Linux (RHEL)、macOS インスタンスにのみ適用されます。

詳細については、AppSpec の「許可」セクション (EC2/オンプレミスデプロイのみ) を参照してください。

hooks

このセクションでは、デプロイ中に特定のデプロイライフサイクルイベントで実行するスクリプトを指定します。

詳細については、「AppSpec の「hooks」セクション」を参照してください。

トピック

- AppSpec の「ファイル」セクション (EC2/オンプレミスデプロイのみ)
- AppSpec の「リソース」セクション (Amazon ECS と AWS Lambda デプロイのみ)
- AppSpec の「許可」セクション (EC2/オンプレミスデプロイのみ)
- AppSpec の「hooks」セクション

AppSpec の「ファイル」セクション (EC2/オンプレミスデプロイのみ)

デプロイの Install イベント中にインスタンスにインストールする、アプリケーションリビジョンのファイルに関する情報を CodeDeploy に提供します。このセクションは、デプロイ中にリビジョンからインスタンス上の場所にファイルをコピーする場合のみ必要です。

このセクションの構造は次のとおりです。

```
files:
  - source: source-file-location-1
    destination: destination-file-location-1
file_exists_behavior: DISALLOW|OVERWRITE|RETAIN
```

複数の `source` と `destination` ペアを設定できます。

`source` は、リビジョンからインスタンスにコピーするファイルまたはディレクトリを識別します。

- `source` はファイルを参照し、指定されたファイルのみがインスタンスにコピーされます。
- `source` はディレクトリを参照し、そのディレクトリのすべてのファイルがインスタンスにコピーされます。
- `source` がシングルスラッシュ (Amazon Linux、RHEL、および Ubuntu Server のインスタンスでは "/"、Windows Server のインスタンスでは "\") である場合、リビジョンのすべてのファイルがインスタンスにコピーされます。

`source` で使用されているパスは `appspec.yml` ファイルに対する相対パスで、 ファイルはリビジョンのルートに存在する必要があります。リビジョンのファイル構造の詳細については、「CodeDeploy のリビジョンを計画する」を参照してください。

`destination` は、ファイルをコピーするインスタンス上の場所を識別します。これは、`/root/destination/directory` (Linux、RHEL、Ubuntuの場合) または `c:\destination\folder` (Windows の場合) のような完全修飾パスである必要があります。

source と destination は、それぞれ文字列で指定されます。

file_exists_behavior 命令はオプションで、CodeDeploy がデプロイターゲットの場所にすでに存在し、前回成功したデプロイの一部ではなかったファイルを処理する方法を指定します。この設定は、以下のいずれかの値を取ることができます。

- DISALLOW: デプロイは失敗です。オプションが何も指定されていないときは、これもデフォルトの動作となります。
- OVERWRITE: 現在デプロイされているアプリケーションリビジョンのファイルのバージョンにより、インスタンスの既存のファイルのバージョンが置き換えられます。
- RETAIN: インスタンスにすでに存在するファイルのバージョンは保持され、新しいデプロイの一部として使用されます。

file_exists_behavior 設定を使用する場合、この設定を理解してください。

- は 1 度だけ指定でき、files: にリストされたすべてのファイルとディレクトリに適用されます。
- は、 --file-exists-behavior AWS CLI オプションと fileExistsBehavior API オプション (どちらもオプション) よりも優先されます。

以下は、Amazon Linux、Ubuntu Server、または RHEL インスタンスの files セクションの例です。

```
files:
  - source: Config/config.txt
    destination: /webapps/Config
  - source: source
    destination: /webapps/myApp
```

この例では、次の 2 つのオペレーションが、Install イベント中に実行されます。

1. 使用するリビジョンの Config/config.txt ファイルをインスタンスの /webapps/Config/config.txt パスにコピーします。
2. リビジョンの source ディレクトリのすべてのファイルを、インスタンスの /webapps/myApp ディレクトリに再帰的にコピーします。

「files」セクションの例

次の例は、files セクションを指定する方法を示しています。これらの例は、Windows Server ファイルとディレクトリ (フォルダ) 構造を示していますが、Amazon Linux、Ubuntu Server、および RHEL インスタンスに簡単に適用することができます。

> **Note**
>
> EC2/オンプレミスデプロイのみ、files セクションを使用します。 AWS Lambda デプロイには適用されません。

次の例では、以下のファイルが source のルートのバンドルに表示されることを前提としています。

- appspec.yml
- my-file.txt
- my-file-2.txt
- my-file-3.txt

```
# 1) Copy only my-file.txt to the destination folder c:\temp.
#
files:
  - source: .\my-file.txt
    destination: c:\temp
#
# Result:
#   c:\temp\my-file.txt
#
# ---------------------
#
# 2) Copy only my-file-2.txt and my-file-3.txt to the destination folder c:\temp.
#
files:
  - source: my-file-2.txt
    destination: c:\temp
  - source: my-file-3.txt
    destination: c:\temp
#
# Result:
```

```
#   c:\temp\my-file-2.txt
#   c:\temp\my-file-3.txt
#
# ---------------------
#
# 3) Copy my-file.txt, my-file-2.txt, and my-file-3.txt (along with the appspec.yml
 file) to the destination folder c:\temp.
#
files:
  - source: \
    destination: c:\temp
#
# Result:
#   c:\temp\appspec.yml
#   c:\temp\my-file.txt
#   c:\temp\my-file-2.txt
#   c:\temp\my-file-3.txt
```

次の例では、appspec.yml が source のルートのバンドルに、3 つのファイルを含む my-folder という名前のフォルダとともに表示されることを前提としています。

- appspec.yml
- my-folder\my-file.txt
- my-folder\my-file-2.txt
- my-folder\my-file-3.txt

```
# 4) Copy the 3 files in my-folder (but do not copy my-folder itself) to the
 destination folder c:\temp.
#
files:
  - source: .\my-folder
    destination: c:\temp
#
# Result:
#   c:\temp\my-file.txt
#   c:\temp\my-file-2.txt
#   c:\temp\my-file-3.txt
#
# ---------------------
#
# 5) Copy my-folder and its 3 files to my-folder within the destination folder c:\temp.
```

```
#
files:
  - source: .\my-folder
    destination: c:\temp\my-folder
#
# Result:
#   c:\temp\my-folder\my-file.txt
#   c:\temp\my-folder\my-file-2.txt
#   c:\temp\my-folder\my-file-3.txt
#
# ---------------------
#
# 6) Copy the 3 files in my-folder to other-folder within the destination folder c:
\temp.
#
files:
  - source: .\my-folder
    destination: c:\temp\other-folder
#
# Result:
#   c:\temp\other-folder\my-file.txt
#   c:\temp\other-folder\my-file-2.txt
#   c:\temp\other-folder\my-file-3.txt
#
# ---------------------
#
# 7) Copy only my-file-2.txt and my-file-3.txt to my-folder within the destination
 folder c:\temp.
#
files:
  - source: .\my-folder\my-file-2.txt
    destination: c:\temp\my-folder
  - source: .\my-folder\my-file-3.txt
    destination: c:\temp\my-folder
#
# Result:
#   c:\temp\my-folder\my-file-2.txt
#   c:\temp\my-folder\my-file-3.txt
#
# ---------------------
#
# 8) Copy only my-file-2.txt and my-file-3.txt to other-folder within the destination
 folder c:\temp.
#
```

```
files:
  - source: .\my-folder\my-file-2.txt
    destination: c:\temp\other-folder
  - source: .\my-folder\my-file-3.txt
    destination: c:\temp\other-folder
#
# Result:
#   c:\temp\other-folder\my-file-2.txt
#   c:\temp\other-folder\my-file-3.txt
#
# ---------------------
#
# 9) Copy my-folder and its 3 files (along with the appspec.yml file) to the
 destination folder c:\temp. If any of the files already exist on the instance,
 overwrite them.
#
files:
  - source: \
    destination: c:\temp
file_exists_behavior: OVERWRITE
#
# Result:
#   c:\temp\appspec.yml
#   c:\temp\my-folder\my-file.txt
#   c:\temp\my-folder\my-file-2.txt
#   c:\temp\my-folder\my-file-3.txt
```

AppSpec の「リソース」セクション (Amazon ECS と AWS Lambda デプロイのみ)

AppSpec ファイルの 'resources' セクションの内容は、デプロイのコンピューティングプラットフォームによって異なります。Amazon ECS デプロイの 'resources' セクションには、Amazon ECS タスク定義、最新の Amazon ECS タスクセットにトラフィックをルーティングするためのコンテナとポート、およびその他のオプション情報が含まれています。 AWS Lambda デプロイ'resources'のセクションには、Lambda 関数の名前、エイリアス、現在のバージョン、およびターゲットバージョンが含まれています。

トピック

- AWS Lambda デプロイの AppSpec 'resources' セクション
- Amazon ECS デプロイ用の AppSpec の「resources」セクション

AWS Lambda デプロイの AppSpec 'resources' セクション

'resources' セクションでは、デプロイする Lambda 関数を指定します。その構造は次のとおりです。

YAML:

```
resources:
  - name-of-function-to-deploy:
      type: "AWS::Lambda::Function"
      properties:
        name: name-of-lambda-function-to-deploy
        alias: alias-of-lambda-function-to-deploy
        currentversion: version-of-the-lambda-function-traffic-currently-points-to
        targetversion: version-of-the-lambda-function-to-shift-traffic-to
```

JSON:

```
"resources": [
    {
        "name-of-function-to-deploy" {
            "type": "AWS::Lambda::Function",
            "properties": {
                "name": "name-of-lambda-function-to-deploy",
                "alias": "alias-of-lambda-function-to-deploy",
                "currentversion": "version-of-the-lambda-function-traffic-currently-
points-to",
                "targetversion": "version-of-the-lambda-function-to-shift-traffic-to"
            }
        }
    }
]
```

各プロパティは文字列で指定します。

- name – 必須。これはデプロイする Lambda 関数の名前です。
- alias – 必須。これは Lambda 関数のエイリアスの名前です。
- currentversion – 必須。これは、トラフィックが現在指している Lambda 関数のバージョンです。値は有効な正の整数である必要があります。
- targetversion – 必須。これは、トラフィックの移行先の Lambda 関数のバージョンです。値は有効な正の整数である必要があります。

Amazon ECS デプロイ用の AppSpec の「resources」セクション

'resources' セクションでは、デプロイする Amazon ECS サービスを指定します。その構造は次のとおりです。

YAML:

```
Resources:
  - TargetService:
      Type: AWS::ECS::Service
      Properties:
        TaskDefinition: "task-definition-arn"
        LoadBalancerInfo:
          ContainerName: "ecs-container-name"
          ContainerPort: "ecs-application-port"
# Optional properties
        PlatformVersion: "ecs-service-platform-version"
        NetworkConfiguration:
          AwsvpcConfiguration:
            Subnets: ["ecs-subnet-1","ecs-subnet-n"]
            SecurityGroups: ["ecs-security-group-1","ecs-security-group-n"]
            AssignPublicIp: "ENABLED | DISABLED"
        CapacityProviderStrategy:
          - Base: integer
            CapacityProvider: "capacityProviderA"
            Weight: integer
          - Base: integer
            CapacityProvider: "capacityProviderB"
            Weight: integer
```

JSON:

```
"Resources": [
    {
        "TargetService": {
            "Type": "AWS::ECS::Service",
            "Properties": {
                "TaskDefinition": "task-definition-arn",
                "LoadBalancerInfo": {
                    "ContainerName": "ecs-container-name",
                    "ContainerPort": "ecs-application-port"
                },
                "PlatformVersion": "ecs-service-platform-version",
```

```
                "NetworkConfiguration": {
                    "AwsvpcConfiguration": {
                        "Subnets": [
                            "ecs-subnet-1",
                            "ecs-subnet-n"
                        ],
                        "SecurityGroups": [
                            "ecs-security-group-1",
                            "ecs-security-group-n"
                        ],
                        "AssignPublicIp": "ENABLED | DISABLED"
                    }
                },
                "CapacityProviderStrategy": [
                    {
                        "Base": integer,
                        "CapacityProvider": "capacityProviderA",
                        "Weight": integer
                    },
                    {
                        "Base": integer,
                        "CapacityProvider": "capacityProviderB",
                        "Weight": integer
                    }
                ]
            }
        }
    }
]
```

各プロパティは、ContainerPort 以外の文字列で指定され、数字である。

- TaskDefinition – 必須。これはデプロイする Amazon ECS サービスのタスク定義です。タスク定義の ARN で指定します。ARN 形式は arn:aws:ecs:*aws-region*:*account-id*:task-definition/*task-definition-family*:*task-definition-revision* です。詳細については、「Amazon リソースネーム (ARNsと AWS サービス名前空間」を参照してください。

> **Note**
>
> ARN の :*task-definition-revision* の割り当ては任意です。省略した場合、Amazon ECS はタスク定義の最新の ACTIVE リビジョンを使用します。

- ContainerName – 必須。これは、Amazon ECS アプリケーションを含む Amazon ECS コンテナの名前です。Amazon ECS タスク定義で指定されたコンテナにする必要があります。
- ContainerPort – 必須。これは、トラフィックのルーティング先となるコンテナ上のポートです。
- PlatformVersion: オプション。デプロイされた Amazon ECS サービス内の、Fargate タスクのプラットフォームのバージョン。詳細については、の「AWS Fargate プラットフォームバージョン」を参照してください。指定されない場合、デフォルトで LATEST が使用されます。
- NetworkConfiguration: オプション。AwsvpcConfiguration で、以下を指定することができます。詳細については、Amazon ECS Container Service API リファレンス の「AwsVpcConfiguration」を参照してください。
  - Subnets: オプション。Amazon ECS サービス内の、1 つ以上のサブネットのカンマ区切りリスト。
  - SecurityGroups: オプション。Amazon Elastic Container Service にある、1 つ以上のセキュリティグループのカンマ区切りリスト。
  - AssignPublicIp: オプション。Amazon ECS サービスの Elastic Network Interface がパブリック IP アドレスを受け取るかどうかを指定する文字列。有効な値は ENABLED および DISABLED です。

  > **Note**
  >
  > NetworkConfiguration の下にあるすべての設定を指定するか、何も指定しない必要があります。たとえば、Subnets を指定する場合は、 SecurityGroups と AssignPublicIp も指定する必要があります。何も指定しない場合、CodeDeploy は現在のネットワーク Amazon ECS の設定を使用します。

- CapacityProviderStrategy: オプション。デプロイに使用したい Amazon ECS キャパシティプロバイダーのリスト。詳細については、Amazon Elastic Container Service デベロッパーガイド の「Amazon ECS キャパシティープロバイダー」を参照してください。キャパシティプロバイダーごとに、次の設定を指定できます。これらの設定の詳細については、AWS CloudFormation ユーザーガイド の「AWS::ECS::ServiceCapacityProviderStrategyItem」を参照してください。
  - Base: オプション。ベース値は、指定されたキャパシティープロバイダーで実行するタスクの最小限の数を指定します。キャパシティープロバイダー戦略では、ベースを定義できるキャパシティープロバイダーは 1 つだけです。値が指定されていない場合は、デフォルト値の 0 が使用されます。
  - CapacityProvider: オプション。容量プロバイダーの短い名前。例: capacityProviderA

- `Weight`: オプション。

  ウエイト値は、指定したキャパシティープロバイダーを使用する起動済みタスクの総数に対する相対的な割合を示します。`weight` 値は、`base` 値が、もし定義されている場合、満たされた後に、考慮されます。

  `weight` 値が指定されていない場合は、デフォルト値の `0` が使用されます。キャパシティープロバイダー戦略内で複数のキャパシティープロバイダーを指定する場合、少なくとも 1 つのキャパシティープロバイダーのウエイト値が 0 より大きい必要があり、ウエイトが `0` のキャパシティープロバイダーはタスクを配置するのに使用しません。すべてのキャパシティープロバイダーが `0` のウエイトを持つ戦略で複数のキャパシティープロバイダーを指定すると、キャパシティープロバイダー戦略を使用する `RunTask` または `CreateService` アクションは失敗します。

  例えば、加重を使用するシナリオが 2 つのキャパシティープロバイダーを含む戦略を定義し、両方の加重が 1 である場合、`base` が満たされたとき、タスクは 2 つのキャパシティープロバイダー間で均等に分割されます。同じロジックを使用して、キャパシティープロバイダー capacityProviderA に 1 のウエイトを指定し、capacityProviderB に 4 のウエイトを指定すると、capacityProviderA を使用して実行されるタスクごとに、4 つのタスクが capacityProviderB を使用します。

AppSpec の「許可」セクション (EC2/オンプレミスデプロイのみ)

`'permissions'` セクションでは、インスタンスへのコピー後に、特別なアクセス許可を `'files'` セクションのファイルおよびディレクトリ/フォルダに適用する方法を指定します。複数の `object` 指示を指定できます。このセクションはオプションです。Amazon Linux、Ubuntu Server、RHEL、macOS インスタンスにのみ適用されます。

> **Note**
>
> macOS インスタンスでは、 `acls`および `context`の手順はサポートされていません。AppSpec ファイルが macOS のいずれかを使用するデプロイは、ファイルをインストールする前に失敗します。他の手順は、このセクションで説明されているように機能します。

> **Note**
>
> EC2/オンプレミスデプロイにのみ、'permissions' セクションを使用します。 AWS Lambda または Amazon ECS のデプロイには使用されません。

このセクションの構造は次のとおりです。

```
permissions:
  - object: object-specification
    pattern: pattern-specification
    except: exception-specification
    owner: owner-account-name
    group: group-name
    mode: mode-specification
    acls:
      - acls-specification
    context:
      user: user-specification
      type: type-specification
      range: range-specification
    type:
      - object-type
```

手順は次のとおりです。

- object – 必須。これは、インスタンスへのファイルシステムオブジェクトのコピー後に、指定されたアクセス権限を適用する一連のファイルシステムオブジェクト (ファイルまたはディレクトリ/フォルダ) です。

  文字列で object を指定します。

- pattern - オプション。アクセス権限を適用するパターンを指定します。指定しない場合、または特殊文字 "**" で指定する場合、type に応じて一致するすべてのファイルまたはディレクトリに、指定されたアクセス権限が適用されます。

  引用符 ("") 付きの文字列で pattern を指定します。

- except - オプション。pattern の例外とするファイルまたはディレクトリを指定します。

  角括弧で囲った文字列のカンマ区切りリストで except を指定します。

- owner - オプション。object の所有者の名前。指定しない場合、既存のすべての所有者が元のファイルに適用されます。それ以外の場合、ディレクトリ/フォルダ構造は、コピーオペレーションによって変更されません。

  文字列で owner を指定します。

- group - オプション。object のグループの名前。指定しない場合、既存のすべてのグループが元のファイルに適用されます。それ以外の場合、ディレクトリ/フォルダ構造は、コピーオペレーションによって変更されません。

  文字列で group を指定します。

- mode - オプション。object に適用されるアクセス権限を指定する数値。モード設定は、Linux の chmod コマンド の構文に従います。

  > **Important**
  >
  > 値の先頭にゼロが含まれる場合は、ダブルクォートで囲むか、先頭のゼロを削除して 3 桁だけにする必要があります。

  > **Note**
  >
  > **u+x** の設定では、mode のような記号表記はサポートされていません。

  例:

  - mode: "0644" は、オブジェクトの所有者に読み書きのアクセス権限 (6)、グループに対する読み取り専用アクセス許可 (4)、およびその他すべてのユーザーに読み取り専用アクセス権限 (4) を与えます。
  - mode: 644 と同じ権限を付与する mode: "0644"。
  - mode: 4755 は setuid 属性を設定し (4)、所有者にフルコントロール権限を与え (7)、グループに対する読み取りと実行の権限を付与し (5)、他のすべてのユーザーに読み取りと実行の権限を付与します (5)。

    その他の例については、Linux の chmod コマンドのドキュメントを参照してください。

    モードを指定しない場合、既存のすべてのモードが元のファイルに適用されます。それ以外の場合、フォルダ構造は、コピーオペレーションによって変更されません。

- `acls` - オプション。1 つ以上のアクセスコントロールリスト (ACL) エントリを表す文字列のリストが、`object` に適用されます。たとえば、**`u:bob:rw`** は、ユーザー **bob** の読み取りおよび書き込みアクセス権限を表します (その他の例については、Linux の `setfacl` コマンドドキュメントの ACL 入力形式の例を参照してください)。複数の ACL エントリを指定できます。`acls` を指定しない場合、既存のすべての ACL が元のファイルに適用されます。それ以外の場合、ディレクトリ/フォルダ構造は、コピーオペレーションによって変更されません。既存の ACL は置き換えられます。

  `acls` を指定します。ダッシュ (-) の後にスペースを続け、その後に文字列を続けます (例: `- u:jane:rw`)。ACL が複数ある場合は、それぞれ個別の行で指定します。

  > **Note**
  >
  > 名前のないユーザー、名前のないグループ、またはその他の同様の ACL エントリを設定すると、AppSpec ファイルは失敗します。これらのタイプのアクセス許可を指定するには、代わりに `mode` を使用します。

- `context` - オプション。Security-Enhanced Linux (SELinux) 対応インスタンスの場合、コピーしたオブジェクトに適用されるセキュリティ関連コンテキストラベルのリスト。ラベルは、`user`, `type`、および `range` を含むキーとして指定されます。(詳細については、SELinux のドキュメントを参照してください)。各キーは文字列で入力します。指定しない場合、既存のすべてのラベルが元のファイルに適用されます。それ以外の場合、ディレクトリ/フォルダ構造は、コピーオペレーションによって変更されません。

  - `user` - オプション。SELinux ユーザー。
  - `type` - オプション。SELinux の型名。
  - `range` - オプション。SELinux の範囲指定子。マルチレベルセキュリティ (MLS) およびマルチカテゴリセキュリティ (MCS) がマシンで有効になっていない限り、この効果はありません。有効になっていない場合は、`range` はデフォルトで **`s0`** になります。

  文字列 で `context` を指定します (例: `user: unconfined_u`)。それぞれの `context` は個別の行で指定されます。

- `type` - オプション。指定された権限を適用するオブジェクトのタイプ。`type` は、**`file`** あるいは **`directory`** に設定できる文字列です。**`file`** を指定した場合、アクセス許可は、コピーオペレーションの後に (`object` 自体ではなく) `object` 内に直接含まれるファイルのみに適用されます。**`directory`** を指定した場合、アクセス権限は、コピーオペレーションの後に (`object` 自体ではなく)、`object` 内のいずれかの場所にあるすべてのディレクトリ/フォルダに再帰的に適用されます。

type を指定します。ダッシュ (-) の後にスペースを続け、その後に文字列を続けます (例: - file) 。

「permissions」セクションの例

次の例は、object、pattern、except、owner、mode、および type の手順を使用して 'permissions' セクションを指定する方法を示しています。この例は、Amazon Linux、Ubuntu Server、RHEL インスタンスにのみ適用されます。この例では、次のファイルとフォルダが、この階層のインスタンスにコピーされることを前提としています。

```
/tmp
  `-- my-app
       |-- my-file-1.txt
       |-- my-file-2.txt
       |-- my-file-3.txt
       |-- my-folder-1
       |     |-- my-file-4.txt
       |     |-- my-file-5.txt
       |     `-- my-file-6.txt
       `-- my-folder-2
             |-- my-file-7.txt
             |-- my-file-8.txt
             |-- my-file-9.txt
            `-- my-folder-3
```

次の AppSpec ファイルは、コピー後にこれらのファイルとフォルダでアクセス権限を設定する方法を示しています。

```
version: 0.0
os: linux
# Copy over all of the folders and files with the permissions they
#  were originally assigned.
files:
  - source: ./my-file-1.txt
    destination: /tmp/my-app
  - source: ./my-file-2.txt
    destination: /tmp/my-app
  - source: ./my-file-3.txt
    destination: /tmp/my-app
  - source: ./my-folder-1
    destination: /tmp/my-app/my-folder-1
```

```
  - source: ./my-folder-2
    destination: /tmp/my-app/my-folder-2
# 1) For all of the files in the /tmp/my-app folder ending in -3.txt
#  (for example, just my-file-3.txt), owner = adm, group = wheel, and
#  mode = 464 (-r--rw-r--).
permissions:
  - object: /tmp/my-app
    pattern: "*-3.txt"
    owner: adm
    group: wheel
    mode: 464
    type:
      - file
# 2) For all of the files ending in .txt in the /tmp/my-app
#  folder, but not for the file my-file-3.txt (for example,
#  just my-file-1.txt and my-file-2.txt),
#  owner = ec2-user and mode = 444 (-r--r--r--).
  - object: /tmp/my-app
    pattern: "*.txt"
    except: [my-file-3.txt]
    owner: ec2-user
    mode: 444
    type:
      - file
# 3) For all the files in the /tmp/my-app/my-folder-1 folder except
#  for my-file-4.txt and my-file-5.txt, (for example,
#  just my-file-6.txt), owner = operator and mode = 646 (-rw-r--rw-).
  - object: /tmp/my-app/my-folder-1
    pattern: "**"
    except: [my-file-4.txt, my-file-5.txt]
    owner: operator
    mode: 646
    type:
      - file
# 4) For all of the files that are immediately under
#  the /tmp/my-app/my-folder-2 folder except for my-file-8.txt,
#  (for example, just my-file-7.txt and
#  my-file-9.txt), owner = ec2-user and mode = 777 (-rwxrwxrwx).
  - object: /tmp/my-app/my-folder-2
    pattern: "**"
    except: [my-file-8.txt]
    owner: ec2-user
    mode: 777
    type:
```

```
      - file
# 5) For all folders at any level under /tmp/my-app that contain
#  the name my-folder but not
#  /tmp/my-app/my-folder-2/my-folder-3 (for example, just
#  /tmp/my-app/my-folder-1 and /tmp/my-app/my-folder-2),
#  owner = ec2-user and mode = 555 (dr-xr-xr-x).
  - object: /tmp/my-app
    pattern: "*my-folder*"
    except: [tmp/my-app/my-folder-2/my-folder-3]
    owner: ec2-user
    mode: 555
    type:
      - directory
# 6) For the folder /tmp/my-app/my-folder-2/my-folder-3,
#  group = wheel and mode = 564 (dr-xrw-r--).
  - object: /tmp/my-app/my-folder-2/my-folder-3
    group: wheel
    mode: 564
    type:
      - directory
```

作成されるアクセス権限は次のとおりです。

```
-r--r--r-- ec2-user root  my-file-1.txt
-r--r--r-- ec2-user root  my-file-2.txt
-r--rw-r-- adm      wheel my-file-3.txt

dr-xr-xr-x ec2-user root  my-folder-1
-rw-r--r-- root     root  my-file-4.txt
-rw-r--r-- root     root  my-file-5.txt
-rw-r--rw- operator root  my-file-6.txt

dr-xr-xr-x ec2-user root  my-folder-2
-rwxrwxrwx ec2-user root  my-file-7.txt
-rw-r--r-- root     root  my-file-8.txt
-rwxrwxrwx ec2-user root  my-file-9.txt

dr-xrw-r-- root     wheel my-folder-3
```

次の例では、「acls」および「context」の手順を追加して、'permissions' セクションを指定する方法を示します。この例は、Amazon Linux、Ubuntu Server、RHEL インスタンスにのみ適用されます。

```
permissions:
  - object: /var/www/html/WordPress
    pattern: "**"
    except: [/var/www/html/WordPress/ReadMe.txt]
    owner: bob
    group: writers
    mode: 644
    acls:
      - u:mary:rw
      - u:sam:rw
      - m::rw
    context:
      user: unconfined_u
      type: httpd_sys_content_t
      range: s0
    type:
      - file
```

AppSpec の「hooks」セクション

AppSpec ファイルの 'hooks' セクションの内容は、デプロイのコンピューティングプラットフォームによって異なります。EC2/オンプレミスのデプロイの 'hooks' セクションには、デプロイライフサイクルイベントフックを 1 つ以上のスクリプトにリンクするマッピングが含まれます。Lambda または Amazon ECS のデプロイの 'hooks' セクションは、デプロイライフサイクルイベント中に実行する Lambda 検証の関数を指定します。イベントフックが存在しない場合、そのイベントに対してオペレーションは実行されません。このセクションは、デプロイの一部としてスクリプトまたは Lambda 検証の関数を実行する場合のみ必須です。

トピック

- Amazon ECS のデプロイ向けの AppSpec の「hooks」セクション
- AWS Lambda デプロイの AppSpec 'hooks' セクション
- EC2/オンプレミスのデプロイ向けの AppSpec の「hooks」セクション

Amazon ECS のデプロイ向けの AppSpec の「hooks」セクション

トピック

- Amazon ECS のデプロイ向けのライフサイクルイベントフックのリスト
- Amazon ECS デプロイでのフックの実行順序。

- 「hooks」 セクションの構造
- Lambda の「フック」関数のサンプル

Amazon ECS のデプロイ向けのライフサイクルイベントフックのリスト

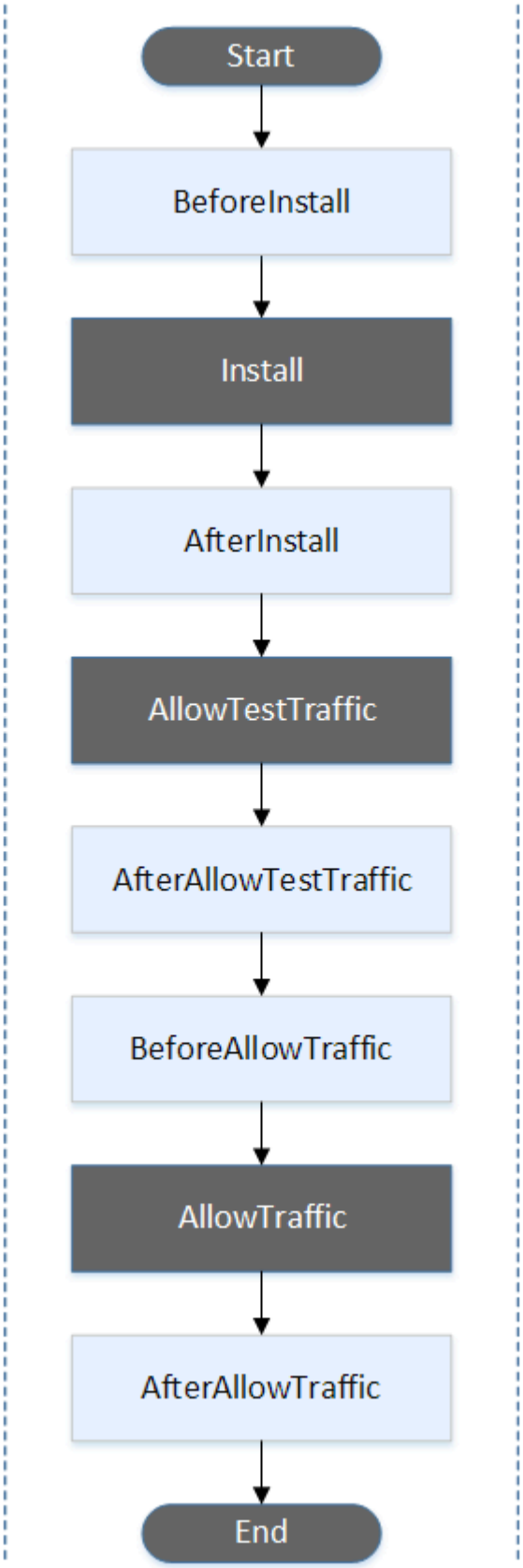
AWS Lambda フックは、ライフサイクルイベントの名前の後に新しい行に文字列で指定された 1 つの Lambda 関数です。各フックはデプロイごとに 1 回実行されます。以下は、Amazon ECS デプロイ中にフックを実行できるライフサイクルイベントの説明です。

- `BeforeInstall` 置き換えタスクセットが作成される前にタスクを実行するために使用します。1 つのターゲットグループが元のタスクセットに関連付けられています。オプションのテストリスナーが指定されている場合、それは元のタスクセットに関連付けられます。この時点で、ロールバックはできません。
- `AfterInstall` 置き換えタスクセットが作成され、ターゲットグループの 1 つがそれに関連付けられた後、タスクを実行するために使用します。オプションのテストリスナーが指定されている場合、それは元のタスクセットに関連付けられます。このライフサイクルイベントでのフック関数の結果により、ロールバックをトリガーできます。
- `AfterAllowTestTraffic` テストリスナーが置き換えタスクセットにトラフィックを提供した後、タスクを実行するために使用します。この時点でのフック関数の結果により、ロールバックをトリガーできます。
- `BeforeAllowTraffic` 2 番目のターゲットグループが置き換えタスクセットに関連付けられた後、かつ、トラフィックが置き換えタスクセットに移行される前に、タスクを実行するために使用します。このライフサイクルイベントでのフック関数の結果により、ロールバックをトリガーできます。
- `AfterAllowTraffic` 2 番目のターゲットグループが置き換えタスクセットにトラフィックを提供した後、タスクを実行するために使用します。このライフサイクルイベントでのフック関数の結果により、ロールバックをトリガーできます。

詳細については、「Amazon ECS デプロイ中の処理で起こっていること」および「チュートリアル: 検証テストを使用して Amazon ECS サービスをデプロイする」を参照してください。

Amazon ECS デプロイでのフックの実行順序。

Amazon ECS デプロイでは、イベントフックは次の順序で実行されます。


Start
BeforeInstall
Install
AfterInstall
AllowTestTraffic
AfterAllowTestTraffic
BeforeAllowTraffic
AllowTraffic
AfterAllowTraffic
End

> **Note**
>
> デプロイの Start、Install、TestTraffic、AllowTraffic、および End イベントはスクリプト化できないため、この図ではグレーで表示されています。

「hooks」 セクションの構造

次の例は、'hooks' セクションの構造の例を示します。

YAML の使用:

```
Hooks:
  - BeforeInstall: "BeforeInstallHookFunctionName"
  - AfterInstall: "AfterInstallHookFunctionName"
  - AfterAllowTestTraffic: "AfterAllowTestTrafficHookFunctionName"
  - BeforeAllowTraffic: "BeforeAllowTrafficHookFunctionName"
  - AfterAllowTraffic: "AfterAllowTrafficHookFunctionName"
```

JSON の使用:

```
"Hooks": [
  {
   "BeforeInstall": "BeforeInstallHookFunctionName"
  },
  {
   "AfterInstall": "AfterInstallHookFunctionName"
  },
  {
   "AfterAllowTestTraffic": "AfterAllowTestTrafficHookFunctionName"
  },
  {
   "BeforeAllowTraffic": "BeforeAllowTrafficHookFunctionName"
  },
  {
   "AfterAllowTraffic": "AfterAllowTrafficHookFunctionName"
  }
 ]
}
```

Lambda の「フック」関数のサンプル

'hooks' セクションを使用して、Amazon ECS のデプロイを検証するために CodeDeploy が呼び出すことができる Lambda 関数を指定します。BeforeInstall、AfterInstall、AfterAllowTestTraffic、BeforeAllowTraffic、および AfterAllowTraffic のデプロイライフサイクルイベントに対して、同じ関数または異なる関数を使用できます。検証テストが完了すると、Lambda AfterAllowTraffic 関数は CodeDeploy を呼び戻し、Succeeded または Failed の結果を配信します。

> **Important**
>
> 1 時間以内に CodeDeploy が Lambda 検証関数から通知されない場合、デプロイは失敗したと見なされます。

Lambda フック関数を呼び出す前に、サーバーは putLifecycleEventHookExecutionStatus コマンドを使用して、デプロイ ID およびライフサイクルイベントフック実行 ID について通知される必要があります。

次に示すのは、Node.js で記述されたサンプルの Lambda フック関数の例です。

```
'use strict';

const aws = require('aws-sdk');
const codedeploy = new aws.CodeDeploy({apiVersion: '2014-10-06'});

exports.handler = (event, context, callback) => {
    //Read the DeploymentId from the event payload.
    var deploymentId = event.DeploymentId;

    //Read the LifecycleEventHookExecutionId from the event payload
    var lifecycleEventHookExecutionId = event.LifecycleEventHookExecutionId;

    /*
     Enter validation tests here.
    */

    // Prepare the validation test results with the deploymentId and
    // the lifecycleEventHookExecutionId for CodeDeploy.
    var params = {
        deploymentId: deploymentId,
```

```
        lifecycleEventHookExecutionId: lifecycleEventHookExecutionId,
        status: 'Succeeded' // status can be 'Succeeded' or 'Failed'
    };

    // Pass CodeDeploy the prepared validation test results.
    codedeploy.putLifecycleEventHookExecutionStatus(params, function(err, data) {
        if (err) {
            // Validation failed.
            callback('Validation test failed');
        } else {
            // Validation succeeded.
            callback(null, 'Validation test succeeded');
        }
    });
};
```

AWS Lambda デプロイの AppSpec 'hooks' セクション

トピック

- AWS Lambda デプロイのライフサイクルイベントフックのリスト
- Lambda 関数のバージョンのデプロイでのフックの実行順
- 「hooks」 セクションの構造
- Lambda の「フック」関数のサンプル

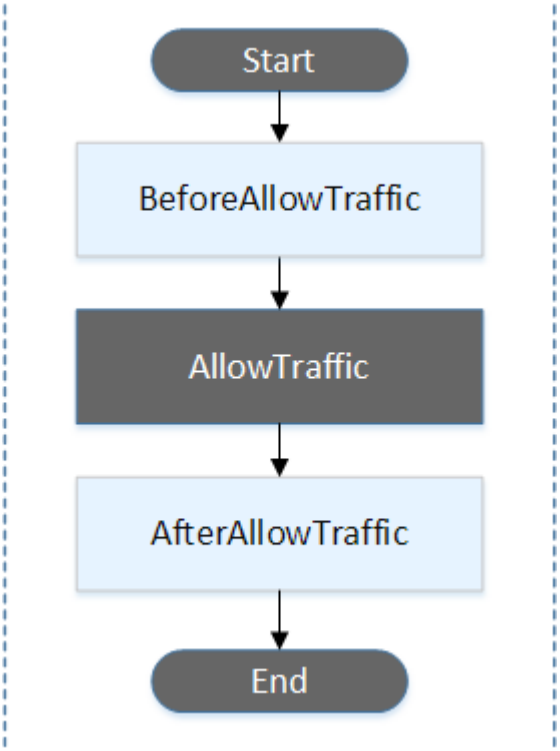
AWS Lambda デプロイのライフサイクルイベントフックのリスト

AWS Lambda フックは、ライフサイクルイベントの名前の後に新しい行に文字列で指定された 1 つの Lambda 関数です。各フックはデプロイごとに 1 回実行されます。以下は、AppSpec ファイルに使用できるフックの説明です。

- BeforeAllowTraffic – これを使用して、トラフィックがデプロイされた Lambda 関数のバージョンに移行する前にタスクを実行します。
- AfterAllowTraffic – これを使用して、トラフィックがデプロイされた Lambda 関数のバージョンに移行した後でタスクを実行します。

Lambda 関数のバージョンのデプロイでのフックの実行順

サーバーレスの Lambda 関数のバージョンのデプロイでは、イベントフックは次の順序で実行されます。



> **Note**
>
> デプロイの Start、AllowTraffic、および End イベントはスクリプト化できません。このため、これらのイベントはこの図でグレーで表示されています。

「hooks」 セクションの構造

次の例は、「hooks」セクションの例を示します。

YAML の使用:

```
hooks:
   - BeforeAllowTraffic: BeforeAllowTrafficHookFunctionName
   - AfterAllowTraffic: AfterAllowTrafficHookFunctionName
```

JSON の使用:

```
"hooks": [{
    "BeforeAllowTraffic": "BeforeAllowTrafficHookFunctionName"
    },
    {
```

```
    "AfterAllowTraffic": "AfterAllowTrafficHookFunctionName"
}]
```

Lambda の「フック」関数のサンプル

「hooks」セクションを使用して、Lambda のデプロイを検証するために CodeDeploy が呼び出すことができる Lambda 関数を指定します。`BeforeAllowTraffic` および `AfterAllowTraffic` デプロイライフサイクルイベントには、同じ関数または異なる関数を使用できます。検証テストが完了すると、Lambda 検証関数は CodeDeploy を呼び戻し、`Succeeded` または `Failed` の結果を配信します。

> **Important**
>
> 1 時間以内に CodeDeploy が Lambda 検証関数から通知されない場合、デプロイは失敗したと見なされます。

Lambda フック関数を呼び出す前に、サーバーは `putLifecycleEventHookExecutionStatus` コマンドを使用して、デプロイ ID およびライフサイクルイベントフック実行 ID について通知される必要があります。

次に示すのは、Node.js で記述されたサンプルの Lambda フック関数の例です。

```
'use strict';

const aws = require('aws-sdk');
const codedeploy = new aws.CodeDeploy({apiVersion: '2014-10-06'});

exports.handler = (event, context, callback) => {
    //Read the DeploymentId from the event payload.
    var deploymentId = event.DeploymentId;

    //Read the LifecycleEventHookExecutionId from the event payload
    var lifecycleEventHookExecutionId = event.LifecycleEventHookExecutionId;

    /*
     Enter validation tests here.
    */

    // Prepare the validation test results with the deploymentId and
    // the lifecycleEventHookExecutionId for CodeDeploy.
```

```
    var params = {
        deploymentId: deploymentId,
        lifecycleEventHookExecutionId: lifecycleEventHookExecutionId,
        status: 'Succeeded' // status can be 'Succeeded' or 'Failed'
    };

    // Pass CodeDeploy the prepared validation test results.
    codedeploy.putLifecycleEventHookExecutionStatus(params, function(err, data) {
        if (err) {
            // Validation failed.
            callback('Validation test failed');
        } else {
            // Validation succeeded.
            callback(null, 'Validation test succeeded');
        }
    });
};
```

EC2/オンプレミスのデプロイ向けの AppSpec の「hooks」セクション

トピック

- ライフサイクルイベントフックのリスト
- ライフサイクルイベントフックの可用性
- デプロイでのフックの実行順
- 「hooks」 セクションの構造
- フックスクリプトでのファイルの参照
- フックの環境変数の可用性
- hooks の例

ライフサイクルイベントフックのリスト

EC2/オンプレミスのデプロイのフックは、デプロイごとに 1 回インスタンスに対して実行されます。フックには実行するスクリプトを 1 つまたは複数指定することができます。ライフサイクルイベントの各フックは、文字列で個別の行に指定します。以下は、AppSpec ファイルに使用できるフックの説明です。

デプロイおよびロールバックの種類別の有効なライフサイクルフックの詳細については、「ライフサイクルイベントフックの可用性」を参照してください。

- ApplicationStop このデプロイライフサイクルイベントは、アプリケーションリビジョンがダウンロードされる前でも発生します。アプリケーションを適切に中止するか、現在インストールされているパッケージを削除してデプロイの準備をする場合は、このイベントのスクリプトを指定できます。このデプロイライフサイクルイベントに使用される AppSpec ファイルとスクリプトは、前回正常にデプロイされたアプリケーションリビジョンのものです。

  > **Note**
  >
  > AppSpec ファイルは、デプロイする前にはインスタンスに存在しません。したがって、ApplicationStop フックは、初めてインスタンスにデプロイするときは実行されません。インスタンスに 2 回目にデプロイするときは、ApplicationStop フックを使用できます。

  最後に正常にデプロイされたアプリケーションリビジョンの場所を特定するため、CodeDeploy エージェントは *deployment-group-id*_last_successful_install ファイルにリストされた場所を探します。このファイルは次の場所にあります。

  Amazon Linux、Ubuntu Server、RHEL Amazon EC2 インスタンスの /opt/codedeploy-agent/deployment-root/deployment-instructions フォルダ。

  Windows Server の Amazon EC2 インスタンスの C:\ProgramData\Amazon\CodeDeploy\deployment-instructions フォルダ。

  ApplicationStop デプロイライフサイクルイベント中に失敗するデプロイをトラブルシューティングするには、「失敗した ApplicationStop、BeforeBlockTraffic、または AfterBlockTraffic デプロイライフサイクルイベントのトラブルシューティング」を参照してください。

- DownloadBundle このデプロイライフサイクルイベント中に、CodeDeploy エージェントはアプリケーションリビジョンファイルを一時的な場所にコピーします。

  Amazon Linux、Ubuntu Server、RHEL Amazon EC2 インスタンスの /opt/codedeploy-agent/deployment-root/*deployment-group-id*/*deployment-id*/deployment-archive フォルダ。

  Windows Server の Amazon EC2 インスタンスの C:\ProgramData\Amazon\CodeDeploy*deployment-group-id**deployment-id*\deployment-archive フォルダ。

このイベントは CodeDeploy エージェント用に予約されていて、スクリプトを実行するために使用することはできません。

DownloadBundle デプロイライフサイクルイベント中に失敗するデプロイをトラブルシューティングするには、「「不明なエラー: 読み取り用に開いていません」で失敗した DownloadBundle デプロイライフサイクルイベントのトラブルシューティング」を参照してください。

- BeforeInstall このデプロイライフサイクルイベントは、ファイルの復号や現在のバージョンのバックアップの作成などの事前インストールタスクに使用できます。
- Install このデプロイのライフサイクルイベントでは、CodeDeploy エージェントが一時的なロケーションからリビジョンファイルを最終的な送信先フォルダにコピーします。このイベントは CodeDeploy エージェント用に予約されていて、スクリプトを実行するために使用することはできません。
- AfterInstall アプリケーションの設定やファイルのアクセス許可の変更などのタスクに、このデプロイライフサイクルイベントを使用できます。
- ApplicationStart 通常、このデプロイライフサイクルイベントを使用して、ApplicationStop 中に停止されたサービスを再起動します。
- ValidateService これが最後のデプロイライフサイクルイベントです。デプロイが正常に完了したことを確認するために使用されます。
- BeforeBlockTraffic このデプロイライフサイクルイベントを使用して、ロードバランサーから登録解除される前のインスタンスでタスクを実行できます。

  BeforeBlockTraffic デプロイライフサイクルイベント中に失敗するデプロイをトラブルシューティングするには、「失敗した ApplicationStop、BeforeBlockTraffic、または AfterBlockTraffic デプロイライフサイクルイベントのトラブルシューティング」を参照してください。
- BlockTraffic このデプロイライフサイクルイベント中は、現在トラフィックの処理中であるインスタンスに対するインターネットトラフィックのアクセスがブロックされます。このイベントは CodeDeploy エージェント用に予約されていて、スクリプトを実行するために使用することはできません。
- AfterBlockTraffic このデプロイライフサイクルイベントを使用して、それぞれのロードバランサーから登録解除された後のインスタンスでタスクを実行できます。

  AfterBlockTraffic デプロイライフサイクルイベント中に失敗するデプロイをトラブルシューティングするには、「失敗した ApplicationStop、BeforeBlockTraffic、または AfterBlockTraffic デプロイライフサイクルイベントのトラブルシューティング」を参照してください。

- `BeforeAllowTraffic` このデプロイライフサイクルイベントを使用して、ロードバランサーに登録される前のインスタンスでタスクを実行できます。
- `AllowTraffic` このデプロイライフサイクルイベント中は、デプロイ後のインスタンスに対するインターネットトラフィックのアクセスが許可されます。このイベントは CodeDeploy エージェント用に予約されていて、スクリプトを実行するために使用することはできません。
- `AfterAllowTraffic` このデプロイライフサイクルイベントを使用して、ロードバランサーに登録された後のインスタンスでタスクを実行できます。

ライフサイクルイベントフックの可用性

次の表に、各デプロイおよびロールバックシナリオで使用できるライフサイクルイベントフックを示します。

ライフサイクルイベント名	Auto Scaling 起動デプロイ[1]	Auto Scaling 終了デプロイ[1]	インプレースデプロイ[1]	Blue/Green デプロイ: 元のインスタンス	Blue/Green デプロイ: 代替インスタンス	Blue/Green デプロイのロールバック: 元のインスタンス	Blue/Green デプロイのロールバック: 代替インスタンス
ApplicationStop	✓	✓	✓		✓		
DownloadBundle[3]	✓		✓		✓		
BeforeInstall	✓		✓		✓		
Install[3]	✓		✓		✓		
AfterInstall	✓		✓		✓		
ApplicationStart	✓		✓		✓		

ライフサイクルイベント名	Auto Scaling 起動デプロイ[1]	Auto Scaling 終了デプロイ[1]	インプレースデプロイ[1]	Blue/Green デプロイ: 元のインスタンス	Blue/Green デプロイ: 代替インスタンス	Blue/Green デプロイのロールバック: 元のインスタンス	Blue/Green デプロイのロールバック: 代替インスタンス
ValidateService	✓		✓		✓		
BeforeBlockTraffic		✓	✓	✓			✓
BlockTraffic[3]		✓	✓	✓			✓
AfterBlockTraffic		✓	✓	✓			✓
BeforeAllowTraffic	✓		✓		✓	✓	
AllowTraffic[3]	✓		✓		✓	✓	
AfterAllowTraffic	✓		✓		✓	✓	

[1] Amazon EC2 Auto Scaling デプロイの詳細については、「Amazon EC2 Auto Scaling と CodeDeploy の連携」を参照してください。

[2] インプレースデプロイのロールバックにも適用されます。

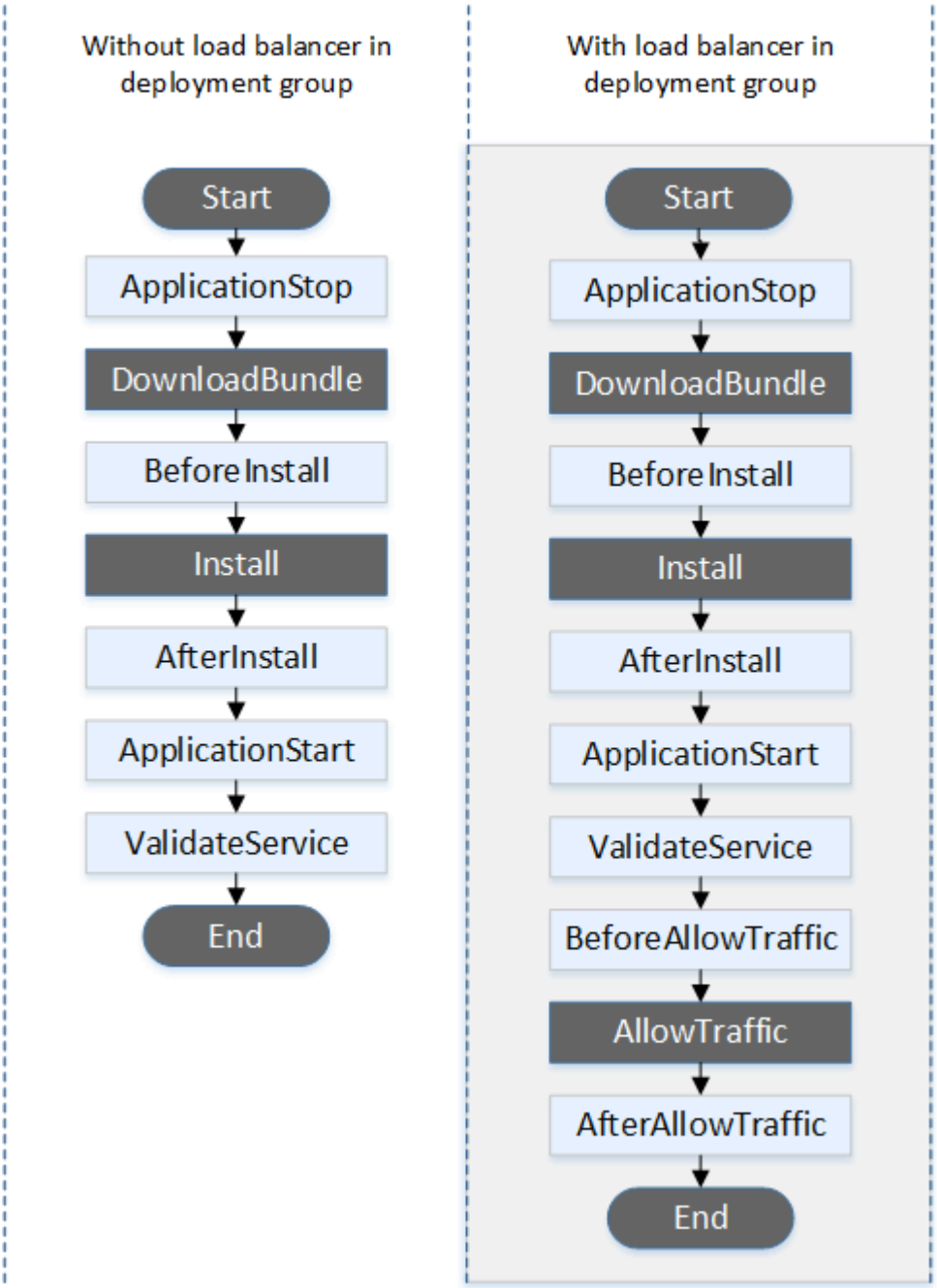
[3] CodeDeploy オペレーション用に予約されています。スクリプトの実行には使用できません。

デプロイでのフックの実行順

Auto Scaling 起動デプロイ

Auto Scaling 起動デプロイ中に、CodeDeploy は次の順序でイベントフックを実行します。

Auto Scaling 起動デプロイの詳細については、「Amazon EC2 Auto Scaling と CodeDeploy の連携」を参照してください。



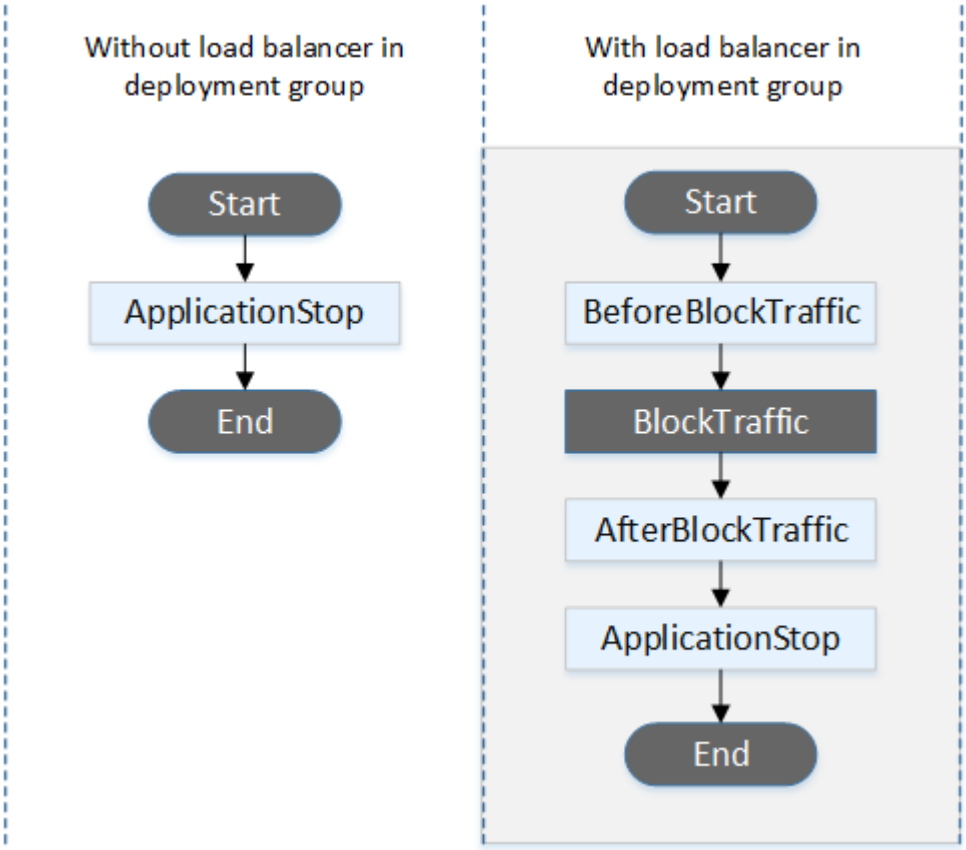
> **Note**
>
> デプロイの Start、DownloadBundle、Install、AllowTraffic、End の各イベントはスクリプト化できないため、この図ではグレーで表示されています。ただし、AppSpec ファイルの

'files' セクションを編集して、Install イベント中にインストールされるものを指定できます。

Auto Scaling 終了デプロイ

Auto Scaling 終了デプロイ中に、CodeDeploy は次の順序でイベントフックを実行します。

Auto Scaling 終了デプロイの詳細については、「Auto Scaling スケールインイベント中の終了デプロイの有効化」を参照してください。



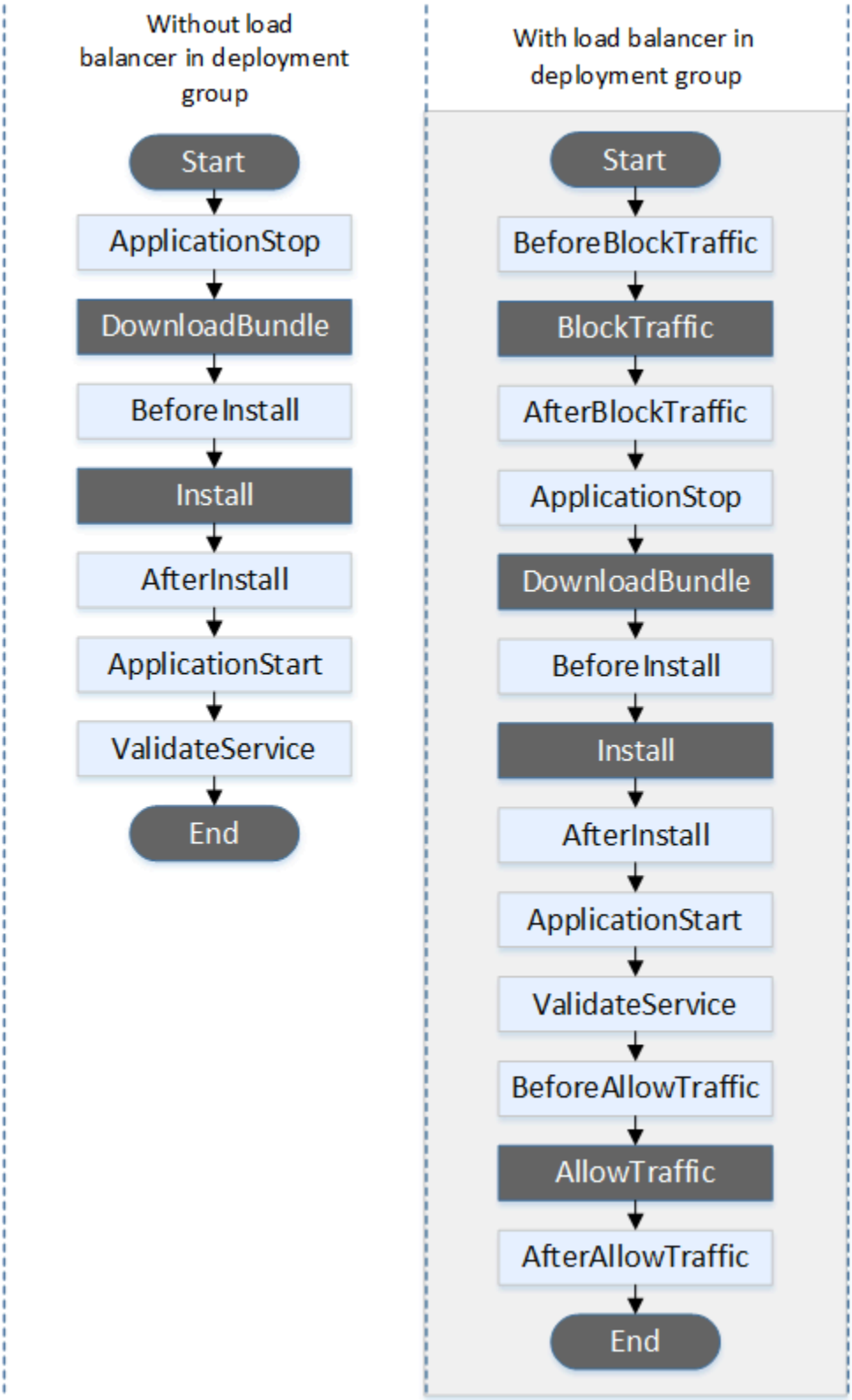
Note

デプロイの Start、BlockTraffic、End の各イベントはスクリプト化できないため、この図ではグレーで表示されています。

インプレースデプロイ

インプレースデプロイのロールバックを含むインプレースデプロイで、イベントフックは次の順序で実行されます。

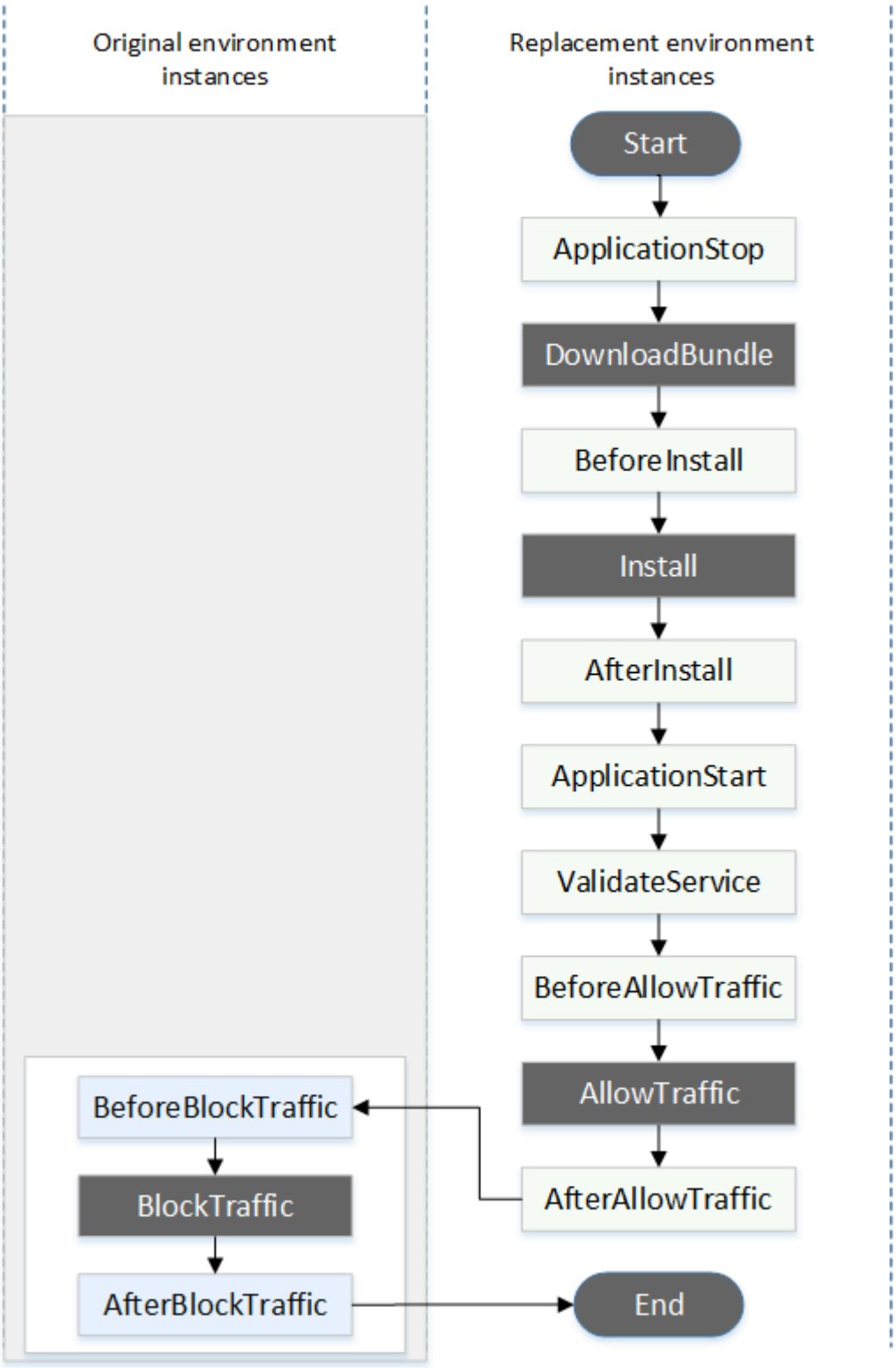
> **Note**
>
> インプレースデプロイの場合、トラフィックのブロックと許可に関する 6 つのフックは、デプロイグループに Elastic Load Balancing から Classic Load Balancer、Application Load Balancer、または Network Load Balancer を指定した場合のみ適用されます。


Without load balancer in deployment group
Start
ApplicationStop
DownloadBundle
BeforeInstall
Install
AfterInstall
ApplicationStart
ValidateService
End
With load balancer in deployment group
Start
BeforeBlockTraffic
BlockTraffic
AfterBlockTraffic
ApplicationStop
DownloadBundle
BeforeInstall
Install
AfterInstall
ApplicationStart
ValidateService
BeforeAllowTraffic
AllowTraffic
AfterAllowTraffic
End

> **Note**
>
> デプロイの Start、DownloadBundle、Install、および End イベントはスクリプト化できないため、この図ではグレーで表示されています。ただし、AppSpec ファイルの `'files'` セクションを編集して、Install イベント中にインストールされるものを指定できます。

Blue/Green デプロイ

Blue/Green デプロイでは、イベントフックは次の順序で実行されます。


Original environment instances
Replacement environment instances
Start
ApplicationStop
DownloadBundle
BeforeInstall
Install
AfterInstall
ApplicationStart
ValidateService
BeforeAllowTraffic
AllowTraffic
AfterAllowTraffic
BeforeBlockTraffic
BlockTraffic
AfterBlockTraffic
End

> **Note**
>
> デプロイの Start、DownloadBundle、Install、BlockTraffic、AllowTraffic、および End イベントはスクリプト化できないため、この図ではグレーで表示されています。ただし、AppSpec ファイルの「files」セクションを編集して、Install イベント中にインストールされるものを指定できます。

「hooks」 セクションの構造

'hooks' セクションは以下の構造を持ちます。

```
hooks:
   deployment-lifecycle-event-name:
     - location: script-location
       timeout: timeout-in-seconds
       runas: user-name
```

デプロイライフサイクルイベント名の後で、次の要素を hook エントリに含めることができます。

location

必須。リビジョンのスクリプトファイルのバンドルでの位置。hooks セクションで指定するスクリプトの場所は、アプリケーションリビジョンバンドルのルートから相対的な位置です。詳細については、「CodeDeploy のリビジョンを計画する」を参照してください。

タイムアウト

オプション。失敗と見なされる前にスクリプトの実行を許可する秒数。デフォルト値は 3600 秒 (1 時間) です。

> **Note**
>
> 3600 秒 (1 時間) は、各デプロイライフサイクルイベントのスクリプト実行で許可される最大の時間です。スクリプトがこの制限を超過した場合、デプロイは停止し、インスタンスへのデプロイは失敗します。各デプロイライフサイクルイベントのすべてのスクリプトで、timeout に指定された合計秒数が、この制限を超えないようにします。

runas

オプション。スクリプトの実行時に偽装するユーザー。デフォルトでは、インスタンス上で実行されている CodeDeploy エージェントです。CodeDeploy はパスワードを保存しないため、runas ユーザーがパスワードを必要とする場合、ユーザーになりすますことはできません。この要素は、Amazon Linux、Ubuntu Server、RHEL インスタンスにのみ適用されます。

フックスクリプトでのファイルの参照

AppSpec の「hooks」セクションで説明しているように、スクリプトを CodeDeploy ライフサイクルイベントにフックし、スクリプト内のファイル (`helper.sh` など) を参照する場合は、以下を使用して `helper.sh` を指定する必要があります。

- (推奨) 絶対パス。「絶対パスの使用」を参照してください。
- 相対パス。「相対パスの使用」を参照してください。

絶対パスの使用

絶対パスを使用してファイルを参照するには、次のいずれかを行うことができます。

- AppSpec ファイルの `files` セクションの `destination` プロパティに絶対パスを指定します。次に、フックスクリプトに同じ絶対パスを指定します。詳細については、「AppSpec の「ファイル」セクション (EC2/オンプレミスデプロイのみ)」を参照してください。
- フックスクリプトに動的絶対パスを指定します。詳細については、「デプロイアーカイブの場所」を参照してください。

デプロイアーカイブの場所

DownloadBundle ライフサイクルイベント中に、CodeDeploy エージェントはデプロイ用のリビジョンを次の形式のディレクトリに抽出します。

root-directory/*deployment-group-id*/*deployment-id*/deployment-archive

パスの *root-directory* 部分は、常に次の表に示すデフォルトに設定されるか、`:root_dir` 構成設定によって制御されます。構成設定の詳細については、「CodeDeploy エージェント設定リファレンス」を参照してください。

エージェントプラットフォーム	デフォルトのルートディレクトリ
Linux — すべての rpm ディストリビューション	`/opt/codedeploy-agent/deployment-root`
Ubuntu サーバー — すべての deb ディストリビューション	`/opt/codedeploy-agent/deployment-root`
Windows サーバー	`%ProgramData%\Amazon\CodeDeploy`

フックスクリプトから、ルートディレクトリパスと DEPLOYMENT_IDおよびDEPLOYMENT_GROUP_ID環境変数を使用して、現在のデプロイアーカイブにアクセスできます。使用できる変数の詳細については、「フックの環境変数の可用性」を参照してください。

例えば、Linux のリビジョンのルートにある data.json ファイルにアクセスする方法は次のとおりです。

```
#!/bin/bash

rootDirectory="/opt/codedeploy-agent/deployment-root" # note: this will be different if
 you
                                                      # customize the :root_dir
 configuration
dataFile="$rootDirectory/$DEPLOYMENT_GROUP_ID/$DEPLOYMENT_ID/deployment-archive/
data.json"
data=$(cat dataFile)
```

別の例として、Windows の Powershell を使用してリビジョンのルートにある data.json ファイルにアクセスする方法は次のとおりです。

```
$rootDirectory="$env:ProgramData\Amazon\CodeDeploy" # note: this will be different if
 you
                                                    # customize the :root_dir
 configuration
$dataFile="$rootDirectory\$env:DEPLOYMENT_GROUP_ID\$env:DEPLOYMENT_ID\deployment-
archive\data.json"
$data=(Get-Content $dataFile)
```

相対パスの使用

相対パスを使用してファイルを参照するには、CodeDeploy エージェントの作業ディレクトリを知っている必要があります。ファイルパスは、このディレクトリからの相対パスです。

次の表は、CodeDeploy エージェントのサポートされているプラットフォームごとの作業ディレクトリを示しています。

エージェントプラットフォーム	プロセス管理メソッド	ライフサイクルイベントスクリプトの作業ディレクトリ
Linux — すべての rpm ディストリビューション	systemd (デフォルト)	`/`
	init.d — 詳細はこちら	`/opt/codedeploy-agent`
Ubuntu サーバー — すべての debian ディストリビューション	すべて	`/opt/codedeploy-agent`
Windows サーバー	該当なし	`C:\Windows\System32`

フックの環境変数の可用性

各デプロイライフサイクルイベントの間、フックスクリプトは次の環境変数にアクセスできます。

APPLICATION_NAME

現在のデプロイの一部である CodeDeploy のアプリケーションの名前 (`WordPress_App` など)。

DEPLOYMENT_ID

CodeDeploy が、現在のデプロイに割り当てられた ID (例えば、`d-AB1CDEF23`)。

DEPLOYMENT_GROUP_NAME

現在のデプロイの一部である CodeDeploy のデプロイグループの名前 (`WordPress_DepGroup` など)。

DEPLOYMENT_GROUP_ID

現在のデプロイの一部である CodeDeploy のデプロイグループの ID (`b1a2189b-dd90-4ef5-8f40-4c1c5EXAMPLE` など)。

LIFECYCLE_EVENT

現在のデプロイライフサイクルイベントの名前 (例: `AfterInstall`)。

これらの環境変数は各デプロイライフサイクルイベントにローカルです。

デプロイバンドルのソースに応じて、スクリプトをフックできる環境変数が他にもあります。

Amazon S3 からのバンドル

- BUNDLE_BUCKET

  デプロイバンドルがダウンロードされた Amazon S3 バケットの名前 (例: `my-s3-bucket`)。
- BUNDLE_KEY

  Amazon S3 バケット内のダウンロードされたバンドル用のオブジェクトキー (例: `WordPress_App.zip`)。
- BUNDLE_VERSION

  バンドル用のオブジェクトバージョン (例: `3sL4kqtJlcpXroDTDmJ+rmSpXd3dIbrHY+MTRCxf3vjVBH40Nr8X8gdRQBpUMLUo`)。この変数は Amazon S3 バケットでオブジェクトバージョニングが有効になっている場合にのみ設定されます。
- BUNDLE_ETAG

  バンドル用のオブジェクト Etag (例: `b10a8db164e0754105b7a99be72e3fe5-4`)。

GitHub からのバンドル

- BUNDLE_COMMIT

  Git によって生成されたバンドルの SHA256 コミットハッシュ (例: `d2a84f4b8b650937ec8f73cd8be2c74add5a911ba64df27458ed8229da804a26`)。

以下のスクリプトは、DEPLOYMENT_GROUP_NAME の値が `Staging` と等しい場合に、Apache HTTP サーバーでリッスンするポートを 80 ではなく 9090 に変更します。このスクリプトは `BeforeInstall` デプロイライフサイクルイベント中に呼び出される必要があります。

```
if [ "$DEPLOYMENT_GROUP_NAME" == "Staging" ]
then
```

```
    sed -i -e 's/Listen 80/Listen 9090/g' /etc/httpd/conf/httpd.conf
fi
```

次のスクリプトの例では、DEPLOYMENT_GROUP_NAME 環境変数の値が Staging に等しい場合に、エラーログに記録されるメッセージの詳細レベルを警告からデバッグに変更します。このスクリプトは BeforeInstall デプロイライフサイクルイベント中に呼び出される必要があります。

```
if [ "$DEPLOYMENT_GROUP_NAME" == "Staging" ]
then
    sed -i -e 's/LogLevel warn/LogLevel debug/g' /etc/httpd/conf/httpd.conf
fi
```

以下のスクリプトの例では、指定されたウェブページを、これらの環境変数の値を表示するテキストで置き換えます。このスクリプトは AfterInstall デプロイライフサイクルイベント中に呼び出される必要があります。

```
#!/usr/bin/python

import os

strToSearch="<h2>This application was deployed using CodeDeploy.</h2>"
strToReplace="<h2>This page for "+os.environ['APPLICATION_NAME']+"
 application and "+os.environ['DEPLOYMENT_GROUP_NAME']+" deployment group with
 "+os.environ['DEPLOYMENT_GROUP_ID']+" deployment group ID was generated by a
 "+os.environ['LIFECYCLE_EVENT']+" script during "+os.environ['DEPLOYMENT_ID']+"
 deployment.</h2>"

fp=open("/var/www/html/index.html","r")
buffer=fp.read()
fp.close()

fp=open("/var/www/html/index.html","w")
fp.write(buffer.replace(strToSearch,strToReplace))
fp.close()
```

hooks の例

hooks エントリの例を次に示します。AfterInstall ライフサイクルイベントに 2 つのフックを指定しています。

```
hooks:
```

```
   AfterInstall:
     - location: Scripts/RunResourceTests.sh
       timeout: 180
     - location: Scripts/PostDeploy.sh
       timeout: 180
```

デプロイプロセスの AfterInstall ステージ中に、Scripts/RunResourceTests.sh スクリプトが実行されます。スクリプトの実行に 180 秒 (3 分) 以上かかる場合、デプロイは成功しません。

「hooks」セクションで指定するスクリプトの場所は、アプリケーションリビジョンバンドルのルートに相対的な位置です。前述の例では、RunResourceTests.sh という名前のファイルが Scripts という名前のディレクトリにあります。Scripts ディレクトリはバンドルのルートレベルにあります。詳細については、「CodeDeploy のリビジョンを計画する」を参照してください。

AppSpec ファイルの例

このトピックでは、 AWS Lambda および EC2/オンプレミスデプロイ用の AppSpec ファイルの例を示します。

トピック

- Amazon ECS デプロイの AppSpec ファイルの例
- Lambda AWS デプロイの AppSpec ファイルの例
- EC2/オンプレミスデプロイの AppSpec ファイルの例

Amazon ECS デプロイの AppSpec ファイルの例

Amazon ECS サービスをデプロイするために YAML で書かれた AppSpec ファイルの例を示します。

```
version: 0.0
Resources:
  - TargetService:
      Type: AWS::ECS::Service
      Properties:
        TaskDefinition: "arn:aws:ecs:us-east-1:111222333444:task-definition/my-task-
definition-family-name:1"
        LoadBalancerInfo:
          ContainerName: "SampleApplicationName"
          ContainerPort: 80
# Optional properties
```

```
        PlatformVersion: "LATEST"
        NetworkConfiguration:
          AwsvpcConfiguration:
            Subnets: ["subnet-1234abcd","subnet-5678abcd"]
            SecurityGroups: ["sg-12345678"]
            AssignPublicIp: "ENABLED"
        CapacityProviderStrategy:
          - Base: 1
            CapacityProvider: "FARGATE_SPOT"
            Weight: 2
          - Base: 0
            CapacityProvider: "FARGATE"
            Weight: 1
Hooks:
  - BeforeInstall: "LambdaFunctionToValidateBeforeInstall"
  - AfterInstall: "LambdaFunctionToValidateAfterInstall"
  - AfterAllowTestTraffic: "LambdaFunctionToValidateAfterTestTrafficStarts"
  - BeforeAllowTraffic: "LambdaFunctionToValidateBeforeAllowingProductionTraffic"
  - AfterAllowTraffic: "LambdaFunctionToValidateAfterAllowingProductionTraffic"
```

JSON で書かれた前述の例のバージョンを示します。

```
{
    "version": 0.0,
    "Resources": [
        {
            "TargetService": {
                "Type": "AWS::ECS::Service",
                "Properties": {
                    "TaskDefinition": "arn:aws:ecs:us-east-1:111222333444:task-
definition/my-task-definition-family-name:1",
                    "LoadBalancerInfo": {
                        "ContainerName": "SampleApplicationName",
                        "ContainerPort": 80
                    },
                    "PlatformVersion": "LATEST",
                    "NetworkConfiguration": {
                        "AwsvpcConfiguration": {
                            "Subnets": [
                                "subnet-1234abcd",
                                "subnet-5678abcd"
                            ],
                            "SecurityGroups": [
```

```
                            "sg-12345678"
                        ],
                        "AssignPublicIp": "ENABLED"
                    }
                },
                "CapacityProviderStrategy": [
                    {
                        "Base" : 1,
                        "CapacityProvider" : "FARGATE_SPOT",
                        "Weight" : 2
                    },
                    {
                        "Base" : 0,
                        "CapacityProvider" : "FARGATE",
                        "Weight" : 1
                    }
                ]
            }
        }
    }
  ],
  "Hooks": [
    {
      "BeforeInstall": "LambdaFunctionToValidateBeforeInstall"
    },
    {
      "AfterInstall": "LambdaFunctionToValidateAfterInstall"
    },
    {
      "AfterAllowTestTraffic": "LambdaFunctionToValidateAfterTestTrafficStarts"
    },
    {
      "BeforeAllowTraffic":
 "LambdaFunctionToValidateBeforeAllowingProductionTraffic"
    },
    {
      "AfterAllowTraffic":
 "LambdaFunctionToValidateAfterAllowingProductionTraffic"
    }
  ]
}
```

デプロイ中のイベントのシーケンスを次に示します。

1. 最新の Amazon ECS アプリケーションが置き換えタスクセットにインストールされる前に、LambdaFunctionToValidateBeforeInstall と呼ばれる Lambda 関数が実行されます。
2. 最新の Amazon ECS アプリケーションが置き換えタスクセットにインストールされた後、トラフィックを受信する前に、LambdaFunctionToValidateAfterInstall と呼ばれる Lambda 関数が実行されます。
3. 置き換えタスクセット上の Amazon ECS アプリケーションが、テストリスナーからのトラフィックの受信を開始した後、LambdaFunctionToValidateAfterTestTrafficStarts と呼ばれる Lambda 関数が実行されます。この関数は、デプロイが続行されるかどうかを判断するために、検証テストを実行する可能性があります。デプロイグループでテストリスナーを指定しない場合、このフックは無視されます。
4. AfterAllowTestTraffic フックの検証テストがすべて完了した後、かつ、最新の Amazon ECS アプリケーションに本稼働トラフィックが提供される前に、LambdaFunctionToValidateBeforeAllowingProductionTraffic と呼ばれる Lambda 関数が実行されます。
5. 置き換えタスクセット上の最新の Amazon ECS アプリケーションに本稼働トラフィックが提供された後に、LambdaFunctionToValidateAfterAllowingProductionTraffic と呼ばれる Lambda 関数が実行されます。

フック中に実行される Lambda 関数は、検証テストを実行したり、トラフィックメトリクスを収集したりできます。

Lambda AWS デプロイの AppSpec ファイルの例

Lambda 関数のバージョンをデプロイするために YAML で書かれた AppSpec ファイルの例を示します。

```
version: 0.0
Resources:
  - myLambdaFunction:
      Type: AWS::Lambda::Function
      Properties:
        Name: "myLambdaFunction"
        Alias: "myLambdaFunctionAlias"
        CurrentVersion: "1"
        TargetVersion: "2"
Hooks:
  - BeforeAllowTraffic: "LambdaFunctionToValidateBeforeTrafficShift"
  - AfterAllowTraffic: "LambdaFunctionToValidateAfterTrafficShift"
```

JSON で書かれた前述の例のバージョンを示します。

```
{
  "version": 0.0,
  "Resources": [{
   "myLambdaFunction": {
    "Type": "AWS::Lambda::Function",
    "Properties": {
     "Name": "myLambdaFunction",
     "Alias": "myLambdaFunctionAlias",
     "CurrentVersion": "1",
     "TargetVersion": "2"
    }
   }
  }],
  "Hooks": [{
    "BeforeAllowTraffic": "LambdaFunctionToValidateBeforeTrafficShift"
      },
      {
    "AfterAllowTraffic": "LambdaFunctionToValidateAfterTrafficShift"
   }
  ]
 }
```

デプロイ中のイベントのシーケンスを次に示します。

1. myLambdaFunction という名前の Lambda 関数のバージョン 1 からバージョン 2 にトラフィックを移行する前に、デプロイでトラフィックの移行を開始する準備が整っていることを確認する、LambdaFunctionToValidateBeforeTrafficShift という名前の Lambda 関数を実行します。
2. LambdaFunctionToValidateBeforeTrafficShift が終了コード 0 (成功) を返した場合は、myLambdaFunction のバージョン 2 へのトラフィックの移行を開始します。このデプロイのデプロイ設定により、トラフィックが移行するレートが決まります。
3. myLambdaFunction という名前の Lambda 関数のバージョン 1 からバージョン 2 へのトラフィックの移行が完了したら、デプロイが正常に完了したことを確認する、LambdaFunctionToValidateAfterTrafficShift という名前の Lambda 関数を実行します。

EC2/オンプレミスデプロイの AppSpec ファイルの例

以下は、Amazon Linux、Ubuntu Server、または RHEL インスタンスへのインプレイスデプロイのための AppSpec ファイルの例です。

> **Note**
>
> Windows Server インスタンスへのデプロイメントでは、runas 要素をサポートしていません。Windows Server インスタンスにデプロイする場合は、AppSpec ファイルに含まれないでください。

```
version: 0.0
os: linux
files:
  - source: Config/config.txt
    destination: /webapps/Config
  - source: source
    destination: /webapps/myApp
hooks:
  BeforeInstall:
    - location: Scripts/UnzipResourceBundle.sh
    - location: Scripts/UnzipDataBundle.sh
  AfterInstall:
    - location: Scripts/RunResourceTests.sh
      timeout: 180
  ApplicationStart:
    - location: Scripts/RunFunctionalTests.sh
      timeout: 3600
  ValidateService:
    - location: Scripts/MonitorService.sh
      timeout: 3600
      runas: codedeployuser
```

Windows Server のインスタンスで、[os: linux] を [os: windows] に変更します。また、destination パスを完全に修飾する必要があります (例: c:\temp\webapps\Config、c:\temp\webapps\myApp)。runas エレメントは含めないでください。

デプロイ中のイベントのシーケンスを次に示します。

1. Scripts/UnzipResourceBundle.sh にあるスクリプトを実行します。

2. 前のスクリプトが終了コード 0 (成功) を返した場合、Scripts/UnzipDataBundle.sh にあるスクリプトを実行します。
3. ファイルを Config/config.txt のパスからパス /webapps/Config/config.txt にコピーします。
4. source ディレクトリのすべてのファイルを再帰的に /webapps/myApp ディレクトリにコピーします。
5. Scripts/RunResourceTests.sh にあるスクリプトを、180 秒 (3 分) のタイムアウトで実行します。
6. Scripts/RunFunctionalTests.sh にあるスクリプトを、3600 秒 (1 時間) のタイムアウトで実行します。
7. Scripts/MonitorService.sh にあるスクリプトを、ユーザー codedeploy として 3600 秒 (1 時間) のタイムアウトで実行します。

AppSpec ファイルの間隔

AppSpec ファイルの間隔の正しい形式を以下に示します。角括弧に含まれた番号は、項目の間に必要なスペースの数を示します。たとえば、[4] は、項目の間に 4 つのスペースを挿入することを意味します。CodeDeploy は、AppSpec ファイルの場所とスペースの数が正しくない場合、デバッグが困難なエラーを発生させることがあります。

```
version:[1]version-number
os:[1]operating-system-name
files:
[2]-[1]source:[1]source-files-location
[4]destination:[1]destination-files-location
permissions:
[2]-[1]object:[1]object-specification
[4]pattern:[1]pattern-specification
[4]except:[1]exception-specification
[4]owner:[1]owner-account-name
[4]group:[1]group-name
[4]mode:[1]mode-specification
[4]acls:
[6]-[1]acls-specification
[4]context:
[6]user:[1]user-specification
[6]type:[1]type-specification
[6]range:[1]range-specification
```

```
[4]type:
[6]-[1]object-type
hooks:
[2]deployment-lifecycle-event-name:
[4]-[1]location:[1]script-location
[6]timeout:[1]timeout-in-seconds
[6]runas:[1]user-name
```

正しい間隔が設定された AppSpec ファイルの例を次に示します。

```
version: 0.0
os: linux
files:
  - source: /
    destination: /var/www/html/WordPress
hooks:
  BeforeInstall:
    - location: scripts/install_dependencies.sh
      timeout: 300
      runas: root
  AfterInstall:
    - location: scripts/change_permissions.sh
      timeout: 300
      runas: root
  ApplicationStart:
    - location: scripts/start_server.sh
    - location: scripts/create_test_db.sh
      timeout: 300
      runas: root
  ApplicationStop:
    - location: scripts/stop_server.sh
      timeout: 300
      runas: root
```

間隔の詳細については、YAML の仕様を参照してください。

AppSpec ファイルの検証とファイルの場所

ファイル構文

AWSが提供する AppSpec Assistant スクリプトを使用して、AppSpec ファイルの内容を検証できます。GitHub 上に、AppSpec ファイルテンプレートとスクリプトを見つけることができます。

代わりに、YAML Lint や Online YAML Parser などのブラウザベースのツールを使用して YAML 構文をチェックすることもできます。

ファイルの場所

AppSpec ファイルをアプリケーションのソースコンテンツのディレクトリ構造のルートディレクトリに配置したことを確認するには、次のいずれかのコマンドを実行します。

ローカルの Linux、macOS、Unix インスタンスでは、以下を実行します。

```
ls path/to/root/directory/appspec.yml
```

AppSpec ファイルがその場所にない場合、「No such file or directory」エラーが表示されます。

ローカル Windows インスタンスの場合

```
dir path\to\root\directory\appspec.yml
```

AppSpec ファイルがその場所にない場合、「File Not Found」エラーが表示されます。

CodeDeploy エージェント設定リファレンス

CodeDeploy エージェントがインストールされている場合、設定ファイルはインスタンスに配置されます。この設定ファイルは、インスタンスとやり取りするときに使用する CodeDeploy のディレクトリパスおよびその他の設定を指定します。ファイルの一部の設定オプションは変更できます。

Amazon Linux、Ubuntu Server、Red Hat Enterprise Linux (RHEL) インスタンスの場合、設定ファイルの名前は、`codedeployagent.yml` です。ファイルは、`/etc/codedeploy-agent/conf` ディレクトリに配置されます。

Windows Server インスタンスの場合、設定ファイルは `conf.yml` という名前になります。ファイルは、`C:\ProgramData\Amazon\CodeDeploy` ディレクトリに配置されます。

設定には以下が含まれます。

:log_aws_wire:	`true` CodeDeploy エージェントが Amazon S3 からワイヤログをキャプチャし、:log_dir: 設定で指定された場所のログファイルに書き込むには、 に設定します。

バージョン 2.0.x 以降では、ファイル名は `codedeploy-agent.aws_wire.log` (0640 アクセス許可で作成) です。

バージョン 1.8.x 以前では、ファイル名は ですcodedeploy-agent.wire.log 。

> **Warning**
>
> ワイヤログの取得に必要な時間のみ、:log_aws_wire: を `true` に設定する必要があります。ワイヤログファイルは、非常に大きなサイズにすばやく拡張できます。ワイヤログ出力には、この設定が に設定されている間にAmazon S3 との間で転送されたファイルのプレーンテキストの内容など、機密情報が含まれている場合がありますtrue。ワイヤログには、CodeDeploy デプロイに関連するアクティビティだけでなく、この設定が AWS に設定されている間に `true` アカウントに関連付けられていたすべての Amazon S3 アクティビティに関する情報が含まれます。

デフォルトの設定は `false` です。

この設定は、すべてのインスタンスタイプに適用されます。この設定を使用できるようにするには、この設定を Windows サーバーインスタンスに追加する必要があります。

:log_dir:	CodeDeploy エージェントオペレーションに関連するログファイルが保存されるインスタンスのフォルダ。 デフォルトの設定は、Amazon Linux、Ubuntu Server、RHEL インスタンス用の `'/var/log/aws/codedeploy-agent'` 、および Windows Server インスタンス用の `C:\ProgramData\Amazon\CodeDeploy\log` です。
:pid_dir:	`codedeploy-agent.pid` が保存されているフォルダ。 このファイルには、CodeDeploy エージェントのプロセス ID (PID) が含まれます。デフォルトの設定は `'/opt/codedeploy-agent/state/.pid'` です。 この設定は、Amazon Linux、Ubuntu Server、RHEL インスタンスにのみ適用されます。
:program_name:	CodeDeploy エージェントのプログラム名。 デフォルトの設定は `codedeploy-agent` です。 この設定は、Amazon Linux、Ubuntu Server、RHEL インスタンスにのみ適用されます。

:root_dir:	関連するリビジョン、デプロイ履歴、およびインスタンスのデプロイスクリプトが保存されるフォルダ。 デフォルトの設定は、Amazon Linux、Ubuntu Server、RHEL インスタンス用の `/opt/codedeploy-agent/deployment-root` 、および Windows Server インスタンス用の `C:\ProgramData\Amazon\CodeDeploy` です。
:verbose:	CodeDeploy エージェントでインスタンスのデバッグメッセージログファイルを表示する場合は、`true` に設定します。 デフォルトの設定は `false` です。
:wait_between_runs:	保留中のデプロイに対する CodeDeploy エージェントによる CodeDeploy のポーリング間隔 (秒単位)。 バージョン 2.0.x 以降では、デフォルト設定は `30` です。 バージョン 1.8.x 以前では、デフォルト設定は `1` です。

:on_premises_config_file:	オンプレミスインスタンスの場合、`codedeploy.onpremises.yml` (Ubuntu Server および RHEL の場合)、または `conf.onpremises.yml` (Windows Server の場合) という名前の設定ファイルの別の場所へのパスです。 デフォルトでは、これらのファイルは Ubuntu Server および RHEL の場合は `/etc/codedeploy-agent/conf/codedeploy.onpremises.yml` 、 Windows Server の場合は `C:\ProgramData\Amazon\CodeDeploy conf.onpremises.yml` に保存されます。 バージョン 1.0.1.686 以降のバージョンの CodeDeploy エージェントで利用できます。
:proxy_uri:	(オプション) CodeDeploy エージェントが CodeDeploy オペレーション AWS のために に接続する HTTP CodeDeploy プロキシ。`https://user:password@my.proxy:443/path?query` のような形式を使用します。 バージョン 1.0.1.824 以降のバージョンの CodeDeploy エージェントで利用できます。

:max_revisions:	(オプション) CodeDeploy エージェントでアーカイブするデプロイグループ用のアプリケーションリビジョンの数。指定された数を超えるリビジョンは削除されます。 正の整数を入力します。値を指定しない場合、CodeDeploy は現在デプロイされたリビジョンに加えて 5 つの最新のリビジョンを保持します。 バージョン 1.0.1.966 以降のバージョンの CodeDeploy エージェント でサポートされます。
:enable_auth_policy:	(オプション) IAM 認可 を使用して、アクセスコントロールを設定し、CodeDeploy エージェントが使用している IAM ロールまたはユーザーのアクセス許可を制限する場合に `true` に設定します。Amazon Virtual Private Cloud で CodeDeploy を使用 にするには、この値は、`true` である必要があります。 デフォルトの設定は `false` です。
:disable_imds_v1:	この設定は、CodeDeploy エージェント 1.7.0 以降で使用できます。 IMDSv2 エラーが発生したときに IMDSv1 へのフォールバックを無効にするには、`true` に設定します。デフォルトは `false` です (フォールバックを有効にします)。
:disable_core_dumps:	バージョン 2.0.x 以降用。 コアダンプ`true`を抑制するには、 に設定します。クラッシュ調査`false`のために を に設定します。 デフォルトの設定は `true` です。

:enable_command_port:	バージョン 2.0.x 以降用。 エージェントのステータスをクエリしtrue、コマンドをローカルに挿入するためのループバック専用 (127.0.0.1) ローカル管理インターフェイスを有効にするには、 に設定します。 デフォルトの設定は false です。
:enable_archive_reuse:	バージョン 2.1.0 以降用。 リビジョンがすでにインスタンスにあるRESTARTデプロイでも、アプリケーションリビジョンを常にダウンロードfalseするには、に設定します。この設定が の場合true、エージェントはインスタンスにすでにあるコピーから をインストールし、そのコピーが欠落しているか不完全である場合はリビジョンをダウンロードします。 デフォルトの設定は true です。 この動作の詳細については、「」を参照してくださいCodeDeploy エージェントのバージョン履歴。
:reject_symlinks_in_bundle:	バージョン 2.0.x 以降用。 シンボリックリンクまたはハードリンクを含むバンドルを拒否trueするには、 に設定します。 デフォルトの設定は false です。

:reject_path_traversal_in_bundle:	バージョン 2.0.x 以降用。 に設定するとtrue、../パストラバーサルまたは絶対パスを含むアーカイブエントリと、バンドルルートをエスケープする AppSpec ソースフィールドが拒否されます。 デフォルトの設定は false です。
:reject_unsafe_permissions_in_bundle:	バージョン 2.0.x 以降用。 SUID/SGID ファイルまたは SUID/SGID を設定する AppSpec アクセス許可モードを含むバンドルを拒否trueするには、 に設定します。 デフォルトの設定は false です。
:reject_unconfined_selinux_in_bundle:	バージョン 2.0.x 以降用。 に設定するとtrue、SELinux タイプを MAC で無効にするファイルを含むバンドルが拒否されます。 デフォルトの設定は false です。
:reject_symlink_permission_targets:	バージョン 2.0.x 以降用。 に設定するtrueと、ターゲットがシンボリックリンクである AppSpec ファイルpermissions エントリが拒否されます。デフォルトでは、エージェントは所有権、モード、ACL、および SELinux の変更を適用するときにリンクに従います。 デフォルトの設定は false です。

:ignore_ownership_in_bundle:	バージョン 2.0.x 以降用。 アーカイブに記録されたユーザー ID tarとグループ ID を適用するroot代わりに、 とtgzバンドルから抽出されたファイルの所有権を にリセットtrueするには、 に設定します。zip バンドル内のファイルは、この設定rootに関係なく によって所有されます。 デフォルトの設定は false です。
:restrict_agent_dir_permissions:	バージョン 2.0.x 以降用。Linux インスタンスのみ。 に設定するとtrue、エージェントインストールルートのディレクトリとファイルが、デフォルトで使用されているワールド読み取りモードではなく、ルートのみのアクセスに制限されます。 デフォルトの設定は false です。
:restrict_log_dir_permissions:	バージョン 2.0.x 以降用。Linux インスタンスのみ。 true に設定すると、エージェントログとアップデーターログを含むエージェントログディレクトリが、デフォルトで使用されているワールド読み取りモードではなく、ルートのみのアクセスに制限されます。 デフォルトの設定は false です。

:restrict_hook_env_to_allowlist:	バージョン 2.0.x 以降用。 基本シェル変数と CodeDeploy デプロイ変数のみを含む最小限の環境でライフサイクルイベントフックを実行するtrueには、 に設定します。デフォルトでは、フックはエージェントプロセスの完全な環境を継承します。 デフォルトの設定は false です。
:strip_loader_env_in_hooks:	バージョン 2.0.x 以降用。Linux インスタンス、およびバージョン 2.1.2 以降の macOS インスタンス。 ライフサイクルイベントフックの環境から LD_PRELOAD 、LD_LIBRARY_PATH 、および LD_AUDIT変数trueを削除するには、 に設定します。macOS では、 設定は代わりに DYLD_INSERT_LIBRARIES 、、DYLD_LIBRARY_PATH 、DYLD_FRAMEWORK_PATH 、、DYLD_VERSIONED_LIBRARY_PATH 、および DYLD_FALLBACK_LIBRARY_PATH DYLD_FALLBACK_FRAMEWORK_PATH DYLD_VERSIONED_FRAMEWORK_PATH 変数を削除します。 デフォルトの設定は false です。

:disable_powershell_profile_in_hooks:	バージョン 2.0.x 以降用。Windows Server インスタンスのみ。 を に設定trueして、 -NoProfile およ び -NonInteractive オプションを使用して.ps1ライフサイクルイベントフックを実行します。デフォルトでは、これらのフックはエージェントが実行するユーザーの PowerShell プロファイルをロードします。 デフォルトの設定は false です。
:archive_max_extraction_size:	バージョン 2.0.x 以降用。 宣言された非圧縮バンドルサイズの上限を設定します。抽出されたコンテンツがこの値を超えると、デプロイは失敗します。 デフォルトは上限なし (無制限) です。
:wait_after_error:	エラーが発生してから通常のオペレーションを再開するまでにエージェントが待機する秒数。 デフォルトの設定は 30 です。
:http_read_timeout:	サービス API コールの秒単位の HTTP 読み取りタイムアウト。 デフォルトの設定は 80 です。
:kill_agent_max_wait_time_seconds:	強制終了するまでエージェントが正常なシャットダウンを待機する最大秒数。バージョン 2.0.x 以降では、この値は 24 時間 (86400 秒) に制限されます。 デフォルト設定は 7200 (2 時間) です。

:enable_deployments_log:	を に設定trueして、集約されたデプロイログ () を書き込みますcodedeploy-agent-deployments.log 。 デフォルトの設定は true です。
:use_fips_mode:	FIPS 準拠のエンドポイントtrueを使用するには、 に設定します。米国リージョンにのみ適用されます。 デフォルトの設定は false です。
:deploy_control_endpoint:	(オプション) CodeDeploy サービスのカスタムコントロールプレーンエンドポイント URL。 デフォルトは上書きされません (エージェントはリージョンからエンドポイントを解決します)。
:s3_endpoint_override:	(オプション) バンドルダウンロード用のカスタム Amazon S3 エンドポイント URL。 デフォルトは上書きされません (エージェントはリージョンから Amazon S3 エンドポイントを解決します)。

:s3_download_max_concurrency:	バージョン 2.1.2 以降用。 エージェントが Amazon S3 からアプリケーションリビジョンをダウンロードするために使用する並列 HTTP 範囲リクエストの数。を より大きい値に設定1して、大きなリビジョンを並列範囲でダウンロードします。では1、エージェントは 1 回のGetObject リクエストでリビジョンをダウンロードします。 上記の値は として64扱われ64、 として扱われ0ます1。 デフォルトの設定は 1 です。
:s3_download_chunk_size:	バージョン 2.1.2 以降用。 :s3_download_max_concurrency: が より大きい場合の各範囲リクエストのサイズ1。バイト数、または "16MiB"や などのサイズ文字列を入力します"16MB"。このサイズ以下のリビジョンは、1 回のリクエストでダウンロードされます。 64 KiB 未満の値は 64 KiB として扱われます。 デフォルトの設定は "8MiB" です。
:ongoing_deployment_tracking:	デプロイの追跡状態に使用されるサブディレクトリの名前。 デフォルトの設定は ongoing-deployment です。

関連トピック

CodeDeploy エージェントの使用

CodeDeploy エージェントのオペレーションの管理

CloudFormation CodeDeploy リファレンスの テンプレート

このセクションでは、CodeDeploy デプロイで動作するように設計された CloudFormation リソース、変換、フックについて説明します。CodeDeploy の CloudFormation フックによって管理されるスタック更新を作成する手順については、「」を参照してください。 を使用して Amazon ECS ブルー/グリーンデプロイを作成する CloudFormation

> **Note**
>
> CloudFormation フックは の CloudFormation コンポーネントの一部 AWS であり、CodeDeploy ライフサイクルイベントフックとは異なります。

CodeDeploy で利用できる他の方法に加えて、 CloudFormation テンプレートを使用して以下のタスクを実行できます。

- アプリケーションを作成します。
- デプロイグループを作成し、ターゲットリビジョンを指定します。
- デプロイ設定を作成します。
- Amazon EC2 インスタンスを作成します。

CloudFormation は、 テンプレートを使用して AWS リソースをモデル化およびセットアップするのに役立つサービスです。 CloudFormation テンプレートは、形式が JSON 標準に準拠しているテキストファイルです。必要なすべての AWS リソースを記述するテンプレートを作成し、CloudFormation がそれらのリソースのプロビジョニングと設定を行います。

詳細については、AWS CloudFormationユーザーガイドの「AWS CloudFormation とは」および「Working with AWS CloudFormation Templates」を参照してください。

組織内で CodeDeploy と互換性のある CloudFormation テンプレートを使用する場合は、管理者として、 CloudFormation が CloudFormation 依存する AWS サービスとアクションへのアクセスとアクセスを許可する必要があります。アプリケーション、デプロイグループ、デプロイ設定を作成するアクセス許可を付与するには、作業するユーザーのアクセス許可セットに次のポリシーを追加します CloudFormation。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudformation:*"
      ],
      "Resource": "*"
    }
  ]
}
```

ポリシーの詳細については、以下のトピックを参照してください。

- Amazon EC2 インスタンスを作成する許可セットのユーザーに追加する必要があるポリシーを表示するには、「CodeDeploy 用の Amazon EC2 インスタンスを作成する (CloudFormation テンプレート)」を参照してください。
- 許可セットにポリシーを追加する方法については、「IAM ユーザーガイド」の「アクセス権限セットを作成します。」を参照してください。
- CodeDeploy のアクションとリソースの限定されたセットにユーザーを制限する方法については、「AWS CodeDeploy の マネージド (事前定義) ポリシー」を参照してください。

次の表は、 CloudFormation テンプレートがユーザーに代わって実行できるアクションと、CloudFormation テンプレートに追加できる AWS リソースタイプとそのプロパティタイプに関する詳細情報へのリンクを示しています。

アクション	CloudFormation リファレンス	参照タイプ
CodeDeploy のアプリケーションを作成します。	AWS::CodeDeploy::Application	CloudFormation リソース
アプリケーションリビジョンのデプロイに使用されるデ	AWS::CodeDeploy::DeploymentGroup	CloudFormation リソース

アクション	CloudFormation リファレンス	参照タイプ
プロイグループの詳細を作成し、指定します。1		
CodeDeploy がデプロイ中に使用する一連のデプロイのルール、デプロイの成功条件、デプロイの失敗条件を作成します。	AWS::CodeDeploy::DeploymentConfig	CloudFormation リソース
Amazon EC2 インスタンスを作成します。2	AWS::EC2::Instance	CloudFormation リソース
変換とAWS::CodeDeploy::BlueGreen フックを使用して、CodeDeploy CloudFormation AWS::CodeDeployBlueGreen ブルー/グリーンデプロイのスタック更新の管理、リソースの作成、トラフィックのシフトを行います。[3]	AWS::CodeDeployBlueGreen	AWS::CodeDeployBlueGreen 変換は、 CloudFormation によりホストされるマクロです。
	AWS::CodeDeploy::BlueGreen	AWS::CodeDeploy::BlueGreen フックはのHookリソースとして構造化されています CloudFormation。指定された CodeDeploy ライフサイクルイベントフックを指すことによって、フックには、CodeDeploy AppSpec ファイルの代わりにパラメータが含まれます。

アクション	CloudFormation リファレンス	参照タイプ
[1] デプロイグループの一部としてデプロイするアプリケーションリビジョンのバージョンを指定する場合、プロビジョニングプロセスが完了するとすぐに、ターゲットリビジョンがデプロイされます。テンプレート設定の詳細については、AWS CloudFormation ユーザーガイド の CodeDeploy DeploymentGroup デプロイメントリビジョン S3Location と CodeDeploy デプロイ DeploymentGroup デプロイメントリビジョン GitHubLocation を参照してください。 2 CodeDeploy がサポートされているリージョンに Amazon EC2 インスタンスを作成するために使用できるテンプレートが用意されています。これらのテンプレートの使用の詳細については、「CodeDeploy 用の Amazon EC2 インスタンスを作成する (CloudFormation テンプレート)」を参照してください。 [3]このデプロイ設定では、 Amazon ECS ブルー/グリーンデプロイのみがサポートされています。 CloudFormationによる Amazon ECS ブルー/グリーンデプロイのデプロイ構成の詳細については、「CloudFormation Blue/Green デプロイのデプロイ設定 (Amazon ECS)」を参照してください。による Amazon ECS ブルー/グリーンデプロイ、 CloudFormation および CodeDeploy でデプロイを表示する方法の詳細については、「」を参照してくださいを使用して Amazon ECS ブルー/グリーンデプロイを作成する CloudFormation。		

Amazon Virtual Private Cloud で CodeDeploy を使用

Amazon Virtual Private Cloud (Amazon VPC) を使用してAWSリソースをホストする場合は、VPC と CodeDeploy の間にプライベート接続を確立できます。この接続を使用すると、CodeDeploy はパブリックインターネットを経由せずに、VPC のリソースと通信できます。

Amazon VPC は、定義した仮想ネットワークでAWSリソースを起動するために使用できるAWSサービスです。VPC を使用することで、IP アドレス範囲、サブネット、ルートテーブル、ネットワークゲートウェイなどのネットワーク設定を制御できます。VPC エンドポイントでは、VPC とAWSサー

ビス間のルーティングはAWSネットワークによって処理され、IAM ポリシーを使用してサービスリソースへのアクセスを制御できます。

VPC を CodeDeploy に接続するには、CodeDeploy の インターフェイス VPC エンドポイント を定義します。インターフェイスエンドポイントは、サポートされているAWSサービス宛てのトラフィックのエントリポイントとして機能するプライベート IP アドレスを持つ Elastic Network Interface です。このエンドポイントは、インターネットゲートウェイ、ネットワークアドレス変換 (NAT) インスタンス、および VPN 接続を必要とせず、信頼性が高くスケーラブルな CodeDeploy への接続を提供します。詳細については、「Amazon VPC ユーザーガイド」の「Amazon VPC とは」を参照してください。

インターフェイス VPC エンドポイントはAWS PrivateLink、プライベート IP アドレスを持つ Elastic Network Interface を使用してAWSサービス間のプライベート通信を可能にするAWSテクノロジーを利用しています。詳細については、「AWS PrivateLink」を参照してください。

以下の手順は、Amazon VPC のユーザー向けです。詳細については、『Amazon VPC ユーザーガイド』の「開始方法」を参照してください。

可用性

CodeDeploy には 2 つの VPC エンドポイントがあります。1 つは CodeDeploy エージェントオペレーション用で、もう 1 つは CodeDeploy API オペレーション用です。次の表は、各エンドポイントでサポートされているAWSリージョンを示しています。

FIPS エンドポイントの詳細については、「AWS CodeDeployエンドポイントとクォータ」を参照してください。

リージョン名	リージョンコード	エージェントのエンドポイント	API エンドポイント	FIPS 準拠リージョン
米国東部 (バージニア北部)	us-east-1	はい	はい	あり
米国東部 (オハイオ)	us-east-2	あり	はい	あり
米国西部 (北カリフォルニア)	us-west-1	あり	はい	あり

リージョン名	リージョンコード	エージェントのエンドポイント	API エンドポイント	FIPS 準拠リージョン
米国西部 (オレゴン)	us-west-2	あり	はい	あり
アフリカ (ケープタウン)	af-south-1	はい	はい	
アジアパシフィック (香港)	ap-east-1	はい	はい	
アジアパシフィック (ハイデラバード)	ap-south-2	はい	はい	
アジアパシフィック (ジャカルタ)	ap-southeast-3	はい	はい	
アジアパシフィック (メルボルン)	ap-southeast-4	はい	はい	
アジアパシフィック (マレーシア)	ap-southeast-5	はい	はい	
アジアパシフィック (ニュージーランド)	ap-southeast-6	はい	はい	
アジアパシフィック (タイ)	ap-southeast-7	はい	はい	
アジアパシフィック (台北)	ap-east-2	はい	はい	

リージョン名	リージョンコード	エージェントのエンドポイント	API エンドポイント	FIPS 準拠リージョン
アジアパシフィック (ムンバイ)	ap-south-1	あり	あり	
アジアパシフィック (大阪)	ap-northeast-3	はい	はい	
アジアパシフィック (ソウル)	ap-northeast-2	あり	はい	
アジアパシフィック (シンガポール)	ap-southeast-1	はい	はい	
アジアパシフィック (シドニー)	ap-southeast-2	はい	はい	
アジアパシフィック (東京)	ap-northeast-1	あり	あり	
カナダ (中部)	ca-central-1	あり	はい	
中国 (北京)	cn-north-1	はい	いいえ	
中国 (寧夏)	cn-northwest-1	いいえ	いいえ	
欧州 (フランクフルト)	eu-central-1	あり	あり	
欧州 (アイルランド)	eu-west-1	あり	はい	
欧州 (ロンドン)	eu-west-2	はい	はい	
欧州 (ミラノ)	eu-south-1	はい	はい	

リージョン名	リージョンコード	エージェントのエンドポイント	API エンドポイント	FIPS 準拠リージョン
欧州 (パリ)	eu-west-3	はい	はい	
欧州 (スペイン)	eu-south-2	はい	はい	
欧州 (ストックホルム)	eu-north-1	はい	はい	
欧州 (チューリッヒ)	eu-central-2	はい	はい	
イスラエル (テルアビブ)	il-central-1	はい	はい	
中東 (バーレーン)	me-south-1	はい	はい	
中東 (UAE)	me-central-1	はい	はい	
メキシコ (中部)	mx-central-1	はい	はい	
南米 (サンパウロ)	sa-east-1	はい	はい	
AWS GovCloud (US-East)	us-gov-east-1	いいえ	なし	はい
AWS GovCloud (US-West)	us-gov-west-1	いいえ	なし	はい

CodeDeploy 用の VPC エンドポイントを作成する

VPC で CodeDeploy の使用を開始するには、CodeDeploy のインターフェイス VPC エンドポイントを作成します。CodeDeploy には、エージェントの Git オペレーションと CodeDeploy API オペレーション用に別々のエンドポイントが必要です。ビジネスニーズに応じて、複数の VPC エンドポイントを作成する必要がある場合があります。CodeDeploy 用の VPC エンドポイントを作成するときは、[AWSサービス] を選択し、[サービス名] で、次のオプションから選択します。

- com.amazonaws.***region***.codecommit: CodeDeploy API オペレーション用の VPC エンドポイントを作成する場合は、このオプションを選択します。たとえば、ユーザーが AWS CLI、CodeDeploy API、またはAWS SDKs を使用して CodeDeploy とやり取りし`CreateApplication`、、、 などのオペレーションを行う場合`GetDeployment`は、このオプションを選択します`ListDeploymentGroups`。
- com.amazonaws.***region***.codedeploy-fips: FIPS エンドポイントの CodeDeploy API オペレーション用に VPC エンドポイントを作成する場合は、このオプションを選択します。
- com.amazonaws.***region***.codecommit: CodeDeploy エージェント オペレーション用の VPC エンドポイントを作成する場合は、このオプションを選択します。また、`:enable_auth_policy:` に `true` エージェント設定ファイルで、必要な権限をアタッチする必要があります 詳細については、「CodeDeploy エージェントと IAM 許可を設定する」を参照してください。

Lambda または ECS デプロイを使用している場合は、用の VPC エンドポイントを作成する必要があります。com.amazonaws.***region***.codedeploy Amazon EC2 デプロイを使用しているお客様については、com.amazonaws.***region***.codedeploy と com.amazonaws.***region***.codedeploy-commands-secure はどちらも VPC エンドポイントが必要です

CodeDeploy エージェントと IAM 許可を設定する

CodeDeploy で Amazon VPC エンドポイントを使用するには、`:enable_auth_policy:` に `true` EC2 またはオンプレミスインスタンスにあるエージェント設定ファイルに次の値を設定する必要があります。設定ファイルの形式の詳細については、「CodeDeploy エージェント設定リファレンス」を参照してください。

Amazon EC2 インスタンスプロファイル (Amazon EC2 インスタンスを使用している場合) または IAM ユーザーまたはロール (オンプレミスインスタンスを使用している場合) に次の IAM アクセス許可を追加する必要があります。

```
{
  "Statement": [
    {
      "Action": [
        "codedeploy-commands-secure:GetDeploymentSpecification",
        "codedeploy-commands-secure:PollHostCommand",
        "codedeploy-commands-secure:PutHostCommandAcknowledgement",
        "codedeploy-commands-secure:PutHostCommandComplete"
      ],
      "Effect": "Allow",
      "Resource": "*"
```

```
    }
  ]
}
```

詳細については、『Amazon VPC ユーザーガイド』の「インターフェイスエンドポイントの作成」を参照してください。

CodeDeploy リソースキットリファレンス

CodeDeploy が依存するファイルの多くは、公開されている AWS リージョン固有の Amazon S3 バケットに保存されます。このようなファイルには、CodeDeploy エージェントのインストールファイル、テンプレート、サンプルアプリケーションファイルがあります。この一連のファイルを CodeDeploy リソースキットと呼びます。

トピック

- リージョン別リソースキットバケット名
- リソースキットの内容
- リソースキットのファイルのリストの表示
- リソースキットのファイルのダウンロード

リージョン別リソースキットバケット名

この表は、本ガイドの一部の手順で必要な *bucket-name* を置換する名前の一覧です。これらは、CodeDeploy リソースキットファイルが含まれている Amazon S3 バケットの名前です。

> **Note**
>
> アジアパシフィック (香港) リージョンの Amazon S3 バケットにアクセスするには、 AWS アカウントでリージョンを有効にする必要があります。詳細については、AWS 「リージョンの管理」を参照してください。

リージョン名	***bucket-name*** 置換	リージョン識別子
米国東部 (バージニア北部)	aws-codedeploy-us-east-1	us-east-1
米国東部 (オハイオ)	aws-codedeploy-us-east-2	us-east-2

リージョン名	*bucket-name* 置換	リージョン識別子
米国西部 (北カリフォルニア)	aws-codedeploy-us-west-1	us-west-1
米国西部 (オレゴン)	aws-codedeploy-us-west-2	us-west-2
アフリカ (ケープタウン)	aws-codedeploy-af-south-1	af-south-1
アジアパシフィック (香港)	aws-codedeploy-ap-east-1	ap-east-1
アジアパシフィック (ハイデラバード)	aws-codedeploy-ap-south-2	ap-south-2
アジアパシフィック (ジャカルタ)	aws-codedeploy-ap-southeast-3	ap-southeast-3
アジアパシフィック (メルボルン)	aws-codedeploy-ap-southeast-4	ap-southeast-4
アジアパシフィック (マレーシア)	aws-codedeploy-ap-southeast-5	ap-southeast-5
アジアパシフィック (ニュージーランド)	aws-codedeploy-ap-southeast-6	ap-southeast-6
アジアパシフィック (タイ)	aws-codedeploy-ap-southeast-7	ap-southeast-7
アジアパシフィック (台北)	aws-codedeploy-ap-east-2	ap-east-2
アジアパシフィック (ムンバイ)	aws-codedeploy-ap-south-1	ap-south-1
アジアパシフィック (大阪)	aws-codedeploy-ap-northeast-3	ap-northeast-3
アジアパシフィック (ソウル)	aws-codedeploy-ap-northeast-2	ap-northeast-2

リージョン名	*bucket-name* 置換	リージョン識別子
アジアパシフィック (シンガポール)	aws-codedeploy-ap-southeast-1	ap-southeast-1
アジアパシフィック (シドニー)	aws-codedeploy-ap-southeast-2	ap-southeast-2
アジアパシフィック (東京)	aws-codedeploy-ap-northeast-1	ap-northeast-1
カナダ (中部)	aws-codedeploy-ca-central-1	ca-central-1
欧州 (フランクフルト)	aws-codedeploy-eu-central-1	eu-central-1
欧州 (アイルランド)	aws-codedeploy-eu-west-1	eu-west-1
欧州 (ロンドン)	aws-codedeploy-eu-west-2	eu-west-2
欧州 (ミラノ)	aws-codedeploy-eu-south-1	eu-south-1
欧州 (パリ)	aws-codedeploy-eu-west-3	eu-west-3
欧州 (スペイン)	aws-codedeploy-eu-south-2	eu-south-2
欧州 (ストックホルム)	aws-codedeploy-eu-north-1	eu-north-1
欧州 (チューリッヒ)	aws-codedeploy-eu-central-2	eu-central-2
イスラエル (テルアビブ)	aws-codedeploy-il-central-1	il-central-1
中東 (バーレーン)	aws-codedeploy-me-south-1	me-south-1
中東 (UAE)	aws-codedeploy-me-central-1	me-central-1
メキシコ (中部)	aws-codedeploy-mx-central-1	mx-central-1
南米 (サンパウロ)	aws-codedeploy-sa-east-1	sa-east-1
AWS GovCloud (US-East)	aws-codedeploy-us-gov-east-1	us-gov-east-1

リージョン名	*bucket-name* 置換	リージョン識別子
AWS GovCloud (US-West)	aws-codedeploy-us-gov-west-1	us-gov-west-1

リソースキットの内容

次の表に、CodeDeploy リソースキットのファイルを一覧表示します。

システム	説明
`LATEST_VERSION`	Amazon EC2 Systems Manager などの更新メカニズムによって使用され、CodeDeploy エージェントの最新バージョンを特定するファイルです。
`VERSION`	自動更新メカニズムが CodeDeploy エージェントバージョン 1.1.0 で削除され、このファイルは使用されなくなりました。インスタンスで実行されているときに CodeDeploy エージェントが自身の更新に使用するファイルです。
`codedeploy-agent.noarch.rpm`	Amazon Linux および Red Hat Enterprise Linux (RHEL) 用の CodeDeploy エージェントです。同じベースファイル名で、異なるバージョン (`-1.0-0` など) の複数のファイルが存在することがあります。
`codedeploy-agent_all.deb`	Ubuntu サーバー用の CodeDeploy エージェントです。同じベースファイル名で、異なるバージョン (`_1.0-0` など) の複数のファイルが存在することがあります。
`codedeploy-agent.msi`	Windows サーバー用の CodeDeploy エージェントです。同じベースファイル名で、異なるバージョン (`-1.0-0` など) の複数のファイルが存在することがあります。

システム	説明
`install`	CodeDeploy エージェントをより簡単にインストールするために使用できるファイルです。 `latestv2/` プレフィックスは、バージョン 2.0.x のインストールスクリプトを提供します。
`CodeDeploy_SampleCF_Template.json`	Amazon Linux または Windows Server を実行する 1 つから 3 つの Amazon EC2 インスタンスから起動するために使用できる AWS CloudFormation テンプレート。同じベースファイル名で、異なるバージョン (`-1.0.0` など) の複数のファイルが存在することがあります。
`CodeDeploy_SampleCF_ELB_Integration.json`	Apache Web Server で実行されている負荷分散されたサンプルウェブサイトを作成するために使用できる AWS CloudFormation テンプレート。アプリケーションは、作成したリージョン内のすべてのアベイラビリティーゾーンにまたがるように設定されています。このテンプレートは、3 つの Amazon EC2 インスタンスと IAM インスタンスプロファイルを作成し、インスタンスに Amazon S3、Amazon EC2 Auto Scaling CloudFormation、Elastic Load Balancing のリソースへのアクセスを許可します。また、ロードバランサーと CodeDeploy サービスロールも作成します。
`SampleApp_ELB_Integration.zip`	Elastic Load Balancing ロードバランサーに登録されている Amazon EC2 インスタンスにデプロイできるサンプルアプリケーションリビジョンです。

システム	説明
SampleApp_Linux.zip	Amazon Linux を実行している Amazon EC2 インスタンス、または Ubuntu サーバーまたは RHEL インスタンスにデプロイできるサンプルアプリケーションリビジョンです。同じベースファイル名で、異なるバージョン (-1.0 など) の複数のファイルが存在することがあります。
SampleApp_Windows.zip	Windows Server インスタンスにデプロイできるサンプルのアプリケーションリビジョンです。同じベースファイル名で、異なるバージョン (-1.0 など) の複数のファイルが存在することがあります。

リソースキットのファイルのリストの表示

ファイルのリストを表示するには、 リージョンの aws s3 ls コマンドを使用します。

> **Note**
>
> 各バケットのファイルは、対応するリージョンのリソースで動作するように設計されています。

- `aws s3 ls --recursive s3://aws-codedeploy-us-east-2 --region us-east-2`
- `aws s3 ls --recursive s3://aws-codedeploy-us-east-1 --region us-east-1`
- `aws s3 ls --recursive s3://aws-codedeploy-us-west-1 --region us-west-1`
- `aws s3 ls --recursive s3://aws-codedeploy-us-west-2 --region us-west-2`
- `aws s3 ls --recursive s3://aws-codedeploy-ca-central-1 --region ca-central-1`
- `aws s3 ls --recursive s3://aws-codedeploy-eu-west-1 --region eu-west-1`

- `aws s3 ls --recursive s3://aws-codedeploy-eu-west-2 --region eu-west-2`
- `aws s3 ls --recursive s3://aws-codedeploy-eu-west-3 --region eu-west-3`
- `aws s3 ls --recursive s3://aws-codedeploy-eu-central-1 --region eu-central-1`
- `aws s3 ls --recursive s3://aws-codedeploy-il-central-1 --region il-central-1`
- `aws s3 ls --recursive s3://aws-codedeploy-ap-east-1 --region ap-east-1`
- `aws s3 ls --recursive s3://aws-codedeploy-ap-northeast-1 --region ap-northeast-1`
- `aws s3 ls --recursive s3://aws-codedeploy-ap-northeast-2 --region ap-northeast-2`
- `aws s3 ls --recursive s3://aws-codedeploy-ap-southeast-1 --region ap-southeast-1`
- `aws s3 ls --recursive s3://aws-codedeploy-ap-southeast-2 --region ap-southeast-2`
- `aws s3 ls --recursive s3://aws-codedeploy-ap-southeast-4 --region ap-southeast-4`
- `aws s3 ls --recursive s3://aws-codedeploy-ap-southeast-5 --region ap-southeast-5`
- `aws s3 ls --recursive s3://aws-codedeploy-ap-southeast-6 --region ap-southeast-6`
- `aws s3 ls --recursive s3://aws-codedeploy-ap-southeast-7 --region ap-southeast-7`
- `aws s3 ls --recursive s3://aws-codedeploy-ap-east-2 --region ap-east-2`
- `aws s3 ls --recursive s3://aws-codedeploy-mx-central-1 --region mx-central-1`
- `aws s3 ls --recursive s3://aws-codedeploy-ap-south-1 --region ap-south-1`
- `aws s3 ls --recursive s3://aws-codedeploy-sa-east-1 --region sa-east-1`

リソースキットのファイルのダウンロード

ファイルをダウンロードするには、 リージョンの aws s3 cp コマンドを使用します。

> **Note**
>
> 最後にピリオド (.) を使用してください。このコマンドは、現在のディレクトリにファイルをダウンロードします。

たとえば、次のコマンドは、バケットの /samples/latest/ フォルダの 1 つから SampleApp_Linux.zip という名前の 1 つのファイルをダウンロードします。

- ```
  aws s3 cp s3://aws-codedeploy-us-east-2/samples/latest/SampleApp_Linux.zip . --region
   us-east-2
  ```
- ```
  aws s3 cp s3://aws-codedeploy-us-east-1/samples/latest/SampleApp_Linux.zip . --region
   us-east-1
  ```
- ```
  aws s3 cp s3://aws-codedeploy-us-west-1/samples/latest/SampleApp_Linux.zip . --region
   us-west-1
  ```
- ```
  aws s3 cp s3://aws-codedeploy-us-west-2/samples/latest/SampleApp_Linux.zip . --region
   us-west-2
  ```
- ```
  aws s3 cp s3://aws-codedeploy-ca-central-1/samples/latest/SampleApp_Linux.zip . --
  region ca-central-1
  ```
- ```
  aws s3 cp s3://aws-codedeploy-eu-west-1/samples/latest/SampleApp_Linux.zip . --region
   eu-west-1
  ```
- ```
  aws s3 cp s3://aws-codedeploy-eu-west-2/samples/latest/SampleApp_Linux.zip . --region
   eu-west-2
  ```
- ```
  aws s3 cp s3://aws-codedeploy-eu-west-3/samples/latest/SampleApp_Linux.zip . --region
   eu-west-3
  ```
- ```
  aws s3 cp s3://aws-codedeploy-eu-central-1/samples/latest/SampleApp_Linux.zip . --
  region eu-central-1
  ```
  ```

- ```
  aws s3 cp s3://aws-codedeploy-il-central-1/samples/latest/SampleApp_Linux.zip . --
  region il-central-1
  ```
- ```
  aws s3 cp s3://aws-codedeploy-ap-east-1/samples/latest/SampleApp_Linux.zip . --region
   ap-east-1
  ```
- ```
  aws s3 cp s3://aws-codedeploy-ap-northeast-1/samples/latest/SampleApp_Linux.zip . --
  region ap-northeast-1
  ```
- ```
  aws s3 cp s3://aws-codedeploy-ap-northeast-2/samples/latest/SampleApp_Linux.zip . --
  region ap-northeast-2
  ```
- ```
  aws s3 cp s3://aws-codedeploy-ap-southeast-1/samples/latest/SampleApp_Linux.zip . --
  region ap-southeast-1
  ```
- ```
  aws s3 cp s3://aws-codedeploy-ap-southeast-2/samples/latest/SampleApp_Linux.zip . --
  region ap-southeast-2
  ```
- ```
  aws s3 cp s3://aws-codedeploy-ap-southeast-4/samples/latest/SampleApp_Linux.zip . --
  region ap-southeast-4
  ```
- ```
  aws s3 cp s3://aws-codedeploy-ap-southeast-5/samples/latest/SampleApp_Linux.zip . --
  region ap-southeast-5
  ```
- ```
  aws s3 cp s3://aws-codedeploy-ap-southeast-6/samples/latest/SampleApp_Linux.zip . --
  region ap-southeast-6
  ```
- ```
  aws s3 cp s3://aws-codedeploy-ap-southeast-7/samples/latest/SampleApp_Linux.zip . --
  region ap-southeast-7
  ```
- ```
  aws s3 cp s3://aws-codedeploy-ap-east-2/samples/latest/SampleApp_Linux.zip . --region
   ap-east-2
  ```
- ```
  aws s3 cp s3://aws-codedeploy-mx-central-1/samples/latest/SampleApp_Linux.zip . --
  region mx-central-1
  ```
- ```
  aws s3 cp s3://aws-codedeploy-ap-south-1/samples/latest/SampleApp_Linux.zip . --
  region ap-south-1
  ```
  ```

- ```
  aws s3 cp s3://aws-codedeploy-sa-east-1/samples/latest/SampleApp_Linux.zip . --region
   sa-east-1
  ```

すべてのファイルをダウンロードするには、リージョンで次のいずれかのコマンドを使用します。

- ```
  aws s3 cp --recursive s3://aws-codedeploy-us-east-2 . --region us-east-2
  ```
- ```
  aws s3 cp --recursive s3://aws-codedeploy-us-east-1 . --region us-east-1
  ```
- ```
  aws s3 cp --recursive s3://aws-codedeploy-us-west-1 . --region us-west-1
  ```
- ```
  aws s3 cp --recursive s3://aws-codedeploy-us-west-2 . --region us-west-2
  ```
- ```
  aws s3 cp --recursive s3://aws-codedeploy-ca-central-1 . --region ca-central-1
  ```
- ```
  aws s3 cp --recursive s3://aws-codedeploy-eu-west-1 . --region eu-west-1
  ```
- ```
  aws s3 cp --recursive s3://aws-codedeploy-eu-west-2 . --region eu-west-2
  ```
- ```
  aws s3 cp --recursive s3://aws-codedeploy-eu-west-3 . --region eu-west-3
  ```
- ```
  aws s3 cp --recursive s3://aws-codedeploy-eu-central-1 . --region eu-central-1
  ```
- ```
  aws s3 cp --recursive s3://aws-codedeploy-il-central-1 . --region il-central-1
  ```
- ```
  aws s3 cp --recursive s3://aws-codedeploy-ap-east-1 . --region ap-east-1
  ```
- ```
  aws s3 cp --recursive s3://aws-codedeploy-ap-northeast-1 . --region ap-northeast-1
  ```
- ```
  aws s3 cp --recursive s3://aws-codedeploy-ap-northeast-2 . --region ap-northeast-2
  ```
- ```
  aws s3 cp --recursive s3://aws-codedeploy-ap-southeast-1 . --region ap-southeast-1
  ```
- ```
  aws s3 cp --recursive s3://aws-codedeploy-ap-southeast-2 . --region ap-southeast-2
  ```
- ```
  aws s3 cp --recursive s3://aws-codedeploy-ap-southeast-4 . --region ap-southeast-4
  ```
  ```

- `aws s3 cp --recursive s3://aws-codedeploy-ap-southeast-5 . --region ap-southeast-5`
- `aws s3 cp --recursive s3://aws-codedeploy-ap-southeast-6 . --region ap-southeast-6`
- `aws s3 cp --recursive s3://aws-codedeploy-ap-southeast-7 . --region ap-southeast-7`
- `aws s3 cp --recursive s3://aws-codedeploy-ap-east-2 . --region ap-east-2`
- `aws s3 cp --recursive s3://aws-codedeploy-mx-central-1 . --region mx-central-1`
- `aws s3 cp --recursive s3://aws-codedeploy-ap-south-1 . --region ap-south-1`
- `aws s3 cp --recursive s3://aws-codedeploy-sa-east-1 . --region sa-east-1`

CodeDeploy のクォータ

次の表では、CodeDeploy のクォータについて説明します。

> **Note**
>
> EC2/オンプレミスのインプレースデプロイ実行時間の制限はさまざまです。2023 年 6 月より前に作成されたカスタムデプロイ設定の場合、制限は 8 時間です。2023 年 6 月以降に作成されたカスタムデプロイ設定の場合、制限は 12 時間です。事前定義済みのデプロイ設定の場合、制限は 12 時間です。

名前	デフォルト	引き上げ可能	説明
AWS Lambda デプロイを数時間で実行する	サポートされている各リージョン: 50	いいえ	AWS Lambda デプロイが実行できる最大時間 (最初のトラフィックシフト

名前	デフォルト	引き上げ可能	説明
			と最後のトラフィックシフトの間の最大時間に対して 48 時間、および 2 つの可能なライフサイクルフックごとに 1 時間)
アカウントあたりの関連付けられたアプリケーション (リージョンごと)	サポートされている各リージョン: 1,000	あり	1 つのリージョンの AWS アカウントに関連付けられているアプリケーションの最大数
デプロイグループごとの関連付けられたアラーム	サポートされている各リージョン: 50	可能	デプロイグループに関連付けられるアラームの最大数
デプロイグループ内の Auto Scaling グループ	サポートされている各リージョン: 10	あり	デプロイグループの Amazon EC2 Auto Scaling グループの最大数
アカウントあたりの同時実行デプロイ	サポートされている各リージョン: 1,300	あり	AWS アカウントに関連付けられた同時デプロイの最大数。Amazon EC2 Auto Scaling グループのスケールアップされた Amazon EC2 インスタンスへの各デプロイは、EC2 インスタンスが関連付けられているアプリケーションごとに、単一の同時デプロイとしてカウントされます。

名前	デフォルト	引き上げ可能	説明
デプロイグループあたりの同時実行デプロイ	サポートされている各リージョン: 1	[いいえ]	デプロイグループへの同時デプロイの最大数。この制限により、同じデプロイグループに対して同じアプリケーションを同時デプロイすることを防ぎます。
アカウントあたりのカスタムデプロイ設定	サポートされている各リージョン: 50	いいえ	AWS アカウントに関連付けられたカスタムデプロイ設定の最大数
単一のアプリケーションに関連付けられたデプロイグループ	サポートされている各リージョン: 1,000	あり	1 つのアプリケーションに関連付けられるデプロイグループの最大数
EC2/オンプレミスの Blue/Green デプロイ実行 (時間)	サポートされている各リージョン: 109	いいえ	EC2/オンプレミスのブルー/グリーンデプロイで実行できる最大時間数 (上記の 2 つの各制限に 48 時間 + 発生する可能性のある 13 のライフサイクルイベントに対して 1 時間)
EC2/オンプレミスのインプレースデプロイ実行 (時間)	サポートされている各リージョン: 12	いいえ	EC2/オンプレミスインプレースデプロイで実行できる最大時間数

名前	デフォルト	引き上げ可能	説明
デプロイグループのイベント通知トリガー	サポートされている各リージョン: 10	あり	デプロイグループのイベント通知トリガーの最大数
アカウントあたりの GitHub 接続トークン	サポートされている各リージョン: 25	いいえ	1 つの AWS アカウントの GitHub 接続トークンの最大数
EC2/オンプレミスの Blue/Green デプロイ時に、デプロイが完了してから元のインスタンスが終了するまでの時間	サポートされている各リージョン: 48	いいえ	EC2/オンプレミスのブルー/グリーンデプロイにおいて、デプロイが完了してから元のインスタンスが終了するまでの最大時間数
EC2/オンプレミスの Blue/Green デプロイ時に、リビジョンのデプロイから、トラフィックが代替インスタンスにシフトするまでの時間	サポートされている各リージョン: 48	いいえ	EC2/オンプレミス のブルー/グリーンデプロイにおいて、リビジョンをデプロイしてから置き換え先インスタンスへトラフィックが移行されるまでの最大時間数

名前	デフォルト	引き上げ可能	説明
デプロイあたりのインスタンス数	us-east-1: 2,000 us-east-2: 2,000 us-west-2: 2,000 eu-central-1: 2,000 他のサポートされている各リージョン: 1,000	あり	1 つのデプロイ内のインスタンスの最大数
Blue/Green デプロイがデプロイに成功した後、元のデプロイからインスタンスを終了するまでの待機時間 (分)	サポートされている各リージョン: 2,800	いいえ	Blue/Green デプロイがデプロイに成功した後、元のデプロイからインスタンスを終了するまでの最大待機時間 (分)
AWS Lambda Canary または線形デプロイ中の最初のトラフィックシフトと最後のトラフィックシフトの間の分数	サポートされている各リージョン: 2,880	いいえ	AWS Lambda Canary または線形デプロイ中の最初のトラフィックシフトと最後のトラフィックシフトの間の最大分数

名前	デフォルト	引き上げ可能	説明
デプロイが失敗するまでの時間 (分) (ライフサイクルイベントが開始しない場合)	サポートされている各リージョン: 5	いいえ	ライフサイクルイベントが (1) コンソールまたは AWS CLI の create-deployment コマンドを使用してデプロイがトリガーされた場合、または (2) 以前のライフサイクルイベントが完了した後にデプロイが失敗するまでの最大分数。
Amazon ECS サービスに関連付けることができるデプロイグループの数	サポートされている各リージョン: 1	[いいえ]	Amazon ECS サービスに関連付けることができるデプロイグループの最大数
BatchGetOnPremisesInstances API アクションへ渡すことができるインスタンスの数	サポートされている各リージョン: 100	いいえ	BatchGetOnPremises Instances API アクションに渡すことができるインスタンスの最大数
アカウントあたりの進行中の同時実行デプロイによって使用されたインスタンスの数	us-east-1: 3,000 他のサポートされている各リージョン: 1,000	あり	進行中かつ 1 つのアカウントに関連付けられている同時デプロイで使用できるインスタンスの最大数
Amazon ECS デプロイ中のトラフィックルートのリスナー数	サポートされている各リージョン: 1	[いいえ]	Amazon ECS デプロイ中のトラフィックルートのリスナーの最大数

名前	デフォルト	引き上げ可能	説明
デプロイライフサイクルイベントが完了しなかった場合に失敗するまでの時間 (秒)	サポートされている各リージョン: 3,600 秒	いいえ	デプロイライフサイクルイベントが完了しなかった場合に失敗するまでの最大時間 (秒)
デプロイグループ名のサイズ	サポートされている各リージョン: 100	いいえ	デプロイグループ名の最大文字数
タグキーのサイズ	サポートされている各リージョン: 128	いいえ	タグキーの最大文字数
タグ値のサイズ	サポートされている各リージョン: 256	いいえ	タグ値の最大文字数
デプロイグループのタグ	サポートされている各リージョン: 10	いいえ	デプロイグループのタグの最大数
AWS Lambda デプロイ中に 1 増分で移行できるトラフィック	サポートされている各リージョン: 99	いいえ	AWS Lambda デプロイ中に 1 増分で移行できるトラフィックの最大割合

CodeDeploy のトラブルシューティング

このセクションのトピックは、CodeDeploy を使用するときに発生することがある問題やエラーの解決にお使いください。

> **Note**
>
> 多くのデプロイ失敗の原因を特定するには、デプロイプロセスで作成されたログファイルを確認できます。分かりやすいように、インスタンス別に表示するのではなく、Amazon CloudWatch Logs を使用してログファイルを集中的にモニタリングすることをお勧めします。詳細については、「Monitoring Deployments with Amazon CloudWatch Tools」を参照してください。

トピック

- 一般的なトラブルシューティングの問題
- EC2/オンプレミスのデプロイに関する問題のトラブルシューティング
- Amazon ECS のデプロイに関する問題のトラブルシューティング
- AWS Lambda デプロイの問題をトラブルシューティングする
- デプロイグループの問題のトラブルシューティング
- インスタンスの問題のトラブルシューティング
- GitHub トークンの問題のトラブルシューティング
- Amazon EC2 Auto Scaling の問題のトラブルシューティング
- のエラーコード AWS CodeDeploy

一般的なトラブルシューティングの問題

トピック

- 一般的なトラブルシューティングのチェックリスト
- CodeDeploy デプロイリソースは一部の AWS リージョンでのみサポートされています
- このガイドの手順が CodeDeploy コンソールと一致しない
- 必要な IAM ロールを取得できない

- 何らかのテキストエディタを使用して AppSpec ファイルとシェルスクリプトを作成すると、デプロイが失敗する場合がある
- macOS の Finder を使用してアプリケーションリビジョンをバンドルすると、デプロイが失敗することがある

一般的なトラブルシューティングのチェックリスト

次のチェックリストを使用して、失敗したデプロイをトラブルシューティングできます。

1. デプロイが失敗した理由を確認するには、「CodeDeploy デプロイの詳細を表示する」および「View Instance Details」を参照してください。原因を特定できない場合は、このチェックリストの項目を確認します。
2. インスタンスが正しく設定されているかどうかを確認します。
   - インスタンスは、指定された EC2 キーペアで起動されましたか? 詳細については、「Amazon EC2 ユーザーガイド」の「EC2 キーペア」を参照してください。
   - 正しい IAM インスタンスプロファイルがインスタンスにアタッチされていますか? 詳細については、「CodeDeploy と共に動作するように Amazon EC2 インスタンスを構成します」および「ステップ 4: Amazon EC2 インスタンス用の IAM インスタンスプロファイルを作成する」を参照してください。
   - インスタンスにタグが付けられていますか? 詳細については、「Amazon EC2 ユーザーガイド」の「コンソールでのタグの操作」を参照してください。
   - CodeDeploy エージェントは、インスタンスにインストール後、更新、実行されていますか? 詳細については、「CodeDeploy エージェントのオペレーションの管理」を参照してください。インストールされているエージェントのバージョンを確認するには、「CodeDeploy エージェントのバージョンを特定します。」を参照してください。
3. アプリケーションとデプロイグループの設定を確認します。
   - アプリケーション設定を確認するには、「CodeDeploy を使用してアプリケーション詳細を表示する」を参照してください。
   - デプロイグループの設定を確認するには、「CodeDeploy を使用したデプロイグループの詳細の表示」を参照してください。
4. アプリケーションリビジョンが正しく設定されていることを確認します
   - AppSpec ファイルの形式を確認します。詳細については、「CodeDeploy 用のアプリケーション仕様ファイルをリビジョンに追加」および「CodeDeploy AppSpec ファイルのリファレンス」を参照してください。

- GitHub レポジトリで Amazon S3 バケットを調べ、アプリケーションリビジョンが予期されている場所にあることを確認します。
- CodeDeploy アプリケーションリビジョンの詳細を調べ、正しく登録されていることを確認します。詳細については、「CodeDeploy を使用したアプリケーションリビジョン詳細の表示」を参照してください。
- Amazon S3 からデプロイする場合は、Amazon S3 バケットをチェックし、アプリケーションリビジョンをダウンロードするアクセス許可が CodeDeploy に付与されていることを確認します。バケットポリシーの詳細については、デプロイの前提条件 を参照してください。
- GitHub からデプロイする場合、GitHub リポジトリをチェックし、アプリケーションリビジョンをダウンロードするアクセス許可が CodeDeploy に付与されていることを確認します。詳細については、「CodeDeploy でデプロイを作成する」および「CodeDeploy のアプリケーションを使用した GitHub の認証」を参照してください。

5. サービスロールが正しく設定されているかどうかを確認します。詳細については、「ステップ 2: CodeDeployのサービスのロールを作成する」を参照してください。
6. 「CodeDeploy の開始方法」のステップに従って次の操作を行ったことを確認します。
   - 適切なアクセス許可を持つユーザーをプロビジョニングしました。
   - AWS CLIをインストールまたはアップグレードして設定する。
   - IAM インスタンスプロファイルとサービスロールを作成する。

   詳細については、「の ID とアクセスの管理 AWS CodeDeploy」を参照してください。
7. AWS CLI バージョン 1.6.1 以降を使用していることを確認します。インストールしたバージョンを確認するには、aws --version を呼び出します。

それでも失敗したデプロイをトラブルシューティングできない場合は、このトピックの他の問題を確認します。

CodeDeploy デプロイリソースは一部の AWS リージョンでのみサポートされています

または AWS CLI CodeDeploy コンソールからアプリケーション、デプロイグループ、インスタンス、またはその他のデプロイリソースが表示されないか、アクセスできない場合は、「」の AWS リージョンとエンドポイントにリストされているリージョンのいずれかを参照していることを確認してくださいAWS 全般のリファレンス。

CodeDeploy デプロイで使用される EC2 インスタンスと Amazon EC2 Auto Scaling グループは、これらの AWS リージョンのいずれかで起動および作成する必要があります。

を使用している場合は AWS CLI、 から aws configure コマンドを実行します AWS CLI。その後、デフォルトの AWS リージョンを表示および設定できます。

CodeDeploy コンソールを使用している場合は、ナビゲーションバーのリージョンセレクタから、サポートされている AWS リージョンのいずれかを選択します。

> **Important**
>
> 中国 (北京) リージョンまたは中国 (寧夏)でサービスを使用するには、そのリージョンのアカウントと認証情報が必要です。他の AWS リージョンのアカウントと認証情報は、北京および寧夏リージョンでは機能せず、その逆も同様です。
> CodeDeploy リソースキットのバケット名や CodeDeploy エージェントのインストール手順など、中国リージョン向けのいくつかのリソースに関する情報は、今回の「CodeDeploy ユーザーガイド」には含まれていません。
> 詳細については:
>
> - 中国 (北京) リージョンでの の開始方法 AWS の CodeDeploy
> - 「CodeDeploy User Guide for the China Regions (中国リージョン向け CodeDeploy ユーザーガイド)」(英語版 | 中国語版)

このガイドの手順が CodeDeploy コンソールと一致しない

このガイドの手順は、新しいコンソールデザインで記述されています。古いバージョンのコンソールを使用した場合、古い概念が反映され、本ガイドの基本的な手順がそのまま適用されます。新しいコンソールのヘルプにアクセスするには、情報アイコンを選択します。

必要な IAM ロールを取得できない

AWS CloudFormation スタックの一部として作成された IAM インスタンスプロファイルまたはサービスロールに依存している場合、スタックを削除すると、すべての IAM ロールも削除されます。これが、IAM ロールが IAM コンソールに表示されなくなり、CodeDeploy が予期どおり動作しなくなる理由と考えられます。この問題を解決するには、削除された IAM ロールを再作成する必要があります。

何らかのテキストエディタを使用して AppSpec ファイルとシェルスクリプトを作成すると、デプロイが失敗する場合がある

テキストエディタによっては、不適合で非表示の文字がファイルに含まれる場合があります。テキストエディタを使用して AppSpec ファイルやシェルスクリプトファイルを作成または変更し、Amazon Linux、Ubuntu Server、または RHEL インスタンスで実行する場合、これらのファイルに依存するデプロイは失敗する可能性があります。CodeDeploy がこれらのファイルをデプロイ中に使用したときに、このような文字が存在すると、トラブルシューティングが困難な AppSpec ファイルの検証エラーにつながり、スクリプトの実行が失敗する可能性があります。

CodeDeploy コンソールのデプロイのイベント詳細ページで、[ログを表示する] を選択します。(または AWS CLI 、 を使用して get-deployment-instance コマンドを呼び出します。) `invalid character`、`command not found`、`file not found` のようなエラーを探します。

この問題に対処するには、次のことをお勧めします。

- 改行 (`^M` 文字) などの非表示の文字を AppSpec ファイルとシェルスクリプトファイルに含めるテキストエディタは使用しない。
- AppSpec ファイルやシェルスクリプトファイルで改行など非表示の文字を表示するテキストエディタを使用して、対象となる可能性のある文字を見つけ、削除できるようにする。このようなタイプのテキストエディタの例については、インターネットで「改行を表示するテキストエディタ」を検索します。
- Amazon Linux、Ubuntu サーバー、または RHEL インスタンスで動作するテキストエディタを使用して、Amazon Linux、Ubuntu サーバー、または RHEL インスタンスで動作するシェルスクリプトファイルを作成してください。このようなタイプのテキストエディタの例については、インターネットで「Linux シェルスクリプトエディタ」を検索します。
- Windows または macOS でテキストエディタを使用して、Amazon Linux、Ubuntu Server、 または RHEL インスタンスで実行するシェルスクリプトファイルを作成する場合は、Windows または macOS 形式のテキストを Unix 形式に変換するプログラムまたはユーティリティを使用する。このようなプログラムとユーティリティの例については、インターネットで「DOS から UNIX へ」または「Mac から UNIX へ」を検索します。必ず、ターゲットのオペレーティングシステムで、変換されたシェルスクリプトファイルをテストします。

macOS の Finder を使用してアプリケーションリビジョンをバンドルすると、デプロイが失敗することがある

Mac の Finder グラフィカルユーザーインタフェース (GUI) アプリケーションを使用して、AppSpec ファイルおよび関連ファイルとスクリプトをアプリケーションリビジョンのアーカイブ (.zip) ファイルにバンドル (zip) すると、デプロイが失敗する場合があります。これは、Finder が .zip ファイルに中間の `__MACOSX` フォルダを作成し、そこにコンポーネントファイルを配置するためです。CodeDeploy はコンポーネントファイルを見つけることができないため、デプロイは失敗します。

この問題に対処するには、 を使用して push コマンド AWS CLI を呼び出し、コンポーネントファイルを期待される構造に圧縮することをお勧めします。または、GUI の代わりにターミナルを使用してコンポーネントファイルを zip 圧縮できます。ターミナルでは、中間の `__MACOSX` フォルダは作成されません。

EC2/オンプレミスのデプロイに関する問題のトラブルシューティング

トピック

- CodeDeploy プラグイン CommandPoller の認証情報不足エラー
- デプロイが失敗し、PKCS7 署名付きメッセージの検証に失敗しました」というメッセージが表示される
- 同じデプロイ先のインスタンスに対する同じファイルのデプロイや再デプロイは失敗し、「指定したファイルはこの場所に既に存在するため、デプロイに失敗しました」というエラーが返されます。
- ファイルパスが長いと、「そのようなファイルまたはディレクトリはありません」というエラーが発生します
- 長時間実行されているプロセスにより、デプロイが失敗することがある
- AllowTraffic ライフサイクルイベントが失敗し、デプロイログにエラーが報告されない場合のトラブルシューティング
- 失敗した ApplicationStop、BeforeBlockTraffic、または AfterBlockTraffic デプロイライフサイクルイベントのトラブルシューティング
- 「不明なエラー: 読み取り用に開いていません」で失敗した DownloadBundle デプロイライフサイクルイベントのトラブルシューティング

- スキップされたすべてのライフサイクルイベントのエラーをトラブルシューティングする
- Windows PowerShell スクリプトで、デフォルトで 64 ビットバージョンの Windows PowerShell スクリプトを使用できない

> **Note**
>
> 多くのデプロイ失敗の原因は、デプロイプロセス中に作成されたログファイルを確認して特定できます。分かりやすいように、インスタンス別に表示するのではなく、Amazon CloudWatch Logs を使用してログファイルを集中的にモニタリングすることをお勧めします。詳細については、「CloudWatch Logs コンソールで CodeDeploy ログを見る」を参照してください。

> **Tip**
>
> EC2/オンプレミスデプロイに関連する多くのトラブルシューティングタスクを自動化するランブックについては、「AWS Systems Manager オートメーションランブックリファレンス」の「AWSSupport-TroubleshootCodeDeploy」を参照してください。

CodeDeploy プラグイン CommandPoller の認証情報不足エラー

`InstanceAgent::Plugins::CodeDeployPlugin::CommandPoller: Missing credentials - please check if this instance was started with an IAM instance profile` のようなエラーが表示された場合は、次のいずれがが原因の可能性があります。

- デプロイ先のインスタンスに、関連付けられている IAM インスタンスプロファイルがない。
- IAM インスタンスプロファイルに、適切なアクセス許可が設定されていない。

IAM インスタンスプロファイルによって、CodeDeploy と通信し、リビジョンを Amazon S3 からダウンロードするためのアクセス許可が CodeDeploy エージェントに付与されている。EC2 インスタンスの場合は、「の ID とアクセスの管理 AWS CodeDeploy」を参照してください。オンプレミスインスタンスの場合は、Working with On-Premises Instances を参照してください。

デプロイが失敗し、PKCS7 署名付きメッセージの検証に失敗しました」というメッセージが表示される

このエラーメッセージは、SHA-1 ハッシュアルゴリズムのみをサポートするバージョンの CodeDeploy エージェントをインスタンスが実行していることを示します。SHA-2 ハッシュアルゴリズムのサポートは、2015 年 11 月にリリースされたバージョン 1.0.1.854 の CodeDeploy エージェントで導入されました。2016 年 10 月 17 日から、バージョン 1.0.1.854 以前の CodeDeploy エージェントがインストールされている場合、デプロイは失敗します。詳細については、「AWS SSL 証明書のハッシュアルゴリズムを SHA256 に変更すること」、「NOTICE: バージョン 1.0.1.85 より古い CodeDeploy ホストエージェントを使用停止にする」、および「CodeDeploy エージェントを更新する」を参照してください。

同じデプロイ先のインスタンスに対する同じファイルのデプロイや再デプロイは失敗し、「指定したファイルはこの場所に既に存在するため、デプロイに失敗しました」というエラーが返されます。

CodeDeploy でファイルをデプロイしようとして、指定したデプロイ先のインスタンスに同じファイルが既に存在している場合、そのインスタンスへのデプロイは失敗する場合があります。「指定したファイルはこの場所 (*location-name*) に既に存在しているため、デプロイに失敗しました」というエラーメッセージが返される場合があります。このエラーが発生するのは、デプロイごとに CodeDeploy では最初にクリーンアップログファイルにリストされているすべてのファイルを以前のデプロイから削除するためです。このクリーンアップファイルにリストされていないファイルがターゲットのインストールフォルダにある場合、CodeDeploy エージェントはデフォルトでこれをエラーとして解釈し、デプロイに失敗します。

> **Note**
>
> Amazon Linux、RHEL、および Ubuntu Server の各インスタンスでは、クリーンアップファイルは `/opt/codedeploy-agent/deployment-root/deployment-instructions/` にあります。Windows Server インスタンスでは、場所は `C:\ProgramData\Amazon\CodeDeploy\deployment-instructions\` です。

このエラーを最も簡単に回避するには、デプロイを失敗させるデフォルト動作とは別のオプションを指定します。デプロイごとに、デプロイを失敗させるか、クリーンアップファイルにリストされていないファイルを上書きするか、インスタンスの既存ファイルを保持するかを選択できます。

上書きオプションは、最後のデプロイ後に手動でインスタンスに追加した同じ名前のファイルを、次回のアプリケーションリビジョンにも追加する場合などに便利です。

保持オプションは、次回のデプロイでインスタンスに追加するファイルを、アプリケーションリビジョンパッケージには追加する必要がない場合に選択できます。保持オプションは、アプリケーションファイルが既に本番稼働環境にあり、これらを CodeDeploy を使用して初めてデプロイする場合にも便利です。詳細については、「EC2/オンプレミスコンピューティングプラットフォームのデプロイ作成 (コンソール)」および「既存のコンテンツでのロールバック動作」を参照してください。

`The deployment failed because a specified file already exists at this location` デプロイメント失敗のトラブルシューティング

CodeDeploy がデプロイ先で検出したコンテンツを上書きまたは保持するオプションを指定しない場合 (または既存のコンテンツを処理するデプロイオプションをプログラムによるコマンドで指定しない場合)、エラーのトラブルシューティングを行うことを選択できます。

次の情報は、コンテンツを保持または上書きしないことを選択した場合にのみ該当します。

同じ名前と場所のファイルの再デプロイを試みる場合、以前に使用した基になるデプロイグループ ID のアプリケーション名およびデプロイグループを指定すると、成功する可能性が高まります。CodeDeploy は、基になるデプロイグループ ID を使用して、再デプロイする前に削除するファイルを識別します。

新しいファイルのデプロイや、インスタンスの同じ場所への同じファイルの再デプロイは、次の理由により失敗することがあります。

- 同じリビジョンの同じインスタンスへの再デプロイのため、別のアプリケーション名を指定した。デプロイグループ名が同じでも、別のアプリケーション名を使用すると、基になる別のデプロイグループ ID が使用されるため、再デプロイは失敗します。
- アプリケーションのデプロイグループを削除して再作成し、同じリビジョンをデプロイグループに対して再デプロイしようとした。デプロイグループ名が同じでも、CodeDeploy は基になるデプロイグループ ID を参照するため、再デプロイは失敗します。
- CodeDeploy のアプリケーションとデプロイグループを削除し、削除したものと同じ名前の新しいアプリケーションとデプロイグループを作成した。その後、以前のデプロイグループにデプロイされていたリビジョンを、同じ名前の新しいデプロイグループに再デプロイしようとした。アプリケーション名とデプロイグループ名が同じであっても、CodeDeploy は削除したデプロイグループの ID を引き続き参照するため、再デプロイは失敗します。

- リビジョンをデプロイグループにデプロイし、同じリビジョンを同じインスタンスの別のデプロイグループにデプロイした。CodeDeploy は別の基になるデプロイグループ ID を参照するため、2 番目のデプロイは失敗します。
- リビジョンを 1 つのデプロイグループにデプロイし、別のリビジョンを同じインスタンスの別のデプロイグループにデプロイした。2 番目のデプロイグループがデプロイを試みる同じ名前と場所のファイルが、少なくとも 1 つある。CodeDeploy は 2 番目のデプロイが開始される前に既存のファイルを削除しないため、2 番目のデプロイは失敗します。どちらのデプロイも、異なるデプロイグループ IDs。
- CodeDeploy でリビジョンデプロイしたが、同じ名前と場所のファイルが少なくとも 1 つある。デフォルトでは、CodeDeploy はデプロイが開始される前に既存のファイルを削除しないため、デプロイは失敗します。

これらの状況に対応するには、次のいずれかを実行します。

- 以前のデプロイされた場所とインスタンスからファイルを削除してから、もう一度デプロイを試みます。
- リビジョンの AppSpec ファイルで、ApplicationStop または BeforeInstall デプロイライフサイクルイベントのいずれかにおいて、リビジョンでインストールするファイルと一致するファイルがいずれかの場所にある場合、これらを削除するためのカスタムスクリプトを指定します。
- 以前のデプロイの一部ではない場所またはインスタンスにファイルをデプロイまたは再デプロイします。
- アプリケーションまたはデプロイグループを削除する前に、インスタンスにコピーするファイルを指定しない AppSpec ファイル を含むリビジョンをデプロイします。このデプロイの場合、削除しようとしているものと同じ、基になるアプリケーションおよびデプロイグループ ID を使用するアプリケーション名とデプロイグループ名を指定します (get-deployment-group コマンドで、デプロイグループ ID を取得することができます。) CodeDeploy は、基になるデプロイグループ ID と AppSpec ファイルを使用して、前回成功したデプロイでインストールしたすべてのファイルを削除します。

ファイルパスが長いと、「そのようなファイルまたはディレクトリはありません」というエラーが発生します

Windows インスタンスへのデプロイでは、appspec.yml ファイルのファイルセクションに 260 文字を超えるファイルパスがあると、デプロイが次のような エラーで失敗することがあります。

```
No such file or directory @ dir_s_mkdir - C:\your-long-file-path
```

このエラーは、Windows がデフォルトで 260 文字を超えるファイルパスを許可していないために発生します。詳細については、Microsoft のドキュメントを参照してください。

CodeDeploy エージェントバージョン 1.4.0 以降では、エージェントのインストールプロセスに応じて、次の 2 つの方法で長いファイルパスを有効にできます。

CodeDeploy エージェントがまだインストールされていない場合:

1. CodeDeploy エージェントをインストールする予定のマシンで、次のコマンドを使用して LongPathsEnabled Windows レジストリキーを有効にします。

   ```
   New-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Control\FileSystem"
             -Name "LongPathsEnabled" -Value 1 -PropertyType DWORD -Force
   ```

2. CodeDeploy エージェントをインストールします。詳細については、「CodeDeploy エージェントをインストールする」を参照してください。

CodeDeploy エージェントがすでにインストールされている場合:

1. CodeDeploy エージェントマシンで、次のコマンドを使用して LongPathsEnabled Windows レジストリキーを有効にします。

   ```
   New-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Control\FileSystem"
   -Name "LongPathsEnabled" -Value 1 -PropertyType DWORD -Force
   ```

2. CodeDeploy エージェントを再起動して、レジストリキーの変更を有効にします。エージェントを再起動するには、次のコマンドを使用します。

   ```
   powershell.exe -Command Restart-Service -Name codedeployagent
   ```

長時間実行されているプロセスにより、デプロイが失敗することがある

Amazon Linux、Ubuntu Server、RHEL インスタンスへのデプロイでは、長時間実行されるプロセスを開始するデプロイスクリプトがある場合、CodeDeploy はデプロイライフサイクルイベントでの待機に長い時間がかかり、その後デプロイに失敗することがあります。これは、そのイベントのフォアグラウンドおよびバックグラウンドで予期されるよりも長くプロセスが実行される場合、プロセスがまだ予期どおり実行されていても、CodeDeploy によってデプロイは中止され、失敗するためです。

たとえば、アプリケーションリビジョンのルートに 2 つのファイル (after-install.sh および sleep.sh) が含まれているとします。その AppSpec ファイルファイル には次の指示が含まれています。

```
version: 0.0
os: linux
files:
  - source: ./sleep.sh
    destination: /tmp
hooks:
  AfterInstall:
    - location: after-install.sh
      timeout: 60
```

after-install.sh ファイルは AfterInstall アプリケーションライフサイクルイベント中に実行されます。そのコンテンツは次のとおりです。

```
#!/bin/bash
/tmp/sleep.sh
```

sleep.sh ファイルには以下が含まれます。これはプログラムの実行を 3 分 (180 秒) 停止し、長時間実行されるプロセスをシミュレートします。

```
#!/bin/bash
sleep 180
```

after-install.sh が sleep.sh を呼び出すと、sleep.sh が起動し、3 分 (180 秒) 実行します。これは CodeDeploy が sleep.sh (および関連で after-install.sh) の実行を停止させると予期した時間を 2 分 (120 秒) 経過した時間です。1 分 (60 秒) のタイムアウト後、sleep.sh は予期どおり実行を継続しますが、CodeDeploy は停止し、AfterInstall アプリケーションライフサイクルイベントでデプロイに失敗します。次のエラーが表示されます。

Script at specified location: after-install.sh failed to complete in 60 seconds.

単純に & でアンパサンド (after-install.sh) を追加して、バックグラウンドで sleep.sh を実行することはできません。

```
#!/bin/bash
```

```
# Do not do this.
/tmp/sleep.sh &
```

この操作を行うと、デプロイは最大でデフォルトの 1 時間のデプロイライフサイクルイベントのタイムアウト期間まで保留状態になり、その後 CodeDeploy は以前のように AfterInstall アプリケーションライフサイクルイベントでデプロイを停止し、失敗させます。

`after-install.sh` で、次のように `sleep.sh` を呼び出します。これにより、CodeDeploy はプロセスの実行開始後に続行できます。

```
#!/bin/bash
/tmp/sleep.sh > /dev/null 2> /dev/null < /dev/null &
```

前の呼び出しで、`sleep.sh` はバックグラウンドで実行を開始し、stdout、stderr、および stdin を `/dev/null` にリダイレクトするプロセスの名前です。

AllowTraffic ライフサイクルイベントが失敗し、デプロイログにエラーが報告されない場合のトラブルシューティング

AllowTraffic ライフサイクルイベント中に Blue/Green デプロイが失敗しても、デプロイログに失敗の原因が表示されない場合があります。

通常、この障害は、デプロイグループへのトラフィック管理に使用される Classic Load Balancer、Application Load Balancer、または Network Load Balancer の Elastic Load Balancing のヘルスチェックが間違って設定されていることが原因です。

この問題を解決するには、ロードバランサーのヘルスチェックの設定エラーを確認して修正します。

Classic Load Balancer については、「Classic Load Balancer のユーザーガイド」の「Configure health checks for your Classic Load Balancer (Classic Load Balancer のヘルスチェックの設定)」、および「Elastic Load Balancing API リファレンスバージョン 2012-06-01」の「ConfigureHealthCheck」を参照してください。

Application Load Balancer については、「Application Load Balancer のユーザーガイド」の「ターゲットグループのヘルスチェック」を参照してください。

Network Load Balancer については、Network Load Balancer ユーザーガイド の「ターゲットグループのヘルスチェック」を参照してください。

失敗した ApplicationStop、BeforeBlockTraffic、または AfterBlockTraffic デプロイライフサイクルイベントのトラブルシューティング

デプロイ時に、CodeDeploy エージェントは ApplicationStop、BeforeBlockTraffic、および AfterBlockTraffic に対して前回の正常なデプロイで AppSpec ファイルに指定されていたスクリプトを実行します。(他のすべてのスクリプトは、現在のデプロイの AppSpec ファイルから実行されます)。これらのスクリプトのいずれかにエラーがあって正常に実行されない場合、デプロイに失敗することがあります。

これらの失敗の原因としては以下のようなことが考えられます。

- CodeDeploy エージェントは *deployment-group-id*_last_successful_install ファイルを正しい場所で見つけたが、*deployment-group-id*_last_successful_install ファイルにリストされている場所が存在しない。

  Amazon Linux、Ubuntu サーバー、および RHEL インスタンス では、このファイルは /opt/codedeploy-agent/deployment-root/deployment-instructions に存在する必要があります。

  Windows Server インスタンスでは、このファイルは C:\ProgramData\Amazon\CodeDeploy\deployment-instructions フォルダに保存されている必要があります。
- *deployment-group-id*_last_successful_install ファイルに示された場所で、AppSpec ファイルが無効であるか、スクリプトが正常に実行されない。
- スクリプトに修正できないエラーがあるため、正常に実行されることはありません。

CodeDeploy コンソールを使用して、これらのいずれかのイベント中にデプロイが失敗した理由を調査します。デプロイの詳細ページで、[View events] を選択します。インスタンスの詳細ページの [ApplicationStop] 行、[BeforeBlockTraffic] 行、または [AfterBlockTraffic] 行で、[ログを表示する] を選択します。または AWS CLI 、 を使用して get-deployment-instance コマンドを呼び出します。

前回の正常なデプロイのスクリプトが正常に実行されないために失敗した場合は、デプロイを作成して [ApplicationStop]、[BeforeBlockTraffic]、および [AfterBlockTraffic] のエラーを無視するように指定します。これには、以下の 2 つの方法があります。

- CodeDeploy コンソールを使用してデプロイを作成します。[デプロイの作成] ページの [ApplicationStop ライフサイクルイベントの障害] で、[このインスタンス上のライフサイクルイベントの障害で、デプロイを失敗させない] を選択します。

- AWS CLI を使用して create-deployment コマンドを呼び出し、 `--ignore-application-stop-failures`オプションを含めます。

アプリケーションリビジョンを再度デプロイすると、これら 3 つのライフサイクルイベントのいずれが失敗してもデプロイは続行されます。これらのライフサイクルイベントの修正済みスクリプトが新しいリビジョンに含まれている場合は、この修正を適用しなくても以降のデプロイは正常に実行されます。

「不明なエラー: 読み取り用に開いていません」で失敗した DownloadBundle デプロイライフサイクルイベントのトラブルシューティング

Amazon S3 からアプリケーションリビジョンをデプロイしようとして、DownloadBundle デプロイライフサイクルイベント中に「`UnknownError: not opened for reading`」というエラーでデプロイが失敗する場合:

- Amazon S3 の内部サーバーエラーが発生しました。アプリケーションリビジョンをもう一度デプロイします。
- EC2 インスタンスの IAM インスタンスプロファイルに、Amazon S3 のアプリケーションリビジョンにアクセスするアクセス許可がありません。Amazon S3 バケットポリシーの詳細については、「Amazon S3 に CodeDeploy のリビジョンをプッシュする (EC2/オンプレミスのデプロイのみ)」と「デプロイの前提条件」 を参照してください。
- デプロイ先のインスタンスは 1 つの AWS リージョン (米国西部 (オレゴン) など) に関連付けられますが、アプリケーションリビジョンを含む Amazon S3 バケットは別の AWS リージョン (米国東部 (バージニア北部) など) に関連付けられます。アプリケーションリビジョンがインスタンスと同じ AWS リージョンに関連付けられている Amazon S3 バケットにあることを確認します。

デプロイのイベント詳細ページの [Download bundle (バンドルのダウンロード)] 行で、[ログを表示する] を選択します。または AWS CLI 、 を使用して get-deployment-instance コマンドを呼び出します。このエラーが発生した場合、エラーコード「`UnknownError`」とエラーメッセージ「`not opened for reading`」のエラーが出力に表示されます。

このエラーの原因を特定するには:

1. インスタンスの少なくとも 1 つでワイヤログを有効にして、もう一度アプリケーションリビジョンをデプロイします。

2. ワイヤログファイルを調べてエラーを見つけます。この問題の一般的なエラーメッセージには、「access denied」という語句が含まれます。
3. ログファイルを確認した後、ワイヤログを無効にして、ログファイルサイズと、今後インスタンスでプレーンテキストで出力に表示される可能性がある機密情報の量を減らすことをお勧めします。

ワイヤログファイルを探し、ワイヤログを有効または無効にする方法については、「CodeDeploy エージェント構成リファレンス」の「`:log_aws_wire:`」を参照してください。

スキップされたすべてのライフサイクルイベントのエラーをトラブルシューティングする

EC2 またはオンプレミスのデプロイのライフサイクルイベントがすべてスキップされた場合は、`The overall deployment failed because too many individual instances failed deployment, too few healthy instances are available for deployment, or some instances in your deployment group are experiencing problems. (Error code: HEALTH_CONSTRAINTS)` のようなエラーが表示されます。考えられる原因と解決策は次のとおりです。

- CodeDeploy エージェントがインスタンスにインストールされていないか、インスタンスで実行されていない。CodeDeploy エージェントが実行されていることを確認するには:
  - Amazon Linux RHEL または Ubuntu server の場合は、以下を実行します。

    ```
    systemctl status codedeploy-agent
    ```

  - Windows の場合は、以下を実行します。

    ```
    powershell.exe -Command Get-Service -Name CodeDeployagent
    ```

  CodeDeploy エージェントがインストールまたは実行されていない場合は、CodeDeploy エージェントが実行されていることの確認 を参照してください。

  インスタンスがポート 443 を使用して CodeDeploy または Amazon S3 のパブリックエンドポイントに到達できない可能性があります。以下のいずれかを行ってください。

  - インスタンスにパブリック IP アドレスを割り当て、そのルートテーブルを使用してインターネットアクセスを許可します。インスタンスに関連付けられているセキュリティグループで、

ポート 443 (HTTPS) 経由で送信アクセスが許可されていることを確認してください。詳細については、「CodeDeploy エージェントの通信プロトコルとポート」を参照してください。

  - インスタンスがプライベートサブネットにプロビジョニングされている場合は、ルートテーブルで、インターネットゲートウェイではなく NAT ゲートウェイを使用します。詳細については、「NAT ゲートウェイ」を参照してください。

- CodeDeploy のサービスロールに、必要なアクセス許可が付与されていない可能性があります。CodeDeploy サービスロールを設定するには、「ステップ 2: CodeDeployのサービスのロールを作成する」を参照してください。
- HTTP プロキシを使用している場合は、CodeDeploy エージェントの設定ファイルの `:proxy_uri:` 設定で指定されていることを確認してください。詳細については、「CodeDeploy エージェント設定リファレンス」を参照してください。
- デプロイメントインスタンスの日時が、デプロイメントリクエストの日時と一致しない場合があります。CodeDeploy エージェントログファイルに、`Cannot reach InstanceService: Aws::CodeDeployCommand::Errors::InvalidSignatureException - Signature expired` のようなエラーがないか確認します。このエラーが表示されている場合は、InvalidSignatureException – Signature expired: [time] is now earlier than [time]」デプロイエラーのトラブルシューティング のステップに従います。詳細については、「CodeDeploy EC2/オンプレミスデプロイのログデータの表示」を参照してください。
- インスタンスのメモリまたはハードディスクの空き容量が不足しているため、CodeDeploy エージェントが停止している可能性があります。インスタンス上のアーカイブされたデプロイメントの数が減るように、CodeDeploy エージェントの `max_revisions` 設定を更新してください。EC2 インスタンスにこのプロセスを行っても問題が解決しない場合は、インスタンスのサイズを大きくすることを検討してください。たとえば、インスタンスタイプが `t2.small` の場合は、`t2.medium` の使用を検討します。詳細については、「CodeDeploy エージェントでインストールされるファイル」、「CodeDeploy エージェント設定リファレンス」、「 インスタンスタイプ」を参照してください。
- デプロイ先のインスタンスに、IAM インスタンスプロファイルがアタッチされていないか、または必要なアクセス許可が付与されていない IAM インスタンスプロファイルがアタッチされている可能性があります。
  - IAM インスタンスプロファイルがインスタンスにアタッチされていない場合は、必要なアクセス許可を付与したインスタンスプロファイルを作成し、アタッチします。
  - IAM インスタンスプロファイルがインスタンスにすでにアタッチされている場合は、必要なアクセス許可があることを確認します。

必要なアクセス許可が付与されているインスタンスプロファイルがアタッチされていることを確認したら、インスタンスを再起動します。詳細については、Amazon EC2 ユーザーガイド の「ステップ 4: Amazon EC2 インスタンス用の IAM インスタンスプロファイルを作成する」と「Amazon EC2 の IAM ロール」を参照してください。

Windows PowerShell スクリプトで、デフォルトで 64 ビットバージョンの Windows PowerShell スクリプトを使用できない

デプロイの一部として実行中の Windows PowerShell スクリプトが 64 ビットの機能に依存している場合 (たとえば、32 ビットアプリケーションで許可されるよりも多くのメモリを消費する、64 ビットバージョンのみで提供されるライブラリを呼び出すなど)、スクリプトはクラッシュするか、予期どおりに実行されない可能性があります。これは、CodeDeploy はデフォルトで 32 ビットバージョンの Windows PowerShell を使用して、アプリケーションリビジョンの一部である Windows PowerShell スクリプトを実行するためです。

64 ビットバージョンの Windows PowerShell で実行する必要があるスクリプトの先頭に、次のようなコードを追加します。

```
# Are you running in 32-bit mode?
#   (\SysWOW64\ = 32-bit mode)

if ($PSHOME -like "*SysWOW64*")
{
  Write-Warning "Restarting this script under 64-bit Windows PowerShell."

  # Restart this script under 64-bit Windows PowerShell.
  #   (\SysNative\ redirects to \System32\ for 64-bit mode)

  & (Join-Path ($PSHOME -replace "SysWOW64", "SysNative") powershell.exe) -File `
    (Join-Path $PSScriptRoot $MyInvocation.MyCommand) @args

  # Exit 32-bit script.

  Exit $LastExitCode
}

# Was restart successful?
Write-Warning "Hello from $PSHOME"
Write-Warning "  (\SysWOW64\ = 32-bit mode, \System32\ = 64-bit mode)"
```

```
Write-Warning "Original arguments (if any): $args"

# Your 64-bit script code follows here...
# ...
```

このコードのファイルパス情報を直観的にはわかりにくいかもしれませんが、32 ビットの Windows PowerShell では次のようなパスが使用されます。

```
c:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
```

64 ビットの Windows PowerShell では次のようなパスが使用されます。

```
c:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
```

Amazon ECS のデプロイに関する問題のトラブルシューティング

トピック

- 置き換えタスクセットを待っている間にタイムアウトが発生します
- 通知が継続されるのを待っている間のタイムアウトの発生
- IAM ロールに十分なアクセス許可がありません
- ステータスコールバックを待っている間に、デプロイがタイムアウトした
- 1 つ以上のライフサイクルイベントの検証機能が失敗したため、デプロイが失敗しました
- 「プライマリタスクセットのターゲットグループはリスナーの後ろにある必要があります」というエラーのため、ELB を更新できませんでした
- Auto Scaling を使用するとデプロイが失敗することがあります
- ALB のみが段階的なトラフィックルーティングをサポートしているため、デプロイグループの作成/更新時には、代わりに AllAtOnce トラフィックルーティングを使用してください
- デプロイが成功しても、置き換えタスクセットは Elastic Load Balancing のヘルスチェックに失敗し、アプリケーションがダウンしています
- 1 つのデプロイグループに複数のロードバランサーをアタッチできますか?
- ロードバランサーなしで CodeDeploy のブルー/グリーンデプロイを実行できますか?
- デプロイ中に Amazon ECS サービスを新しい情報で更新する方法を教えてください。

置き換えタスクセットを待っている間にタイムアウトが発生します

問題: CodeDeploy を使用して Amazon ECS アプリケーションをデプロイする際に、次のエラーメッセージが表示される。

```
The deployment timed out while waiting for the replacement task set to
become healthy. This time out period is 60 minutes.
```

考えられる原因: このエラーは、タスク定義ファイルまたはその他のデプロイ関連ファイルに誤りがある場合に発生する可能性があります。例えば、タスク定義ファイルの `image` フィールドにタイプミスがあると、Amazon ECS は間違ったコンテナイメージを取得しようとして失敗し続けるため、このエラーが発生します。

解決方法と次のステップ:

- タスク定義ファイルやその他のファイルの入力ミスや設定の問題を修正します。
- 関連する Amazon ECS サービスイベントをチェックして、置き換えタスクが正常に動作しない理由を調べます。Amazon ECS イベントに関する詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「Amazon ECS イベント」を参照してください。
- Amazon Elastic Container Service デベロッパーガイドの「Amazon ECS トラブルシューティング」セクションで、イベント内のメッセージに関連するエラーについて参照してください。

通知が継続されるのを待っている間のタイムアウトの発生

問題: CodeDeploy を使用して Amazon ECS アプリケーションをデプロイする際に、次のエラーメッセージが表示される。

```
The deployment timed out while waiting for a notification to continue. This
time out period is n minutes.
```

考えられる原因: このエラーは、デプロイグループの作成時に [トラフィックを再ルーティングするタイミングを指定します] フィールドで待機時間を指定したものの、待機時間が経過する前にデプロイを完了できなかった場合に発生する可能性があります。

解決方法と次のステップ:

- デプロイグループで、[トラフィックを再ルーティングするタイミングを指定します] をより長い時間に設定して再デプロイします。詳細については、「Amazon ECS デプロイ用のデプロイグループを作成する (コンソール)」を参照してください。

- デプロイグループで、[トラフィックを再ルーティングするタイミングを指定します] を [すぐにトラフィックを再ルーティング] に変更して、再デプロイします。詳細については、「Amazon ECS デプロイ用のデプロイグループを作成する (コンソール)」を参照してください。
- `--deployment-wait-type` オプションを に設定して、aws deploy continue-deployment AWS CLI コマンドを再デプロイしてから実行しますREADY_WAIT。[トラフィックを再ルーティングするタイミングを指定します] で指定した時間が経過する前に、このコマンドを必ず実行してください。

IAM ロールに十分なアクセス許可がありません

問題: CodeDeploy を使用して Amazon ECS アプリケーションをデプロイしている際に、次のエラーメッセージが表示されます。

```
The IAM role role-arn does not give you permission to perform operations in the following AWS service: AWSLambda.
```

考えられる原因: このエラーは、AppSpec ファイルの Hooks セクションで Lambda 関数を指定したが、Lambda サービスに CodeDeploy アクセス許可を付与しなかった場合に発生する可能性があります。

解決方法: CodeDeploy サービスロールに `lambda:InvokeFunction` アクセス許可を追加します。このアクセス許可を追加するには、**AWSCodeDeployRoleForECS** か **AWSCodeDeployRoleForECSLimited** の AWSマネージドポリシーのいずれかをロールに追加します。これらのポリシーと、ポリシーを CodeDeploy サービスロールに追加する方法については、「ステップ 2: CodeDeployのサービスのロールを作成する」を参照してください。

ステータスコールバックを待っている間に、デプロイがタイムアウトした

問題: CodeDeploy を使用して Amazon ECS アプリケーションをデプロイしている際に、次のエラーメッセージが表示されます。

```
The deployment timed out while waiting for a status callback. CodeDeploy expects a status callback within one hour after a deployment hook is invoked.
```

考えられる原因: このエラーは、AppSpec ファイルの Hooks セクションで Lambda 関数を指定したが、Lambda 関数が CodeDeploy に `Succeeded` または `Failed` ステータスを返すのに必要な `PutLifecycleEventHookExecutionStatus` API を呼び出せなかった場合に発生する可能性があります。

解決方法と次のステップ:

- AppSpec ファイルで指定した Lambda 関数が使用する Lambda 実行ロールに `codedeploy:putlifecycleEventHookExecutionStatus` アクセス許可を追加します。このアクセス許可により、Lambda 関数には CodeDeploy に `Succeeded` または `Failed` のステータスを返すアクセス許可が付与されます。Lambda 実行ロールの詳細については、「AWS Lambda ユーザーガイド」の「Lambda 実行ロール」を参照してください。
- Lambda 関数のコードと実行ログを確認して、Lambda 関数が CodeDeploy の `PutLifecycleEventHookExecutionStatus` API を呼び出して、ライフサイクル検証テストが `Succeeded` か `Failed` であるかを CodeDeploy に通知していることを確認します。`putlifecycleEventHookExecutionStatus` API の詳細については、「AWS CodeDeploy API リファレンス」の「PutLifecycleEventHookExecutionStatus」を参照してください。Lambda 実行ログの詳細については、「AWS Lambdaでの Amazon CloudWatch Logs の使用」を参照してください。

1 つ以上のライフサイクルイベントの検証機能が失敗したため、デプロイが失敗しました

問題: CodeDeploy を使用して Amazon ECS アプリケーションをデプロイしている際に、次のエラーメッセージが表示されます。

```
The deployment failed because one or more of the lifecycle event validation functions failed.
```

考えられる原因: このエラーは、AppSpec ファイルの `Hooks` セクションで Lambda 関数を指定したが、Lambda 関数が `PutLifecycleEventHookExecutionStatus` を呼び出した際に CodeDeploy に `Failed` を返した場合に発生する可能性があります。この失敗は、ライフサイクル検証テストが失敗したことを CodeDeploy に示します。

考えられる次のステップ: Lambda 実行ログを確認して、検証テストコードが失敗している理由を確認します。Lambda 実行ログの詳細については、「AWS Lambdaでの Amazon CloudWatch Logs の使用」を参照してください。

「プライマリタスクセットのターゲットグループはリスナーの後ろにある必要があります」というエラーのため、ELB を更新できませんでした

問題: CodeDeploy を使用して Amazon ECS アプリケーションをデプロイしている際に、次のエラーメッセージが表示されます。

```
The ELB could not be updated due to the following error: Primary taskset
target group must be behind listener
```

考えられる原因:このエラーは、オプションのテストリスナーを設定しており、そのリスナーに間違ったターゲットグループが設定されている場合に発生する可能性があります。CodeDeploy のテストリスナーの詳細については、「Amazon ECS デプロイを開始する前に」と「Amazon ECS デプロイ中の処理で起こっていること」を参照してください。タスクセットの詳細については、「Amazon Elastic Container Service API リファレンス」の「TaskSet」と、「AWS CLI コマンドリファレンス」の「Amazon ECS」セクションの「describe-task-set」を参照してください。

考えられる解決方法: Elastic Load Balancing の本稼働リスナーとテストリスナーの両方が、現在ワークロードを処理しているターゲットグループを指していることを確認します。確認すべき箇所は 3 つあります。

- Amazon EC2 のロードバランサーの「リスナーとルール」の設定。詳細については、「Application Load Balancer のユーザーガイド」の「Application Load Balancer のリスナー」、または「Network Load Balancer のユーザーガイド」の「Network Load Balancer のリスナー」を参照してください。
- Amazon ECS のクラスター内のサービスのネットワーク設定。詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「Application Load Balancerおよび Network Load Balancerの場合」を参照してください。
- CodeDeploy のデプロイグループ設定。詳細については、「Amazon ECS デプロイ用のデプロイグループを作成する (コンソール)」を参照してください。

Auto Scaling を使用するとデプロイが失敗することがあります

問題: CodeDeploy で Auto Scaling を使用中、デプロイが失敗することがあることに気付きました。この問題の症状の詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「サービスの自動スケーリングと ブルー/グリーンデプロイタイプを使用するように設定されたサービスでは、自動スケーリングはデプロイ中にブロックされませんが、状況によってはデプロイが失敗する場合があります」というトピックを参照してください。

考えられる原因: この問題は、CodeDeploy プロセスと Auto Scaling プロセスが競合する場合に発生する可能性があります。

解決方法: `RegisterScalableTarget` API (または対応する`register-scalable-target` AWS CLI コマンド) を使用して CodeDeploy デプロイ中に Auto Scaling プロセスを一時停止および再開し

ます。詳細については、「Application Auto Scaling ユーザーガイド」の「Application Auto Scaling のスケーリングの一時停止と再開」を参照してください。

> **Note**
>
> CodeDeploy は `RegisterScaleableTarget` を直接呼び出すことはできません。この API を使用するには、Amazon Simple Notification Service (または Amazon CloudWatch) に通知またはイベントを送信するように CodeDeploy を設定する必要があります。次に、Lambda 関数を呼び出すように Amazon SNS (または CloudWatch) を設定し、`RegisterScalableTarget` API を呼び出すように Lambda 関数を設定する必要があります。Auto Scaling オペレーションを一時停止するには `SuspendedState` パラメータを`true` に設定し、再開するには `false` に設定し、`RegisterScalableTarget` API を呼び出す必要があります。
>
> CodeDeploy が送信する通知またはイベントは、デプロイが開始したとき (Auto Scaling の一時停止オペレーションをトリガーする場合)、またはデプロイが成功したとき、失敗したとき、または停止したとき (Auto Scaling 再開オペレーションをトリガーする場合) に発生する必要があります。
>
> Amazon SNS 通知または CloudWatch イベントを生成するように CodeDeploy を設定する方法については、「Amazon CloudWatch Events を使用したデプロイのモニタリング」と「Monitoring Deployments with Amazon SNS Event Notifications」を参照してください。

ALB のみが段階的なトラフィックルーティングをサポートしているため、デプロイグループの作成/更新時には、代わりに AllAtOnce トラフィックルーティングを使用してください

問題: CodeDeploy でデプロイグループを作成または更新している際に、次のエラーメッセージが表示されます。

```
Only ALB supports gradual traffic routing, use AllAtOnce Traffic routing instead when you create/update Deployment group.
```

考えられる原因: このエラーは Network Load Balancer を使用していて、`CodeDeployDefault.ECSAllAtOnce` 以外の事前定義されたデプロイ設定を使用しようとした場合に発生する可能性があります。

解決方法:

- 事前定義されたデプロイ設定を CodeDeployDefault.ECSAllAtOnce に変更します。これは Network Load Balancer がサポートする唯一の事前定義されたデプロイ設定です。

  事前定義されたデプロイ設定の詳細については、「Amazon ECS コンピューティングプラットフォームの事前定義されたデプロイ設定」を参照してください。

- ロードバランサーを Application Load Balancer に変更します。Application Load Balancer は、事前定義されたデプロイ設定をすべてサポートします。Application Load Balancer の作成の詳細については、「CodeDeploy Amazon ECS デプロイ用のロードバランサー、ターゲットグループ、リスナーをセットアップする」を参照してください。

デプロイが成功しても、置き換えタスクセットは Elastic Load Balancing のヘルスチェックに失敗し、アプリケーションがダウンしています

問題: CodeDeploy ではデプロイが成功したと表示されているのに、置き換えタスクセットは Elastic Load Balancing からのヘルスチェックに失敗し、アプリケーションがダウンしています。

考えられる原因: この問題は、CodeDeploy 一括デプロイを実行した際に、置き換え (グリーン) タスクセットに Elastic Load Balancing ヘルスチェックが失敗する原因となる誤ったコードが含まれている場合に発生する可能性があります。all-at-onceデプロイ設定では、トラフィックが移行された後 (つまり、CodeDeploy のAllowTrafficライフサイクルイベントが発生した後)、ロードバランサーのヘルスチェックが置き換えタスクセットで実行を開始します。そのため、トラフィックがシフトした後、置き換えタスクセットでヘルスチェックが失敗しますが、以前は失敗しません。CodeDeploy が生成するライフサイクルイベントについては、「Amazon ECS デプロイ中の処理で起こっていること」を参照してください。

解決方法:

- デプロイ設定を 1 回にすべてから canary またはリニアに変更します。Canary または線形設定では、CodeDeploy がアプリケーションを代替環境にインストールしている間、およびトラフィックが移行される前 (つまり、Installライフサイクルイベント中およびAllowTrafficイベント前) に、ロードバランサーのヘルスチェックが代替タスクセットで実行を開始します。アプリケーションのインストール中、トラフィックが移行する前にチェックを実行できるようにすることで、アプリケーションが一般公開される前に誤ったアプリケーションコードが検出され、デプロイが失敗します。

  canary デプロイまたはリニアデプロイを設定する方法については、「CodeDeploy を使用して、デプロイグループの設定を変更します。」を参照してください。

Amazon ECS のデプロイ中に実行される CodeDeploy ライフサイクルイベントの詳細については、「Amazon ECS デプロイ中の処理で起こっていること」を参照してください。

> **Note**
>
> canary デプロイ設定およびリニアデプロイ設定は、Application Load Balancer でのみサポートされます。

- 1 回にすべてデプロイ設定を維持したい場合は、テストリスナーをセットアップし、`BeforeAllowTraffic` ライフサイクルフックを使用して置き換えタスクセットのヘルスステータスを確認します。詳細については、「Amazon ECS のデプロイ向けのライフサイクルイベントフックのリスト」を参照してください。

1 つのデプロイグループに複数のロードバランサーをアタッチできますか?

いいえ。複数の Application Load Balancer または Network Load Balancer を使用する場合は、CodeDeploy ブルー/グリーンデプロイの代わりに Amazon ECS ローリング更新を使用してください。ローリング更新の詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「ローリング更新」を参照してください。Amazon ECS で複数のロードバランサーを使用する方法の詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「サービスに複数のターゲットグループを登録する」を参照してください。

ロードバランサーなしで CodeDeploy のブルー/グリーンデプロイを実行できますか?

いいえ、ロードバランサーなしで CodeDeploy のブルー/グリーンデプロイを実行することはできません。ロードバランサーを使用できない場合は、代わりに Amazon ECS のローリング更新機能を使用してください。Amazon ECS のローリング更新機能の詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「ローリング更新」を参照してください。

デプロイ中に Amazon ECS サービスを新しい情報で更新する方法を教えてください。

CodeDeploy がデプロイの実行中に Amazon ECS サービスを新しいパラメータで更新するようにするには、AppSpec ファイルの `resources` セクションでパラメータを指定します。CodeDeploy でサポートされている Amazon ECS パラメータは、タスク定義ファイルやコンテナ名パラメー

タなど、ごく一部です。CodeDeploy が更新できる Amazon ECS パラメータの一覧については、「Amazon ECS デプロイ用の AppSpec の「resources」セクション」を参照してください。

> **Note**
>
> CodeDeploy でサポートされていないパラメータで Amazon ECS サービスを更新する必要がある場合は、以下のタスクを実行します。
>
> 1. 更新したいパラメータを指定して Amazon ECS の `UpdateService` API を呼び出します。更新できるパラメータの一覧については、「Amazon Elastic Container Service API リファレンス」の「UpdateService」を参照してください。
> 2. 変更をタスクに適用するには、新しい Amazon ECS ブルー/グリーンデプロイを作成します。詳細については、「Amazon ECS コンピューティングプラットフォームのデプロイの作成 (コンソール)」を参照してください。

AWS Lambda デプロイの問題をトラブルシューティングする

トピック

- AWS Lambda ロールバックが設定されていない Lambda デプロイを手動で停止すると、デプロイが失敗する

AWS Lambda ロールバックが設定されていない Lambda デプロイを手動で停止すると、デプロイが失敗する

場合によっては、デプロイで指定された Lambda 関数のエイリアスで、2 つの異なる関数バージョンが参照されることがあります。そのため、Lambda 関数をデプロイしようとすると失敗します。Lambda デプロイでは、ロールバックが設定されていない場合に手動で停止すると、この状態になることがあります。続行するには、 AWS Lambda コンソールを使用して、 関数が 2 つのバージョン間でトラフィックをシフトするように設定されていないことを確認します。

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/lambda/ で AWS Lambda コンソールを開きます。
2. 左側のペインで、[関数] を選択します。
3. CodeDeploy デプロイ内にある Lambda 関数の名前を選択します。

4. 「エイリアス」から、CodeDeploy デプロイで使用されているエイリアスを選択し、「編集」を選択します。
5. [加重エイリアス] から [**none**] を選択します。これにより、このエイリアスで、トラフィックの割合やウェイトが 2 つ以上のバージョンに移行されることはありません。[バージョン] で選択されているバージョンを書き留めます。
6. [保存] を選択します。
7. CodeDeploy コンソールを開き、ステップ 5 のドロップダウンメニューで表示されているバージョンをデプロイします。

デプロイグループの問題のトラブルシューティング

デプロイグループの一部としてインスタンスにタグを付けても、アプリケーションが自動的に新しいインスタンスにデプロイされない

CodeDeploy は新しくタグが付けられたインスタンスに自動的にアプリケーションをデプロイしません。デプロイグループで新しいデプロイを作成する必要があります。

CodeDeploy を使用すると、Amazon EC2 Auto Scaling グループの新しい EC2 インスタンスへの自動デプロイを有効にすることができます。詳細については、「CodeDeploy と Amazon EC2 Auto Scaling の統合」を参照してください。

インスタンスの問題のトラブルシューティング

トピック

- タグは正しく設定する必要がある
- AWS CodeDeploy エージェントはインスタンスにインストールして実行する必要があります
- デプロイ中にインスタンスを削除した場合、デプロイは最大 1 時間は失敗しません。
- ログファイルの分析によるインスタンスでのデプロイの失敗の調査
- 誤って削除した場合は、新しい CodeDeploy ログファイルを作成します。
- InvalidSignatureException – Signature expired: [time] is now earlier than [time]」デプロイエラーのトラブルシューティング

タグは正しく設定する必要がある

デプロイに使用されるインスタンスに正しくタグが付けられていることを確認するには、 list-deployment-instances コマンドを使用します。出力に EC2 インスタンスがない場合は、EC2 コンソールを使用して、インスタンスにタグが設定されていることを確認します。詳細については、「Amazon EC2 ユーザーガイド」の「コンソールでのタグの操作」を参照してください。

> **Note**
>
> インスタンスにタグを付け、直後に CodeDeploy を使用してそのインスタンスにアプリケーションをデプロイする場合、インスタンスはデプロイに含まれないこともあります。これは、CodeDeploy がタグを読み込むまでに数分かかることがあるためです。インスタンスにタグを付けてから、そのインスタンスへのデプロイを試みるまでに、少なくとも 5 分待つことをお勧めします。

AWS CodeDeploy エージェントはインスタンスにインストールして実行する必要があります

インスタンスに CodeDeploy エージェントがインストールされて実行されていることを確認する方法については、「CodeDeploy エージェントが実行されていることの確認」を参照してください。

CodeDeploy エージェントをインストール、アンインストール、または再インストールする方法については、「CodeDeploy エージェントをインストールする」を参照してください。

デプロイ中にインスタンスを削除した場合、デプロイは最大 1 時間は失敗しません。

CodeDeploy は、各デプロイライフサイクルイベントの実行が完了するまでに 1 時間の枠を設定しています。これにより、実行時間が長いスクリプトにも十分な時間が提供されます。

ライフサイクルイベントの進行中に、スクリプト実行が完了しない場合 (たとえば、インスタンスが終了された、CodeDeploy エージェントがシャットダウンされたなど)、デプロイのステータスが Failed になるまで、最大で 1 時間かかることがあります。スクリプトに指定されたタイムアウト時間が 1 時間未満である場合も同様です。これは、インスタンスが終了すると CodeDeploy エージェントがシャットダウンし、それ以上のスクリプトを処理できなくなるためです。

インスタンスがライフサイクル間、または最初のライフサイクルイベントのステップが開始する前に削除された場合、タイムアウトはわずか 5 分後に発生します。

ログファイルの分析によるインスタンスでのデプロイの失敗の調査

デプロイのインスタンスのステータスが Succeeded 以外のいずれかである場合は、デプロイのログファイルデータを確認すると、問題の特定に役立ちます。デプロイのログデータへのアクセス方法については、「CodeDeploy EC2/オンプレミスデプロイのログデータの表示」を参照してください。

誤って削除した場合は、新しい CodeDeploy ログファイルを作成します。

誤ってインスタンスのデプロイログファイルを削除した場合、CodeDeploy は代わりのログファイルを作成しません。新しいログファイルを作成するには、インスタンスにサインインし、以下のコマンドを実行します。

Amazon Linux、Ubuntu Server、または RHEL インスタンスの場合、以下のコマンドをこの順序で 1 つずつ実行します。

```
systemctl stop codedeploy-agent
```

```
systemctl start codedeploy-agent
```

Windows Server インスタンスの場合

```
powershell.exe -Command Restart-Service -Name codedeployagent
```

InvalidSignatureException – Signature expired: [time] is now earlier than [time]」デプロイエラーのトラブルシューティング

CodeDeploy では、オペレーションを実行するために正確な時間の参照が必要です。インスタンスの日時が正しく設定されていない場合は、デプロイリクエストの署名と一致しないことがあり、その場合は CodeDeploy によってリクエストが却下されます。

誤った時間設定に関連するデプロイの失敗を回避する方法については、次のトピックを参照してください。

- Linux インスタンスの時刻の設定
- Windows インスタンスの時刻を設定する

GitHub トークンの問題のトラブルシューティング

GitHub OAuth トークンが無効です

2017 年 6 月以降に作成された CodeDeploy アプリケーションでは、各 AWS リージョンの GitHub OAuth トークンを使用します。特定の AWS リージョンに関連付けられたトークンを使用すると、GitHub リポジトリにアクセスできる CodeDeploy アプリケーションをより詳細に制御できます。

GitHub トークンエラーが発生した場合は、すでに無効になっている古いトークンの場合があります。

無効な GitHub OAuth トークンを修正するには

1. 以下のいずれかの方法を使用して、古いトークンを削除してください。
   - API を使って古いトークンを削除するには、DeleteGitHubAccountToken を使用します。
   - AWS Command Line Interfaceを使用して古いトークンを削除するには:
     a. トークンが存在するコンピュータに移動します。
     b. AWS CLI がこのコンピュータにインストールされていることを確認します。インストール方法については、AWS CLIユーザーガイド の「AWS Command Line Interface のインストール、更新、およびアンインストール」を参照してください。
     c. トークンが存在するコンピュータで、次のコマンドを入力します。

        ```
        aws delete-git-hub-account-token
        ```

        コマンド構文の詳細については、「delete-git-hub-account-token」を参照してください。
2. 新しい OAuth トークンを追加します。詳細については、「GitHub と CodeDeploy との統合」を参照してください。

GitHub OAuth トークンの最大数を超えました

CodeDeploy のデプロイを作成する際に許可されている GitHub トークンの最大数は 10 です。GitHub OAuth トークンに関するエラーが発生した場合、トークンが 10 個以下であることを確認します。トークンが 10 個以上ある場合は、最初に作成されたトークンが無効になります。たとえば、トークンが 11 個ある場合、最初に作成したトークンが無効になります。トークンが 12 個ある

場合、最初に作成した 2 個のトークンが無効になります。CodeDeploy API を使用して古いトークンを削除する方法については、「DeleteGitHubAccountToken」を参照してください。

Amazon EC2 Auto Scaling の問題のトラブルシューティング

トピック

- Amazon EC2 Auto Scaling の一般的なトラブルシューティング
- CodeDeployRole は、次の AWS サービスでオペレーションを実行するアクセス許可を付与しません: AmazonAutoScaling」エラー
- Amazon EC2 Auto Scaling グループのインスタンスのプロビジョニングと終了が繰り返されてリビジョンをデプロイできない
- Amazon EC2 Auto Scaling インスタンスを終了または再起動すると、デプロイが失敗する場合がある
- 複数のデプロイグループを 1 つの Amazon EC2 Auto Scaling グループに関連付けることは避ける
- Amazon EC2 Auto Scaling グループの EC2 インスタンスが起動に失敗し、「ハートビートのタイムアウト」というエラーが表示される
- Amazon EC2 Auto Scaling ライフサイクルフックの不一致により、Amazon EC2 Auto Scaling グループへの自動デプロイが停止または失敗することがある
- 「デプロイグループのインスタンスが見つからないため、デプロイに失敗しました」というエラー

Amazon EC2 Auto Scaling の一般的なトラブルシューティング

Amazon EC2 Auto Scaling グループの EC2 インスタンスへのデプロイは、次の理由で失敗する場合があります。

- Amazon EC2 Auto Scaling は、EC2 インスタンスを継続的に起動および終了します。CodeDeploy がアプリケーションリビジョンを自動的にデプロイできない場合、Amazon EC2 Auto Scaling は継続的に EC2 インスタンスを起動および終了させます。

  CodeDeploy デプロイグループから Amazon EC2 Auto Scaling グループの関連付けを解除するか、Amazon EC2 Auto Scaling グループの設定を変更し、必要なインスタンス数が現在のインスタンス数に一致するようにします (これにより、Amazon EC2 Auto Scaling がそれ以上の EC2 インスタンスを起動できないようにします)。詳細については、「CodeDeploy を使用して、デプロイグループの設定を変更します。」または「Amazon EC2 Auto Scaling のマニュアルスケーリング」を参照してください。

- CodeDeploy エージェントが応答しません。EC2 インスタンスの起動直後に実行される初期化スクリプト (cloud-init スクリプトなど) の実行に 1 時間以上かかる場合、CodeDeploy エージェントがインストールされないことがあります。CodeDeploy には、CodeDeploy エージェントが保留中のデプロイに応答するための 1 時間のタイムアウトがあります。この問題に対応するには、CodeDeploy アプリケーションリビジョンに初期化スクリプトを移動します。
- Amazon EC2 Auto Scaling グループの EC2 インスタンスは、デプロイ中に再起動します。デプロイ中に EC2 インスタンスが再起動したか、デプロイコマンドの処理中に CodeDeploy エージェントがシャットダウンした場合、デプロイは失敗することがあります。詳細については、「Amazon EC2 Auto Scaling インスタンスを終了または再起動すると、デプロイが失敗する場合がある」を参照してください。
- 複数のアプリケーションリビジョンが Amazon EC2 Auto Scaling グループの同じ EC2 インスタンスに同時にデプロイされた。Amazon EC2 Auto Scaling グループの同じ EC2 インスタンスに複数のアプリケーションリビジョンをデプロイする場合、デプロイの 1 つに数分以上実行されるスクリプトがあると、失敗することがあります。Amazon EC2 Auto Scaling グループの同じ EC2 インスタンスに複数のアプリケーションリビジョンをデプロイしないでください。
- Amazon EC2 Auto Scaling グループの一部として起動された新しい EC2 インスタンスに対して、デプロイが失敗する。このシナリオでは、デプロイでスクリプトを実行すると、Amazon EC2 Auto Scaling グループで EC2 インスタンスを起動できなくなる場合があります。(Amazon EC2 Auto Scaling グループの他の EC2 インスタンスは、正常に実行されているように見える場合があります)。この問題に対応するには、最初にその他のすべてのスクリプトが完了していることを確認します。
  - CodeDeploy エージェントは AMI に含まれていません。新しいインスタンスの起動中に cfn-init コマンドを使用して CodeDeploy エージェントをインストールする場合は、 CloudFormation テンプレートの `cfn-init`セクションの最後にエージェントインストールスクリプトを配置します。
  - CodeDeploy エージェントを AMI に含める場合: インスタンスの作成時にエージェントが `Stopped` 状態になるように設定し、次に `cfn-init` スクリプトライブラリの最終ステップとして、エージェントを起動するスクリプトを含めます

CodeDeployRole は、次の AWS サービスでオペレーションを実行するアクセス許可を付与しません: AmazonAutoScaling」エラー

起動テンプレートを使用して作成された Auto Scaling グループを使用するデプロイでは、次のアクセス権限が必要です。これらは、 `AWSCodeDeployRole` AWS 管理ポリシーによって付与されたアクセス許可に追加されます。

- EC2:RunInstances
- EC2:CreateTags
- iam:PassRole

これらのアクセス許可がない場合、このエラーが表示されることがあります。詳細については、「Amazon EC2 Auto Scaling ユーザーガイド」の「チュートリアル: CodeDeploy を使用して、Auto Scaling グループにアプリケーションをデプロイします。」、「Auto Scaling グループの起動テンプレートの作成」、および「アクセス許可」を参照してください。

Amazon EC2 Auto Scaling グループのインスタンスのプロビジョニングと終了が繰り返されてリビジョンをデプロイできない

エラーが原因で、Amazon EC2 Auto Scaling グループ内の新しくプロビジョニングされたインスタンスに正常にデプロイできない場合があります。その結果として、インスタンスは正常な状態にならず、デプロイは失敗します。デプロイが正常に実行または完了されないため、インスタンスは作成後すぐに終了されます。この場合、Amazon EC2 Auto Scaling グループの設定により、正常なホスト数の最小要件を満たすために別のバッチのインスタンスがプロビジョニングされます。このバッチも終了され、同じサイクルが繰り返されます。

エラーの原因として以下が考えられます。

- Amazon EC2 Auto Scaling グループのヘルスチェックに失敗しました。
- アプリケーションリビジョンのエラー。

この問題を回避するには、次の手順に従います。

1. Amazon EC2 Auto Scaling グループに属していない EC2 インスタンスを手動で作成します。インスタンスに一意の EC2 インスタンスタグを付けます。
2. 新しいインスタンスを該当するデプロイグループに追加します。
3. 新しい、エラーのないアプリケーションリビジョンをデプロイグループにデプロイします。

これにより、Amazon EC2 Auto Scaling グループの今後のインスタンスにアプリケーションリビジョンがデプロイされるようになります。

> **Note**
>
> デプロイが成功したことを確認したら、 AWS アカウントへの継続的な料金が発生しないように、作成したインスタンスを削除します。

Amazon EC2 Auto Scaling インスタンスを終了または再起動すると、デプロイが失敗する場合がある

EC2 インスタンスが Amazon EC2 Auto Scaling を通じて起動され、その後で終了または再起動されると、そのインスタンスへのデプロイは、次の理由で失敗する場合があります。

- 進行中のデプロイで、スケールインイベントまたはその他の終了イベントにより、インスタンスは Amazon EC2 Auto Scaling グループからデタッチされた後に削除されます。デプロイは完了できないため、失敗します。
- インスタンスが再起動されたが、その起動に 5 分間以上かかる。CodeDeploy はこれをタイムアウトとして扱います。サービスにより、インスタンスに対する現在およびそれ以降のすべてのデプロイが失敗します。

この問題に対応するには:

- 一般的に、インスタンスが削除または再起動される前に、すべてのデプロイが完了したことを確認します。インスタンスの起動または再起動後に、すべてのデプロイが開始されることを確認します。
- Amazon EC2 Auto Scaling の設定に Windows Server ベースの Amazon Machine Image (AMI) を指定し、EC2Config サービスを使用して、インスタンスのコンピュータ名を設定すると、デプロイは失敗します。この問題を解決するには、Windows Server ベース AMI で、[EC2 Service Properties] (EC2 サービスプロパティ) の [General] (全般) タブにある [Set Computer Name] (コンピュータ名の設定) をオフにします。このチェックボックスをオフにすると、この動作は、その Windows Server の基本 AMI で起動されるすべての新しい Windows Server Amazon EC2 Auto Scaling インスタンスで、この動作が無効になります。この動作が有効になっている Windows Server Amazon EC2 Auto Scaling インスタンスでは、このチェックボックスをオフにする必要はありません。再起動後に、失敗したデプロイをこれらのインスタンスに再デプロイします。

複数のデプロイグループを 1 つの Amazon EC2 Auto Scaling グループに関連付けることは避ける

各 Amazon EC2 Auto Scaling グループには 1 つのデプロイグループのみを関連付けることをお勧めします。

これは、複数のデプロイグループに関連付けられたフックを持つインスタンスを Amazon EC2 Auto Scaling がスケールアップする場合、すべてのフックに対して一度に通知を送信するためです。これにより、各インスタンスの複数のデプロイが同時に開始されます。複数のデプロイが同時に CodeDeploy エージェントへコマンドを送信すると、ライフサイクルイベントとデプロイの開始または前のライフサイクルイベント終了の間に 5 分間のタイムアウトに達する場合があります。その場合は、予想通りにデプロイがプロセスされていてもデプロイが失敗します。

> **Note**
>
> ライフサイクルイベント内にあるスクリプトのデフォルトのタイムアウトは、デフォルトで 30 分です。AppSpec ファイル内のタイムアウトを別の値に変更できます。詳細については、「EC2/オンプレミスデプロイに AppSpec ファイルを追加する」を参照してください。

複数のデプロイが同時に実行を試みた場合、デプロイが発生する順序を制御することはできない。

最後に、インスタンスへのデプロイが失敗した場合、Amazon EC2 Auto Scaling は直ちにインスタンスを終了します。その最初のインスタンスがシャットダウンすると、実行中の他のデプロイも失敗します。CodeDeploy には、CodeDeploy エージェントが保留中のデプロイに応答するための 1 時間のタイムアウトがあるため、各インスタンスのタイムアウトは最大 60 分かかる場合があります。

Amazon EC2 Auto Scaling の詳細については、「内部構造: CodeDeploy と Auto Scaling の統合」を参照してください。

Amazon EC2 Auto Scaling グループの EC2 インスタンスが起動に失敗し、「ハートビートのタイムアウト」というエラーが表示される

Amazon EC2 Auto Scaling グループが新しい EC2 インスタンスの起動に失敗し、次のようなメッセージを生成する場合があります。

```
Launching a new EC2 instance <instance-Id>. Status Reason: Instance failed
to complete user's Lifecycle Action: Lifecycle Action with token<token-Id>
was abandoned: Heartbeat Timeout.
```

このメッセージは通常、以下のいずれかを示します。

- AWS アカウントに関連付けられた同時デプロイの最大数に達しました。デプロイの制限の詳細については、「CodeDeploy のクォータ」を参照してください。
- Auto Scaling グループは、あまりにも多くの EC2 インスタンスを早く起動しようとしました。新しいインスタンスごとに RecordLifecycleActionHeartbeat または CompleteLifecycleAction への API コールがスロットルされました。
- 関連付けられたデプロイグループが更新または削除される前に、CodeDeploy のアプリケーションが削除された。

  アプリケーションまたはデプロイグループを削除すると、CodeDeploy はそれに関連付けられていた Amazon EC2 Auto Scaling フックのクリーンアップを試みますが、一部のフックが残る場合があります。デプロイグループを削除するコマンドを実行すると、残りのフックが出力で返ります。ただし、アプリケーションを削除するコマンドを実行した場合、残りのフックは出力に表示されません。

  したがって、アプリケーションを削除する前に、アプリケーションと関連付けられたすべてのデプロイグループを削除することをお勧めします。コマンド出力を使用して、手動で削除する必要があるライフサイクルフックを識別できます。

「ハートビートタイムアウト」エラーメッセージが表示された場合は、以下を実行して、残りのライフサイクルフックが原因かどうかを判断し、問題を解決できます。

1. 次のいずれかを行います。

   - delete-deployment-group コマンドを呼び出して、ハートビートタイムアウトの原因となっている Auto Scaling グループに関連付けられたデプロイグループを削除します。
   - Auto Scaling グループ名の NULL ではない空のリストを指定して、update-deployment-group コマンドを呼び出し、CodeDeploy が管理する全ての Auto Scaling ライフサイクルフックをデタッチすることができます。

     たとえば、次のコマンドを入力します AWS CLI 。

     ```
     aws deploy update-deployment-group --application-name my-example-app
     --current-deployment-group-name my-deployment-group --auto-scaling-
     groups
     ```

別の例として、Java で CodeDeploy API を使用する場合は、`UpdateDeploymentGroup` を呼び出し、`autoScalingGroups` を `new ArrayList<String>()` に設定します。これにより、`autoScalingGroups` を空のリストに設定し、既存のリストを削除します。デフォルトの `null` を使用すると、`autoScalingGroups` がそのまま残ってしまうので使用しないでください。

呼び出しの出力を確認します。出力に `hooksNotCleanedUp` 構造と Amazon EC2 Auto Scaling ライフサイクルフックの一覧が含まれている場合、ライフサイクルフックの残りがあります。

2. describe-lifecycle-hooks コマンドを呼び出し、起動に失敗した EC2 インスタンスに関連付けられている Amazon EC2 Auto Scaling グループの名前を指定します。出力で、次のいずれかを確認します。

   - Amazon EC2 Auto Scaling ライフサイクルフック名で、ステップ 1 で特定した `hooksNotCleanedUp` 構造に対応するもの
   - Amazon EC2 Auto Scaling ライフサイクルフック名で、失敗している Auto Scaling グループに関連するデプロイグループの名前を含むもの
   - Amazon EC2 Auto Scaling のライフサイクルフック名で、CodeDeploy デプロイのハートビートタイムアウトの原因となった可能性のあるもの

3. フックがステップ 2 でリストされたカテゴリのいずれかに該当する場合は、delete-lifecycle-hook コマンドを呼び出して削除します。Amazon EC2 Auto Scaling グループとライフサイクルフックをコールで指定します。

   > **Important**
   >
   > ステップ 2 で説明したように、問題の原因となっているフックのみを削除してください。実行可能なフックを削除すると、デプロイが失敗したり、CodeDeploy がアプリケーションリビジョンをスケールアウトした EC2 インスタンスにデプロイできなくなったりする可能性があります。

4. update-deployment-group または create-deployment-group コマンドを、必要な Auto Scaling グループ名で呼び出します。CodeDeploy は、新しい UUID を持つ Auto Scaling フックを再インストールします。

> **Note**
>
> CodeDeploy デプロイグループから Auto Scaling グループをデタッチすると、Auto Scaling グループへの進行中のデプロイはすべて失敗し、Auto Scaling グループによってスケールアウトされる新しい EC2 インスタンスは CodeDeploy からアプリケーションリビジョンを受けとれなくなる可能性があります。CodeDeploy で Auto Scaling を再度動作させるには、Auto Scaling グループをデプロイグループに再アタッチし、新しい `CreateDeployment` を呼び出しして、フリート全体のデプロイを開始する必要があります。

Amazon EC2 Auto Scaling ライフサイクルフックの不一致により、Amazon EC2 Auto Scaling グループへの自動デプロイが停止または失敗することがある

Amazon EC2 Auto Scaling と CodeDeploy はライフサイクルフックを使用して、Amazon EC2 Auto Scaling グループで起動された後にデプロイするアプリケーションリビジョンと、そのデプロイ先の EC2 インスタンスを判断します。ライフサイクルフックとそれに関する情報が Amazon EC2 Auto Scaling と CodeDeploy で正確に一致しない場合、自動デプロイは停止または失敗することがあります。

Amazon EC2 Auto Scaling グループへのデプロイに失敗する場合、Amazon EC2 Auto Scaling および CodeDeploy のライフサイクルフック名が一致するかどうかを確認します。そうでない場合は、これらの AWS CLI コマンド呼び出しを使用します。

最初に、Amazon EC2 Auto Scaling グループとデプロイグループの両方のライフサイクルフック名の一覧を取得します。

1. describe-lifecycle-hooks コマンドを呼び出し、CodeDeploy のデプロイグループに関連付けられた Amazon EC2 Auto Scaling グループの名前を指定します。出力の `LifecycleHooks` リストで、`LifecycleHookName` のそれぞれの値を書き留めます。
2. get-deployment-group コマンドを呼び出し、Amazon EC2 Auto Scaling グループに関連付けられたデプロイグループの名前を指定します。出力の `autoScalingGroups` リストで、名前の値が Amazon EC2 Auto Scaling グループ名と一致する項目を見つけ、対応する hook の値を書き留めます。

ここで、2 セットのライフサイクルフック名を比較します。それらが 1 文字ずつ正確に一致する場合は、これが問題ではありません。このセクションの他の場所で説明されている Amazon EC2 Auto Scaling の他のトラブルシューティングステップを試してください。

ただし、2 セットのライフサイクルフック名が 1 文字ずつ正確に一致しない場合は、次の操作を行います。

1. get-deployment-group コマンド出力にもないライフサイクルフック名が describe-lifecycle-hooks コマンド出力にある場合は、次の操作を行います。

   a. describe-lifecycle-hooks コマンド出力のライフサイクルフック名ごとに、delete-lifecycle-hook コマンドを呼び出します。

   b. update-deployment-group コマンドを呼び出し、元の Amazon EC2 Auto Scaling のグループ名を指定します。CodeDeploy は、Amazon EC2 Auto Scaling グループに新しい代替のライフサイクルフックを作成し、そのライフサイクルフックをデプロイグループに関連付けます。これで、Amazon EC2 Auto Scaling グループに新しいインスタンスを追加すると、自動デプロイが開始されます。

2. describe-lifecycle-hooks コマンド出力にもないライフサイクルフック名が get-deployment-group コマンド出力にある場合は、次の操作を行います。

   a. update-deployment-group コマンドを呼び出し、元の Amazon EC2 Auto Scaling のグループ名を指定しないようにします。

   b. update-deployment-group コマンドを再度呼び出し、今度は元の Amazon EC2 Auto Scaling グループの名前を指定します。CodeDeploy は、Amazon EC2 Auto Scaling グループに不足しているライフサイクルフックを再作成します。これで、Amazon EC2 Auto Scaling グループに新しいインスタンスを追加すると、自動デプロイが開始されます。

1 文字ごとに正確に一致する 2 セットのライフサイクルフック名を取得したら、アプリケーションリビジョンをもう一度デプロイしますが、Amazon EC2 Auto Scaling グループに追加された新しいインスタンスにのみデプロイします。Amazon EC2 Auto Scaling グループに既に存在するインスタンスに対しては、デプロイは自動的に行われません。

「デプロイグループのインスタンスが見つからないため、デプロイに失敗しました」というエラー

以下のような CodeDeploy エラーが発生した場合は、このセクションをお読みください。

```
The deployment failed because no instances were found for your deployment
group. Check your deployment group settings to make sure the tags for your
EC2 instances or Auto Scaling groups correctly identify the instances you
want to deploy to, and then try again.
```

このエラーの原因として考えられるのは、以下の通りです。

1. デプロイグループの設定には、正しくない Auto Scaling グループ、EC2 インスタンス、またはオンプレミスインスタンスのタグが含まれています。この問題を解決するには、タグが正しいことをチェックしてから、アプリケーションを再デプロイしてください。
2. デプロイ開始後、フリートがスケールアウトしました。このシナリオでは、フリートで `InService` 状態の正常なインスタンスが表示されますが、上記のエラーも表示されます。この問題を解決するには、アプリケーションを再デプロイします。
3. Auto Scaling グループには、`InService` 状態のインスタンスは含まれません。このシナリオでは、CodeDeploy は少なくとも 1 つのインスタンスが `InService` 状態である必要があるため、フリート全体のデプロイを実行しようとすると、上記のエラーメッセージが表示されてデプロイが失敗します。`InService` 状態でインスタンスがない場合、様々な理由が考えられます。その一部を紹介します。
   - Auto Scaling グループのサイズを `0` にスケジュール (または手動で設定) しました。
   - Auto Scaling は、不正な EC2 インスタンスを検出したため (例えば、EC2 インスタンスにハードウェア障害が発生した)、すべてキャンセルし、`InService` 状態には何も残さないようにしました。
   - `0` から `1` へのスケールアウトイベントで、CodeDeploy は前回成功したリビジョン (最後に成功したリビジョン) をデプロイしましたが、前回のデプロイから正常ではない状態に陥っていました。これにより、スケールアウトしたインスタンスのデプロイが失敗し、そのため、Auto Scaling がインスタンスをキャンセルし、`InService` 状態のインスタンスが一つも残らないという事態が発生しました。

     `InService` 状態のインスタンスがないことがわかった場合、次の手順 To troubleshoot the error if there are no instances in the InService state の説明に従ってトラブルシューティングします。

InService 状態のインスタンスが存在しない場合のエラーのトラブルシューティングについて

1. Amazon EC2 コンソールで、[希望する容量] の設定を確認します。ゼロの場合は、正の数に設定します。インスタンスが `InService` になるのを待ちます。これは、デプロイが成功したことを

意味します。問題が解決されたので、このトラブルシューティングの手順の残りのステップはスキップできます。[Desired Capacity] (希望する容量) の設定については、「Amazon EC2 Auto Scaling ユーザーガイド」の「Auto Scaling グループの容量制限を設定する」を参照してください。

2. Auto Scaling が希望する容量を満たすために新しい EC2 インスタンスを起動し続けようとするが、スケールアウトを実行できない場合は、通常、Auto Scaling のライフサイクルフックが失敗していることが原因です。この問題については、次のようにトラブルシューティングします。

   a. 失敗している Auto Scaling ライフサイクルフックイベントを確認するには、Amazon EC2 Auto Scaling ユーザーガイドの「Auto Scaling グループのスケーリングアクティビティの確認」を参照してください。

   b. 失敗したフックの名前が `CodeDeploy-managed-automatic-launch-deployment-hook-`*`DEPLOYMENT_GROUP_NAME`* の場合、CodeDeploy に移動し、デプロイグループを見つけて、Auto Scaling によって開始され、失敗したデプロイを見つけます。次に、デプロイが失敗した理由を調べます。

   c. デプロイが失敗した理由 (例えば、CloudWatch アラームが発生していたなど) がわかっていて、リビジョンを変更せずに問題を修正できる場合は、今すぐ実行してください。

   d. 調査後、CodeDeploy の最後に成功したリビジョンが正常でなくなり、Auto Scaling グループに正常なインスタンスがゼロであると判断した場合、デプロイのデッドロックシナリオになります。この問題を解決するには、CodeDeploy のライフサイクルフックを Auto Scaling グループから一時的に削除し、フックを再インストールして新しい (正常な) リビジョンを再デプロイすることで、不正な CodeDeploy リビジョンを修正する必要があります。手順については、こちらを参照してください。

      - To fix the deployment deadlock issue (CLI)
      - To fix the deployment deadlock issue (console)

デプロイのデッドロックの問題を解決するには (CLI)

1. (オプション) CodeDeploy エラーの原因となっている CI/CD パイプラインをブロックして、この問題の修正中に予期しないデプロイが発生しないようにします。
2. Auto Scaling の現在の [DesiredCapacity] 設定を書き留めます:

   ```
   aws autoscaling describe-auto-scaling-groups --auto-scaling-group-name ASG_NAME
   ```

   場合によっては、この手順の最後に、この数値にスケールバックする必要があります。

3. Auto Scaling の [DesiredCapacity] 設定を 1 にしてください。開始の際に希望する容量が 1 より大きい場合、オプションとなります。それを 1 に減らすことで、インスタンスのプロビジョニングとデプロイにかかる時間が短くなり、トラブルシューティングが高速化されます。Auto Scaling の希望する容量がもともと 0 に設定されている場合、1 に増やす必要があります。これは必須です。

   aws autoscaling set-desired-capacity — auto-scaling-group-name *ASG_NAME* —desired-capacity 1

   > **Note**
   >
   > この手順の残りのステップでは、[DesiredCapacity] を 1 に 設定したものとします。

   この時点で、Auto Scaling は 1 つのインスタンスへのスケーリングを試みます。次に、CodeDeploy が追加したフックがまだ存在するため、CodeDeploy がデプロイを試みても失敗し、Auto Scaling はインスタンスをキャンセルして希望容量の 1 に達するようにインスタンスを再起動しようとしますが、これも失敗します。キャンセル再起動ループに入っています。

4. デプロイグループから Auto Scaling グループの登録を解除します。

   > **Warning**
   >
   > 次のコマンドは、ソフトウェアなしで新しい EC2 インスタンスを起動します。コマンドを実行する前に、ソフトウェアを実行していない Auto Scaling `InService` インスタンスが許容されることを確認します。例えば、インスタンスに関連付けられているロードバランサーがソフトウェアなしでこのホストにトラフィックを送信していないことを確認してください。

   > **Important**
   >
   > 以下に示す CodeDeploy コマンドを使用して、フックを削除します。Auto Scaling サービスによるフック削除は、CodeDeploy で認識されないため、削除しないでください。

```
aws deploy update-deployment-group --application-name APPLICATION_NAME
--current-deployment-group-name DEPLOYMENT_GROUP_NAME --auto-scaling-
groups
```

このコマンドを実行すると、次の処理が実行されます。

a. CodeDeployは、デプロイグループから Auto Scaling グループを登録解除します。

b. CodeDeploy は Auto Scaling グループから Auto Scaling ライフサイクルフックを削除します。

c. デプロイ失敗の原因となっていたフックが存在しなくなるため、Auto Scaling は既存の EC2 インスタンスをキャンセルし、希望容量にスケーリングする新しいインスタンスを直ちに起動します。新しいインスタンスは、InService 状態にまもなく移行するはずです。新しいインスタンスには、ソフトウェアは含まれません。

5. EC2 インスタンスが InService 状態になるのを待ちます。その状態を確認するには、次のコマンドを使用します。

```
aws autoscaling describe-auto-scaling-groups --auto-scaling-group-names
ASG_NAME --query AutoScalingGroups[0].Instances[*].LifecycleState
```

6. EC2 インスタンスにフックを追加し直します。

> **Important**
>
> 以下に示す CodeDeploy コマンドを使用して、フックを追加します。Auto Scaling サービスによるフックの追加は、CodeDeploy で認識されないため、利用しないでください。

```
aws deploy update-deployment-group --application-name APPLICATION_NAME
--current-deployment-group-name DEPLOYMENT_GROUP_NAME --auto-scaling-
groups ASG_NAME
```

このコマンドを実行すると、次の処理が実行されます。

a. CodeDeploy は、Auto Scaling ライフサイクルフックを EC2 インスタンスに再インストールします。

b. CodeDeploy は、Auto Scaling グループをデプロイグループに再登録します。

7. 正常であることがわかり、使用したい Amazon S3 または GitHub のリビジョンで、フリート全体のデプロイを作成します。

   たとえば、リビジョンが、my-revision-bucket という Amazon S3 バケットにある .zip ファイルで、オブジェクトキーが httpd_app.zip である場合、次のコマンドを入力します。

   ```
   aws deploy create-deployment --application-name APPLICATION_NAME
   --deployment-group-name DEPLOYMENT_GROUP_NAME --
   revision "revisionType=S3,s3Location={bucket=my-revision-
   bucket,bundleType=zip,key=httpd_app.zip}"
   ```

   現在、Auto Scaling グループに InService インスタンスが一つ存在するため、このデプロイは機能するはずで、エラー [デプロイグループのインスタンスが見つからないため、デプロイに失敗しました] は表示されなくなります。

8. 以前にスケールインしていた場合、デプロイが成功したら、Auto Scaling グループをスケールアウトして元の容量に戻します。

   ```
   aws autoscaling set-desired-capacity --auto-scaling-group-name ASG_NAME
   --desired-capacity ORIGINAL_CAPACITY
   ```

デプロイのデッドロックの問題を解決するには (コンソール)

1. (オプション) CodeDeploy エラーの原因となっている CI/CD パイプラインをブロックして、この問題の修正中に予期しないデプロイが発生しないようにします。
2. Amazon EC2 コンソールにアクセスし、Auto Scaling の [希望する容量] の設定を書き留めます。場合によっては、この手順の最後に、この数値にスケールバックする必要があります。この設定の見つけ方の詳細については、「Auto Scaling グループの容量制限の設定」を参照してください。
3. 必要な EC2 インスタンスの数を 1 に設定:

   希望する容量が 1 より大きい場合、オプションとなります。それを 1 に減らすことで、インスタンスのプロビジョニングとデプロイにかかる時間が短くなり、トラブルシューティングが高速化されます。Auto Scaling の 希望する容量 がもともとに 0 設定されている場合、1 に増やす必要があります。これは必須です。

> **Note**
>
> この手順の残りのステップでは、1 に 希望する容量 を設定したものとします。

a. https://console.aws.amazon.com/ec2/ でAmazon EC2 コンソールを開き、ナビゲーションペインで [Auto Scaling グループ] を選択します。
b. 適切なリージョンを選択します。
c. 問題がある Auto Scaling グループに移動します。
d. [グループの詳細] で、[編集] を選択します。
e. **1** に 希望する容量 を設定。
f. [更新] を選択します。

4. デプロイグループから Auto Scaling グループの登録を解除します。

> **Warning**
>
> 次のサブステップでは、ソフトウェアなしで新しい EC2 インスタンスを起動します。コマンドを実行する前に、ソフトウェアを実行していない Auto Scaling `InService` インスタンスが許容されることを確認します。例えば、インスタンスに関連付けられているロードバランサーがソフトウェアなしでこのホストにトラフィックを送信していないことを確認してください。

a. https://console.aws.amazon.com/codedeploy/ で、CodeDeploy コンソールを開きます。
b. 適切なリージョンを選択します。
c. ナビゲーションペインで、[アプリケーション] を選択します。
d. CodeDeploy アプリケーションの名前を選択します。
e. CodeDeploy デプロイグループ名を選択する。
f. [編集] を選択します。
g. 環境設定 では、Amazon EC2 Auto Scaling グループ の選択を解除します。

> **Note**
>
> 環境設定が定義されていない場合、コンソールでは設定を保存できません。チェックをバイパスするには、どのホストにも解決しないことがわかっている EC2 または On-premises のタグを一時的に追加してください。タグを追加するには、Amazon EC2 インスタンス または オンプレミスインスタンス を選択し、タグの キー として **EC2** または **On-premises** を追加します。タグの 値 は、空欄でも構いません。

h. [Save changes] (変更の保存) をクリックします。

これらのサブステップを完了すると、次のようになります。

i. CodeDeployは、デプロイグループから Auto Scaling グループを登録解除します。

ii. CodeDeploy は Auto Scaling グループから Auto Scaling ライフサイクルフックを削除します。

iii. デプロイ失敗の原因となっていたフックが存在しなくなるため、Auto Scaling は既存の EC2 インスタンスをキャンセルし、希望容量にスケーリングする新しいインスタンスを直ちに起動します。新しいインスタンスは、InService 状態にまもなく移行するはずです。新しいインスタンスには、ソフトウェアは含まれません。

5. EC2 インスタンスが InService 状態になるのを待ちます。その状態を確認するには:

a. Amazon EC2 コンソールの https://console.aws.amazon.com/ec2/ を開いてください。

b. ナビゲーションペインで、[Auto Scaling Groups] をクリックしてください。

c. [Auto Scaling グループ] を選択します。

d. コンテンツペインで、インスタンス管理 タブを選択します。

e. インスタンス で、ライフサイクル 列がインスタンスの横で InService と表示されていることを確認します。

6. 削除にした方法と同じ方法で、Auto Scaling グループを CodeDeploy デプロイグループに再登録します。

a. https://console.aws.amazon.com/codedeploy/ で、CodeDeploy コンソールを開きます。

b. 適切なリージョンを選択します。

c. ナビゲーションペインで、[アプリケーション] を選択します。

d. CodeDeploy アプリケーションの名前を選択します。

e. CodeDeploy デプロイグループ名を選択する。

f. [編集] を選択します。

g. 環境設定 で、Amazon EC2 Auto Scaling グループ を選択し、リストから Auto Scaling グループを選択します。

h. Amazon EC2 インスタンス または オンプレミスインスタンス で、追加したタグを見つけて削除します。

i. Amazon EC2 インスタンスまたはオンプレミスインスタンス の横にあるチェックボックスの選択を解除します。

j. [Save changes] (変更の保存) をクリックします。

この設定により、ライフサイクルフックが Auto Scaling グループに再インストールされます。

7. 正常であることがわかり使用したい Amazon S3 または GitHub のリビジョンで、フリート全体のデプロイを作成します。

   たとえば、リビジョンが、`my-revision-bucket` という Amazon S3 バケットにある .zip ファイルで、オブジェクトキーが `httpd_app.zip` である場合、以下を実行します。

   a. CodeDeploy コンソールの、[デプロイグループ] ページで、[デプロイの作成] を選択します。

   b. [Revision type (リビジョンのタイプ)] の場合は、[My application is stored in Amazon S3 (Amazon S3 に保存されているアプリケーション)] を選択します。

   c. リビジョンの場所 は、`s3://my-revision-bucket/httpd_app.zip` を選択します。

   d. [リビジョンファイルの種類] で、[`.zip`] を選択します。

   e. [デプロイの作成] を選択します。

   現在、Auto Scaling グループに `InService` インスタンスが一つ存在するため、このデプロイは機能するはずで、エラー [デプロイグループのインスタンスが見つからないため、デプロイに失敗しました] は表示されなくなります。

8. 以前にスケールインしていた場合、デプロイが成功したら、Auto Scaling グループをスケールアウトして元の容量に戻します。

   a. https://console.aws.amazon.com/ec2/ でAmazon EC2 コンソールを開き、ナビゲーションペインで [Auto Scaling グループ] を選択します。

   b. 適切なリージョンを選択します。

c. Auto Scaling グループに移動します。

d. [グループの詳細] で、[編集] を選択します。

e. 希望する容量 元の値に戻す設定をします。

f. [更新] を選択します。

のエラーコード AWS CodeDeploy

このトピックでは、CodeDeploy エラーに関するリファレンス情報を提供します。

エラーコード	説明
AGENT_ISSUE	CodeDeploy デプロイは、 エージェントの問題により失敗しました。このデプロイグループのすべてのインスタンスで、エージェントがインストールされ、実行されていることを確認します。 詳細はこちら: • CodeDeploy エージェントが実行されていることの確認 • CodeDeploy エージェントをインストールする • CodeDeploy エージェントの使用
AUTO_SCALING_IAM_ROLE_PERMISSIONS	デプロイグループに関連付けられたサービスロールに、次の AWS のサービスでオペレーションを実行するために必要なアクセス権限がありません。 詳細はこちら: • ステップ 2: CodeDeployのサービスのロールを作成する • AWS サービスにアクセス許可を委任するロールの作成

エラーコード	説明
HEALTH_CONSTRAINTS	デプロイに失敗した個別のインスタンスが多すぎる、デプロイに使用できる正常なインスタンスが少なすぎる、またはデプロイグループの一部のインスタンスで問題が発生しているため、全体的なデプロイが失敗しました。 詳細はこちら: • Instance Health • インスタンスの問題のトラブルシューティング • EC2/オンプレミスのデプロイに関する問題のトラブルシューティング
HEALTH_CONSTRAINTS_INVALID	デプロイ設定で定義されている正常なインスタンスの最小数が使用できないため、デプロイを開始できません。デプロイ設定を更新するか、このデプロイグループのインスタンス数を増やすことで、正常なインスタンスの必要な数を減らすことができます。 詳細はこちら: • Instance Health • CodeDeploy のためにインスタンスを用いた操作

エラーコード	説明
IAM_ROLE_MISSING	デプロイグループに指定された名前のサービスロールが存在しないため、デプロイに失敗しました。正しいサービスロール名を使用していることを確認します。 詳細はこちら: • ステップ 2: CodeDeployのサービスのロールを作成する • CodeDeploy を使用して、デプロイグループの設定を変更します。
IAM_ROLE_PERMISSIONS	CodeDeploy にはロールを引き受けるために必要なアクセス許可がないか、使用している IAM ロールが AWS サービスでオペレーションを実行するアクセス許可を付与していません。 詳細はこちら: • ステップ 1: セットアップ • ステップ 2: CodeDeployのサービスのロールを作成する • ステップ 4: Amazon EC2 インスタンス用の IAM インスタンスプロファイルを作成する

エラーコード	説明
NO_INSTANCES	これは、次のいずれかの条件によって発生する可能性があります。 • EC2 オンプレミス Blue/Green デプロイで Amazon EC2 タグを使用すると、適切に設定されない場合があります。CodeDeploy デプロイグループで、Blue インスタンスと Green インスタンスに含まれていることを確認してください。Amazon EC2 コンソールを使用してインスタンスが正しくタグ付けされていることを確認できます。 • Amazon EC2 Auto Scaling グループを使用する場合は、Auto Scaling group グループに十分なキャパシティーがない場合があります。デプロイに十分なキャパシティーが Auto Scaling グループにあることを確認します。Amazon EC2 Auto Scaling グループのキャパシティーを確認するには、Amazon EC2 コンソールを使用して正常なインスタンスの数を確認します。 • EC2/オンプレミス Blue/Green デプロイの場合、Blue と Green のフリートのサイズが異なる場合があります。両方のフリートのサイズが同じであることを確認します。 詳細はこちら: • Tagging Instances for Deployments • チュートリアル: CodeDeploy を使用して、Auto Scaling グループにアプリケーションをデプロイします。 • Blue/Green デプロイ (コンソール) のアプリケーションを作成します。

エラーコード	説明
OVER_MAX_INSTANCES	デプロイの対象となるインスタンス数が、アカウントで許可されるインスタンス数より多いため、デプロイに失敗しました。このデプロイの対象となるインスタンスの数を減らすには、このデプロイグループのタグ設定を更新するか、対象インスタンスの一部を削除します。または、 AWS サポート に連絡して制限の引き上げをリクエストすることもできます。 詳細はこちら: • CodeDeploy を使用して、デプロイグループの設定を変更します。 • CodeDeploy のクォータ • 制限の引き上げのリクエスト
THROTTLED	IAM ロール AWS CodeDeploy で許可されているリクエストよりも多くのリクエストが行われたため、デプロイに失敗しました。リクエスト数を減らしてみてください。 詳細はこちら: • クエリ API リクエスト率
UNABLE_TO_SEND_ASG	デプロイグループが Amazon EC2 Auto Scaling グループで正しく設定されていないため、デプロイに失敗しました。CodeDeploy コンソールで、デプロイグループから Amazon EC2 Auto Scaling グループを削除し、もう一度追加してください。 詳細はこちら: • 内部構造: CodeDeploy と Auto Scaling の統合

関連トピック

CodeDeploy のトラブルシューティング

CodeDeploy リソース

CodeDeploy を利用する際に役立つ関連リソースは以下の通りです。

リファレンスガイドとサポートリソース

- AWS CodeDeploy API リファレンス — 一般的なパラメータやエラーコードなど、CodeDeploy アクションとデータ型に関する説明、構文、使用例。
- CodeDeploy テクニカル FAQ CodeDeploy に関するお客様からのよくある質問です。
- AWS サポートセンター — AWS サポート ケースを作成および管理するためのハブ。フォーラム、技術上のよくある質問、サービス状態ステータス、 AWS Trusted Advisorなど、他の役立つリソースへのリンクも含まれています。
- AWS サポートプラン — AWS サポート プランに関する情報のプライマリウェブページ。
- お問い合わせ — AWS 請求、アカウント、イベント、不正使用、その他の問題に関するお問い合わせの一元的な連絡先です。
- AWS サイト規約 — 当社の著作権と商標、お客様のアカウント、ライセンス、サイトアクセス、およびその他のトピックに関する詳細情報。

サンプル

- GitHub の CodeDeploy サンプル — CodeDeploy 用のサンプルおよびテンプレートのシナリオです。
- CodeDeploy Jenkins プラグイン — CodeDeploy 用の Jenkins プラグインです。
- CodeDeploy— CodeDeploy エージェントのオープンソースバージョンです。

ブログ

- AWS DevOps ブログ — 開発者、システム管理者、アーキテクト向けのインサイト。

AWS ソフトウェア開発キットとツール

次の AWS SDKsとツールは、CodeDeploy によるソリューション開発をサポートしています。

- AWS SDK for .NET
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP
- AWS SDK for Python (Boto)
- AWS SDK for Ruby
- AWS Toolkit for Eclipse — パート 1、2、3。
- AWS Tools for Windows PowerShell — PowerShell 環境で の機能を公開する Windows AWS SDK for .NET PowerShell コマンドレットのセット。
- AWS Tools for PowerShellの CodeDeploy コマンドレット — PowerShell 環境での CodeDeploy の機能を公開する Windows PowerShell コマンドレットのセットです。
- AWS Command Line Interface — AWS サービスにアクセスするための統一されたコマンドライン構文。 AWS CLI は、単一のセットアッププロセスを使用して、サポートされているすべてのサービスへのアクセスを有効にします。
- AWS 開発者ツール — CodeDeploy と を使用して革新的なアプリケーションを構築するのに役立つドキュメント、コードサンプル、リリースノート、その他の情報を提供する開発者ツールとリソースへのリンク AWS。

ドキュメント履歴

次の表は、CodeDeploy ユーザーガイドの最新版リリース以降の、新しく拡張された機能をサポートするために行われた、このユーザーガイドの重要な変更点について説明しています。

- API バージョン: 2014-10-06

変更	説明	日付
macOS をサポートする CodeDeploy エージェント v2.1.2 リリース	AWS CodeDeploy エージェントはバージョン 2.1.2 に更新され、EC2 Mac インスタンスでの macOS のサポートが追加されました。詳細については、CodeDeploy エージェントのバージョン履歴」および macOS 用の CodeDeploy エージェントのインストール」を参照してください。	2026 年 10 月 13 日
CodeDeploy エージェント v2.1.1 リリース	AWS CodeDeploy エージェントはバージョン 2.1.1 に更新されました。詳細については、「CodeDeploy エージェントのバージョン履歴」を参照してください。	2026 年 10 月 2 日
CodeDeploy エージェント v2.1.0 リリース	AWS CodeDeploy エージェントはバージョン 2.1.0 に更新されました。詳細については、「CodeDeploy エージェントのバージョン履歴」を参照してください。	2026 年 9 月 7 日
CodeDeploy エージェント v2.0.1 リリース	AWS CodeDeploy エージェントはバージョン 2.0.1 に更新されました。詳細について	2026 年 8 月 24 日

	は、「CodeDeploy エージェントのバージョン履歴」を参照してください。	
CodeDeploy エージェント v2.0.0 リリース	AWS CodeDeploy エージェントはバージョン 2.0.0 に更新されました。詳細については、「CodeDeploy エージェントのバージョン履歴」を参照してください。	2026 年 7 月 15 日
CodeDeploy エージェント v1.8.1 リリース	AWS CodeDeploy エージェントはバージョン 1.8.1 に更新されました。詳細については、「CodeDeploy エージェントのバージョン履歴」を参照してください。	2026 年 2 月 3 日
CodeDeploy エージェント v1.8.0 リリース	AWS CodeDeploy エージェントはバージョン 1.8.0 に更新されました。詳細については、「CodeDeploy エージェントのバージョン履歴」を参照してください。	2025 年 7 月 31 日
CodeDeploy が既存の AWS 管理ポリシーを更新しました	`AWSCodeDeployDeployerAccess` が更新されました。詳細については、「AWS 管理ポリシーの更新」を参照してください。	2024 年 12 月 16 日
CodeDeploy が既存の AWS 管理ポリシーを更新しました	`AWSCodeDeployReadOnlyAccess` が更新されました。詳細については、「AWS 管理ポリシーの更新」を参照してください。	2024 年 12 月 16 日

CodeDeploy が既存の AWS 管理ポリシーを更新しました	`AWSCodeDeployFullAccess` が更新されました。詳細については、「AWS 管理ポリシーの更新」を参照してください。	2024 年 12 月 16 日
CodeDeploy エージェント v1.7.1 リリース	AWS CodeDeploy エージェントはバージョン 1.7.1 に更新されました。詳細については、「CodeDeploy エージェントのバージョン履歴」を参照してください。	2024 年 11 月 14 日
Amazon S3 バケット名を更新しました	このガイドの Amazon S3 バケットの例を更新して、 によって予約された名前を使用しました AWS。	2024 年 6 月 17 日
代替テキスト (alt text) を追加しました	このガイドのすべてのイメージを更新し、代替テキストを追加しました。代替テキストはスクリーンリーダーによって読み取られ、視覚障害のあるユーザーにとってのドキュメントのアクセシビリティ向上に役立ちます。	2024 年 5 月 22 日
CodeDeploy エージェント v1.7.0 リリース	AWS CodeDeploy エージェントはバージョン 1.7.0 に更新されました。詳細については、「CodeDeploy エージェントのバージョン履歴」を参照してください。	2024 年 3 月 6 日

コマンドの変更	`sudo service codedeploy-agent` *`status|start|stop`* コマンドは、CodeDeploy エージェントのプロセス管理にベストプラクティスである systemd を使用しないため、推奨されなくなりました。systemd を確実に使用するには、次の例に示すように `systemctl` コマンドを使用します: `sytemctl start codedeploy-agent`。次のトピックを更新して `systemctl` コマンドを使用しました: Amazon Linux または RHEL 用の CodeDeploy エージェントのインストール、Ubuntu Server 用の CodeDeploy エージェントのインストール、スキップされたすべてのライフサイクルイベントのエラーをトラブルシューティングする、誤って削除してしまった場合は、新しい CodeDeploy ログファイルを作成してください。	2024 年 1 月 12 日
トピックの追加	「CodeDeploy エージェントプロセスの管理」と「ライフサイクルイベントスクリプトでのファイルの参照」トピックを追加しました。	2024 年 1 月 12 日

CodeDeploy がゾーン設定をサポートするようになりました	「CodeDeploy を使用してデプロイ設定を作成する」トピックを更新してゾーン設定情報を反映しました。	2023 年 12 月 7 日
CodeDeploy が終了デプロイをサポートするようになりました	終了デプロイ機能について説明する「Auto Scaling スケールインイベント中の終了デプロイの有効化」トピックを追加しました。また、「EC2/オンプレミスデプロイの AppSpec「フック」セクション」、「インプレースデプロイ用のデプロイグループを作成する (コンソール)」、「EC2/オンプレミスブルー/グリーンデプロイ用のデプロイグループを作成する (コンソール)」の各トピックを更新し、この機能を反映しました。	2023 年 12 月 7 日
JSON 形式の修正	AppSpec の「リソース」セクション (Amazon ECS と AWS Lambda デプロイのみ) トピックの JSON コードサンプルのフォーマットを修正しました。	2023 年 12 月 3 日
トラブルシューティングのトピックを追加しました	Amazon ECS デプロイに関する問題のトラブルシューティングトピックを追加しました。	2023 年 10 月 24 日

AppSpec ファイル名を更新しました	「CodeDeploy AppSpec ファイルのリファレンス」を更新して、AppSpec ファイルは EC2/オンプレミスデプロイ用に `appspec.yml` という名前にする必要があることを指定しました。	2023 年 10 月 5 日
CodeDeploy で複数のロードバランサーがサポートされるようになりました	「インプレースデプロイ用のデプロイグループを作成する (コンソール)」、「EC2/オンプレミスのブルー/グリーンデプロイ用のデプロイグループを作成する (コンソール)」、および「CodeDeploy Amazon EC2 デプロイの Elastic Load Balancing のロードバランサーを設定する」トピックを更新し、複数のロードバランサーのサポートを示すようにしました。	2023 年 9 月 26 日
VPC トピックのリージョンを更新しました	「Amazon Virtual Private Cloud で CodeDeploy を使う」トピックの表を更新し、追加のリージョンサポートを示しました。具体的には、アジアパシフィック (ハイデラバード)、アジアパシフィック (メルボルン)、欧州 (ミラノ)、欧州 (スペイン)、欧州 (チューリッヒ) の各リージョンを更新して、エージェントエンドポイントのサポートを示すようにしました。	2023 年 9 月 22 日

「リソースキットのリージョン」トピックを更新しました	AWS リージョン別のリソースキットバケット名セクションに、アジアパシフィック (大阪)、アジアパシフィック (ハイデラバード)、カナダ (中部)、欧州 (スペイン)、欧州 (チューリッヒ)、中東 (アラブ首長国連邦) の各リージョンを追加しました。また、これらのリージョンや欠落していたその他のリージョンの IAM ポリシーも更新しました。	2023 年 9 月 22 日
エージェントのインストールと更新のトピックを短くしました	「Windows Server 用 CodeDeploy エージェントのインストール」および「Windows Server 用 CodeDeploy エージェントの更新」トピックを短縮しました。冗長な Amazon S3 バケット URL と Amazon S3 COPY コマンドを削除しました。	2023 年 9 月 21 日
アジアパシフィック (ジャカルタ) リージョンを追加しました	「リージョン別のリソースキットバケット名」にアジアパシフィック (ジャカルタ) を追加しました。	2023 年 9 月 21 日
CodeDeploy が既存の AWS 管理ポリシーを更新しました	AWSCodeDeployRole マネージドポリシーを更新しました。詳細については、「AWS での AWS マネージドポリシーに関する更新」を参照してください。	2023 年 8 月 16 日

制限を追加しました	「CodeDeploy の制限」トピックに制限を追加しました。制限は、デプロイグループに関連付けられるアラームの最大数です。	2023 年 8 月 15 日
ロードバランサーに関するステップを修正しました	「EC2/オンプレミスブルー/グリーンデプロイ (コンソール) のデプロイグループを作成する」の手順を修正しました。ロードバランサーのステップがオプションとしてマークされるようになりました。	2023 年 8 月 3 日
Amazon ECS トピックの文言が明確になりました	「チュートリアル: Amazon ECS にアプリケーションをデプロイする」の文言を明確にしました。これで、アプリケーションをデプロイしていることがわかるようになりました。以前は、Amazon ECS サービスをデプロイしているという表現でした。	2023 年 8 月 3 日
CodeDeploy がイスラエル (テルアビブ) リージョンで利用可能になりました	CodeDeploy がイスラエル (テルアビブ) リージョン (il-central-1) で利用可能になりました。CodeDeploy エージェントのセットアップの手順が含まれているいくつかのトピックは、この新しいリージョンの可用性を反映するように更新されています。	2023 年 7 月 31 日

トピックの更新	「EC2/オンプレミスデプロイの問題のトラブルシューティング」トピックを更新し、ランブックを使用してトラブルシューティングタスクを自動的に実行する方法についてのヒントを追加しました。	2023 年 7 月 7 日
トピックの更新	「Amazon ECS デプロイの AppSpec「リソース」セクション」トピックを更新し、タスク定義 ARN に関する詳細情報を追加しました。	2023 年 7 月 7 日
トピックの更新	ステップ 1: オンプレミスインスタンス AWS CLI に をインストールして設定するトピックをトラブルシューティング情報で更新しました。	2023 年 7 月 7 日
トピックの更新	「サービス間の混乱した代理問題の防止」トピックを更新し、 CloudFormationによる Amazon ECS ブルー/グリーンデプロイに関する情報を追加しました。	2023 年 7 月 6 日
トピックの更新	「サービス間の混乱した代理問題の防止」トピックを更新し、 CloudFormationによる Amazon ECS ブルー/グリーンデプロイに関する情報を追加しました。	2023 年 7 月 6 日

トピックの更新	「EC2/オンプレミスコンピューティングプラットフォームの事前定義されたデプロイ設定」トピックを更新しました。Auto Scaling グループでの CodeDeployDefault.HalfAtATime 事前定義されたデプロイ設定の動作に関するメモを追加しました。	2023 年 6 月 29 日
トピックの更新	Transport Layer Security (TLS) プロトコルの新しい最小バージョンと推奨バージョンを示すために、Infrastructure security in AWS CodeDeploy トピックを更新しました。	2023 年 6 月 28 日
制限の更新	以下の制限「EC2/オンプレミスインプレースデプロイで実行できる最大時間数」を変更しました。詳細については、「制限」を参照してください。	2023 年 6 月 27 日
トピックの更新	「ステップ 3: CodeDeploy ユーザーの権限を制限する」トピックが更新され、詳細な手順が追加されました。	2023 年 5 月 31 日
CodeDeploy が既存の AWS 管理ポリシーを更新しました	AWSCodeDeployFullAccess マネージドポリシーを更新しました。詳細については、「AWS での AWS マネージドポリシーに関する更新」を参照してください。	2023 年 5 月 16 日

CodeDeploy エージェント v1.6.0 リリース	AWS CodeDeploy エージェントはバージョン 1.6.0 に更新されました。詳細については、「CodeDeploy エージェントのバージョン履歴」を参照してください。	2023 年 3 月 30 日
CodeDeploy エージェント v1.5.0 リリース	AWS CodeDeploy エージェントはバージョン 1.5.0 に更新されました。詳細については、「CodeDeploy エージェントのバージョン履歴」を参照してください。	2023 年 3 月 3 日
Amazon ECS コンピューティングプラットフォーム更新	Amazon ECS コンピューティングプラットフォームでのデプロイは、アジアパシフィック (ジャカルタ) リージョンでサポートされるようになりました。	2023 年 2 月 8 日
CodeDeploy が既存の AWS 管理ポリシーを更新しました	AWSCodeDeployRole マネージドポリシーを更新しました。詳細については、「AWS での AWS マネージドポリシーに関する更新」を参照してください。	2023 年 2 月 3 日
トピックの更新	Amazon Virtual Private Cloud で CodeDeploy を使用するトピックが、新しいリージョンと変更された AWS リージョンで更新されました。	2023 年 2 月 2 日

トピックの更新	CodeDeploy がアジアパシフィック (メルボルン) リージョン (ap-southeast-4) で利用可能になりました。CodeDeploy エージェントの設定手順を含むいくつかのトピックが、これらの新しいリージョンの可用性を反映するように更新されました。	2023 年 1 月 26 日
セキュリティのベストプラクティスの更新	「CodeDeploy の開始方法」セクションとその他のいくつかのセクションが、 AWS セキュリティのベストプラクティスに準拠するように更新されました。	2023 年 1 月 23 日
CodeDeploy エージェント v1.4.1 リリース	AWS CodeDeploy エージェントはバージョン 1.4.1 に更新されました。詳細については、「CodeDeploy エージェントのバージョン履歴」を参照してください。	2022 年 12 月 6 日
トラブルシューティングのトピックを追加しました	Windows 用の CodeDeploy エージェントで使用される長いファイルパスが原因で発生するエラーのトラブルシューティング方法に関するトピックを追加しました。詳細については、「長いファイルパスで「No such file or directory」エラーが表示される」を参照してください。	2022 年 12 月 6 日

制限を変更しました	次の制限が変更されました: AWS 「アカウントに関連付けられたカスタムデプロイ設定の最大数」。制限は 200 になりました。制限に関する詳細については、「制限」トピックを参照してください。	2022 年 9 月 7 日
CodeDeploy エージェント v1.4.0 リリース	AWS CodeDeploy エージェントはバージョン 1.4.0 に更新されました。詳細については、「CodeDeploy エージェントのバージョン履歴」を参照してください。	2022 年 8 月 31 日
いくつかの制限を修正しました。	次の制限が修正されました。AWS 「アカウントに関連付けられた同時デプロイの最大数」は 1000 になりました。「1 つのデプロイ内のインスタンスの最大数」は 1000 になりました。「進行中かつ 1 つのアカウントに関連付けられている同時デプロイで使用できるインスタンスの最大数」は 1000 になりました。アカウントに関連付けられたカスタムデプロイ設定の最大数が 100 AWS になりました。制限に関する詳細については、「制限」トピックを参照してください。	2022 年 8 月 8 日
各リージョンでサポートされている CodeDeploy エンドポイントを示す表を追加しました。	詳細については、[Amazon Virtual Private Cloud で CodeDeploy を使う] を参照してください。	2022 年 4 月 20 日

Amazon ECS ブルー/グリーンデプロイの新しい制限を追加しました。	Amazon ECS ブルー/グリーンデプロイにおいてリビジョンをデプロイしてから置き換え先環境へトラフィックが移行されるまでの最大時間数は 120 時間になりました。詳細については、「制限」トピックの「デプロイ」を参照してください。	2022 年 4 月 12 日
混乱した代理問題の防止方法に関するトピックを追加しました	詳細については、「AWS CodeDeployのAWS Identity and Access Management」を参照してください。	2022 年 3 月 14 日
CodeDeploy が既存の AWS 管理ポリシーを更新しました	AmazonEC2RoleforAWSCodeDeployLimited ロールが更新されました。詳細については、「AWS 管理ポリシーの更新」を参照してください。	2021 年 11 月 22 日
CodeDeploy が既存の AWS 管理ポリシーを更新しました	AWS CodeDeployRole が更新されました。詳細については、「AWS 管理ポリシーの更新」を参照してください。	2021 年 5 月 18 日
CodeDeploy エージェント v1.3.2 リリース	AWS CodeDeploy エージェントはバージョン 1.3.2 に更新されました。詳細については、「CodeDeploy エージェントのバージョン履歴」を参照してください。	2021 年 5 月 6 日

CodeDeploy は古くなった Amazon EC2 インスタンスの更新をサポートします	CodeDeploy は古くなった Amazon EC2 インスタンスの自動更新をサポートするようになりました。詳細については、「デプロイグループの詳細オプションの設定」を参照してください。	2021 年 2 月 23 日
CodeDeploy エージェント v1.3.1 リリース	AWS CodeDeploy エージェントはバージョン 1.3.1 に更新されました。詳細については、「CodeDeploy エージェントのバージョン履歴」を参照してください。	2020 年 12 月 22 日
CodeDeploy エージェント v1.3.0 リリース	AWS CodeDeploy エージェントはバージョン 1.3.0 に更新されました。詳細については、「CodeDeploy エージェントのバージョン履歴」を参照してください。	2020 年 11 月 10 日
CodeDeploy エージェント v1.2.1 リリース	AWS CodeDeploy エージェントはバージョン 1.2.1 に更新されました。詳細については、「CodeDeploy エージェントのバージョン履歴」を参照してください。	2020 年 9 月 23 日

CodeDeploy は、 を搭載した Amazon VPC エンドポイントをサポートします。 AWS PrivateLink	Amazon Virtual Private Cloud (Amazon VPC) を使用して AWS リソースをホストする場合、VPC と CodeDeploy の間にプライベート接続を確立できます。この接続を使用すると、CodeDeploy はパブリックインターネットを経由せずに、VPC のリソースと通信できます。詳細については、[Amazon Virtual Private Cloud で CodeDeploy を使う] を参照してください。	2020 年 8 月 11 日
CodeDeploy サービスの制限を更新	アカウントあたりのアプリケーション数と、アプリケーションごとのデプロイグループ数の上限を 1000 に更新しました。CodeDeploy サービスの制限に関する詳細については、「CodeDeploy サービスの制限」を参照してください。	2020 年 8 月 6 日
CodeDeploy エージェント v1.1.2 リリース	AWS CodeDeploy エージェントはバージョン 1.1.2 に更新されました。詳細については、「CodeDeploy エージェントのバージョン履歴」を参照してください。	2020 年 8 月 4 日

CodeDeploy エージェント 1.1.0 のリリースおよび Amazon EC2 Systems Manager との統合	CodeDeploy エージェントのバージョン 1.1.0 が利用可能になりました。詳細については、「CodeDeploy エージェントのバージョン履歴」を参照してください。Amazon EC2 Systems Manager を使用して、Amazon EC2 またはオンプレミスのインスタンスで CodeDeploy エージェントのインストールと更新を自動的に管理できるようになりました。詳細については、「Amazon EC2 Systems Manager を使用した CodeDeploy エージェントのインストール」を参照してください。	2020 年 6 月 30 日
CodeDeploy は、 を使用した Amazon ECS ブルー/グリーンデプロイの管理をサポートします。 CloudFormation	を使用して CloudFormation 、CodeDeploy を通じて Amazon ECS ブルー/グリーンデプロイを管理できるようになりました。デプロイを生成するには、Green と Blue のリソースを定義し、 CloudFormationで使用するトラフィックルーティングと安定化の設定を指定します。詳細については、「Create an Amazon ECS blue/green deployment through CloudFormation」を参照してください。	2020 年 5 月 19 日

CodeDeploy が Amazon ECS ブルー/グリーンデプロイの加重トラフィック移行をサポート	CodeDeploy が Amazon ECS Blue/Green デプロイの加重トラフィック移行をサポートするようになりました。デプロイ設定を選択または作成して、デプロイでのトラフィックのシフト間隔と、各間隔でトラフィックをシフトする割合を指定します。この変更を反映するために、トピック「Amazon ECS コンピューティングプラットフォームのデプロイ設定」を更新しました。	2020 年 2 月 6 日
セキュリティ、認証、アクセスコントロールに関するトピックを更新しました	CodeDeploy のセキュリティ、認証、およびアクセスコントロールに関する情報は、新しいセキュリティの章にまとめられました。詳細については、「セキュリティ」を参照してください。	2019 年 11 月 26 日
CodeDeploy が通知ルールをサポートします	通知ルールを使用して、デプロイの重要な変更をユーザに通知できるようになりました。詳細については、「通知ルールを作成する」を参照してください。	2019 年 11 月 5 日

トピックの更新	CodeDeploy がアジアパシフィック (香港) リージョン (ap-east-1) リージョンで利用可能になりました。CodeDeploy エージェントのセットアップの手順が含まれているいくつかのトピックは、この新しいリージョンの可用性を反映するように更新されています。このリージョンへのアクセスを明示的に有効にする必要があります。詳細については、AWS「リージョンの管理」を参照してください。	2019 年 4 月 25 日

トピックの更新	AWS CodeDeploy は、Amazon ECS サービス内のコンテナ化されたアプリケーションのブルー/グリーンデプロイをサポートするようになりました。新しい Amazon ECS コンピューティングプラットフォームを使用する CodeDeploy アプリケーションは、コンテナ化されたアプリケーションを同じ Amazon ECS サービスの新しい置き換えタスクセットにデプロイします。複数のトピックが追加および更新されており、「AWS CodeDeploy コンピューティングプラットフォームの概要」、「Amazon ECS コンピューティングプラットフォームのデプロイ」、「Amazon ECS デプロイ用の AppSpec ファイル構造」、「Amazon ECS サービスデプロイ用のアプリケーションの作成 (コンソール)」など、この変更が反映されています。	2018 年 11 月 27 日
更新された CodeDeploy エージェント	AWS CodeDeploy エージェントはバージョン 1.0.1.1597 に更新されました。詳細については、「CodeDeploy エージェントのバージョン履歴」を参照してください。	2018 年 11 月 15 日

コンソールの更新	このガイドの手順が、CodeDeploy コンソールの新しい設計に合わせて更新されました。	2018 年 10 月 30 日
CodeDeploy エージェントでサポートされている新しい最小バージョン	サポートされている AWS CodeDeploy エージェントの最小バージョンは 1.8.x になりました。詳細については、「CodeDeploy エージェントのバージョン履歴」を参照してください。	2018 年 8 月 7 日

以前の更新

次の表に、2018 年 6 月以前の「AWS CodeDeploy ユーザーガイド」の各リリースにおける重要な変更点を示します。

変更	説明	変更日
トピックの更新	CodeDeploy が欧州 (パリ) リージョン (eu-west-3) リージョンで利用可能になりました。CodeDeploy エージェントのセットアップの手順が含まれているいくつかのトピックは、この新しいリージョンの可用性を反映するように更新されています。	2017 年 12 月 19 日
トピックの更新	CodeDeploy が中国 (寧夏) リージョンで利用可能になりました。 中国 (北京) リージョンまたは中国 (寧夏) リージョンでサービスを使用するには、そのリージョンのアカウントと認証情報が必要です。他の AWS リージョンのアカウントと認証情報は、北京および寧夏リージョンでは機能せず、その逆も同様です。 CodeDeploy リソースキットのバケット名や CodeDeploy エージェントのインストール手順など、中国リージョ	2017 年 12 月 11 日

変更	説明	変更日
	ン向けのいくつかのリソースに関する情報は、今回の「CodeDeploy ユーザーガイド」には含まれていません。 詳細については: • 中国 (北京) リージョンでの の開始方法 AWS の CodeDeploy • 中国リージョン向け CodeDeploy ユーザーガイド (英語版 | 中国語版)	
トピックの更新	CodeDeploy が Lambda 関数のデプロイをサポートするようになりました。 AWS Lambda のデプロイは、既存の Lambda 関数からの着信トラフィックを、更新された Lambda 関数バージョンに移行します。デプロイ設定を選択または作成して、デプロイ内のトラフィックシフト間隔の数と各間隔でシフトするトラフィックの割合を指定します。 AWS Lambda デプロイは AWS サーバーレスアプリケーションモデル (AWS SAM) でサポートされるため、 AWS SAM デプロイ設定を使用して、 AWS Lambda デプロイ中にトラフィックをシフトする方法を管理できます。複数のトピックが追加および更新されており、「CodeDeploy コンピューティングプラットフォームの概要」、「AWS Lambda コンピューティングプラットフォームでのデプロイ」、「AWS Lambda コンピューティングプラットフォームのデプロイを作成する (コンソール)」、「AWS Lambda 関数デプロイ用のアプリケーションを作成する (コンソール)」、「AWS Lambda デプロイ用の AppSpec ファイルを追加する」など、この変更が反映されています。	2017 年 11 月 28 日

変更	説明	変更日
新しいトピック	CodeDeploy は、CodeDeploy エージェントがインストールされているローカルマシンまたはインスタンスへの直接的なデプロイをサポートするようになりました。ローカルにデプロイをテストし、エラーがある場合は、CodeDeploy エージェントのエラーログを使用してデバッグできます。また、ローカルデプロイを使用してアプリケーションリビジョンの整合性、 AppSpec ファイルのコンテンツなどをテストできます。詳細については、「CodeDeploy エージェントを使用してローカルマシン上のデプロイパッケージを検証する」を参照してください。	2017 年 11 月 16 日
トピックの更新	デプロイグループ内の Elastic Load Balancing ロードバランサーの CodeDeploy サポートが拡張され、Blue/Green デプロイおよびインプレースデプロイの両方で Network Load Balancer が含まれました。これで、デプロイグループの Application Load Balancer、Classic Load Balancer、Network Load Balancer が選択できます。ロードバランサーは Blue/Green デプロイでは必須、インプレースデプロイでは任意です。「Integrating CodeDeploy with Elastic Load Balancing」、「インプレースデプロイ (コンソール) 用のアプリケーションを作成」、「デプロイの前提条件」、「Integrating CodeDeploy with Elastic Load Balancing」、および「インプレースデプロイ (コンソール) 用のアプリケーションを作成」を含む、数多くのトピックが更新され、この追加サポートが反映されています。	2017 年 9 月 12 日

変更	説明	変更日
トピックの更新	デプロイグループ内の Elastic Load Balancing ロードバランサーの CodeDeploy サポートが拡張され、Blue/Green デプロイおよびインプレースデプロイの両方で Application Load Balancer が含まれました。これで、デプロイグループの Application Load Balancer と Classic Load Balancer が選択できます。ロードバランサーは Blue/Green デプロイでは必須、インプレースデプロイでは任意です。「Integrating CodeDeploy with Elastic Load Balancing」、「CodeDeploy でアプリケーションを作成する」、「CodeDeploy でデプロイグループを作成する」を含むトピックが更新され、この追加サポートが反映されています。	2017 年 8 月 10 日
新しく更新されたトピック	CodeDeploy で、デプロイグループに含まれるインスタンスの結合と交差の識別に複数のタググループを使用できるようになりました。単一のタググループを使用する場合は、グループ内の少なくとも 1 つのタグによって識別されたインスタンスがデプロイグループに含まれます。複数のタググループを使用する場合は、タググループそれぞれの少なくとも 1 つのタグによって識別されたインスタンスのみが含まれます。でプログループにインスタンスを追加する新しい方法の詳細については、「Tagging Instances for Deployments」を参照してください。このサポートを反映するように更新されたその他のトピックには、「インプレースデプロイ (コンソール) 用のアプリケーションを作成」、「Blue/Green デプロイ (コンソール) のアプリケーションを作成します。」、「インプレースデプロイ用のデプロイグループを作成する (コンソール)」、「EC2/オンプレミス Blue/Green デプロイ用のデプロイグループを作成する (コンソール)」、「Deployments」、および チュートリアル: CodeDeploy を使用して、GitHub からアプリケーションをデプロイします。 の ステップ 5: アプリケーションおよびデプロイグループを作成します。 があります。	2017 年 7 月 31 日

変更	説明	変更日
トピックの更新	Windows Server インスタンスで CodeDeploy エージェントをインストールする 2 つの方法が「Windows サーバー用の CodeDeploy エージェントです。」に追加されました。Windows PowerShell コマンドに加えて、インストールファイルのダウンロードに、直接 HTTPS リンクを使用する方法と Amazon S3 コピーコマンドを使用する方法の説明が追加されました。ファイルがインスタンスにダウンロードまたはコピーされた後、手動でインストールを実行できます。	2017 年 7 月 12 日
トピックの更新	CodeDeploy で GitHub アカウントおよびリポジトリへの接続を管理する方法を改善しました。CodeDeploy アプリケーションを GitHub リポジトリに関連付けるために、GitHub アカウントへの接続を最大 25 まで作成して保存できるようになりました。各接続は複数のリポジトリをサポートできます。最大 25 個の異なる GitHub アカウントへの接続を作成できます。また、1 つのアカウントへの複数の接続を作成できます。アプリケーションを GitHub アカウントに接続すると、必要なアクセス権限が CodeDeploy で管理されるため、ユーザー側のアクションは一切不要です。このサポートを反映するために「GitHub リポジトリに格納されているリビジョンについての情報を指定します」、「GitHub と CodeDeploy との統合」、および「チュートリアル: CodeDeploy を使用して、GitHub からアプリケーションをデプロイします。」を更新しました。	2017 年 5 月 30 日

変更	説明	変更日
トピックの更新	以前は、CodeDeploy エージェントが、前回の成功したデプロイのアプリケーションリビジョンに含まれていないファイルをデプロイ先で検出すると、現行のデプロイはデフォルトで失敗しました。CodeDeploy では、これらのファイルをエージェントで処理する方法として、デプロイを失敗させる、コンテンツを保持する、またはコンテンツを上書きするオプションが提供されるようになりました。このサポートを反映するために「CodeDeploy でデプロイを作成する」を更新し、新しいセクション 既存のコンテンツでのロールバック動作 が CodeDeploy を使用した再デプロイおよびデプロイのロールバック に追加されました。	2017 年 5 月 16 日
トピックの更新	CodeDeploy コンソールまたは AWS CLIを使用して Elastic Load Balancing の Classic Load Balancer をデプロイグループに割り当てることができるようになりました。インプレースデプロイ中は、ロードバランサーにより、デプロイ先のインスタンスに対するインターネットトラフィックのルーティングがブロックされ、そのインスタンスへのデプロイが完了した時点でインスタンスに対するトラフィックのルーティングが再開されます。この新しいサポートを反映するために、「他の AWS サービスとの統合」、「Integrating CodeDeploy with Elastic Load Balancing」、「インプレースデプロイ (コンソール) 用のアプリケーションを作成」、「インプレースデプロイ用のデプロイグループを作成する (コンソール)」、「AppSpec の「hooks」セクション」など、いくつかのトピックを更新しました。トラブルシューティングガイドに新しいセクション「失敗した ApplicationStop、BeforeBlockTraffic、または AfterBlockTraffic デプロイライフサイクルイベントのトラブルシューティング」を追加しました。	2017 年 4 月 27 日

変更	説明	変更日
トピックの更新	CodeDeploy コンソールまたは AWS CLIを使用して Elastic Load Balancing の Classic Load Balancer をデプロイグループに割り当てることができるようになりました。インプレースデプロイ中は、ロードバランサーにより、デプロイ先のインスタンスに対するインターネットトラフィックのルーティングがブロックされ、そのインスタンスへのデプロイが完了した時点でインスタンスに対するトラフィックのルーティングが再開されます。この新しいサポートを反映するために、「他の AWS サービスとの統合」、「Integrating CodeDeploy with Elastic Load Balancing」、「インプレースデプロイ (コンソール) 用のアプリケーションを作成」、「インプレースデプロイ用のデプロイグループを作成する (コンソール)」、「AppSpec の「hooks」セクション」など、いくつかのトピックを更新しました。トラブルシューティングガイドに新しいセクション「失敗した ApplicationStop、BeforeBlockTraffic、または AfterBlockTraffic デプロイライフサイクルイベントのトラブルシューティング」を追加しました。	2017 年 5 月 1 日

変更	説明	変更日
トピックの更新	CodeDeploy が中国 (北京) リージョンで利用可能になりました。 中国 (北京) リージョンまたは中国 (寧夏)でサービスを使用するには、そのリージョンのアカウントと認証情報が必要です。他の AWS リージョンのアカウントと認証情報は、北京および寧夏リージョンでは機能せず、その逆も同様です。 CodeDeploy リソースキットのバケット名や CodeDeploy エージェントのインストール手順など、中国リージョン向けのいくつかのリソースに関する情報は、今回の「CodeDeploy ユーザーガイド」には含まれていません。 詳細については: • 中国 (北京) リージョンでの の開始方法 AWS の CodeDeploy • 中国リージョン向け CodeDeploy ユーザーガイド (英語版 | 中国語版)	2017 年 3 月 29 日

変更	説明	変更日
新しく更新されたトピック	Blue/Green デプロイの新しい CodeDeploy サポートを反映させるいくつかの新しいトピックが導入され、その中では、デプロイグループのインスタンス (元の環境) が、別の一連のインスタンス (置き換え先環境) で置き換えられます。「Blue/Green デプロイの概要」では、CodeDeploy が使用する Blue/Green 方法の高度な説明を提供しています。新しい追加トピックには、「Blue/Green デプロイ (コンソール) のアプリケーションを作成します。」、「EC2/オンプレミス Blue/Green デプロイ用のデプロイグループを作成する (コンソール)」と「CodeDeploy Amazon EC2 デプロイ用の Elastic Load Balancing でロードバランサーをセットアップする」が含まれます。 CodeDeploy でデプロイを作成する、CodeDeploy でデプロイ設定を使用する、CodeDeploy でアプリケーションを作成する、CodeDeploy でのデプロイグループの使用、CodeDeploy でのデプロイグループの使用、AppSpec の「hooks」セクション を含む、数多くのトピックも更新されました。	2017 年 1 月 25 日
新しく更新されたトピック	新しいトピック「」では[register-on-premises-instance] コマンド (IAM セッション ARN) を使用してオンプレミスインスタンスを登録、 を通じて定期的に更新される一時的な認証情報を使用してオンプレミスインスタンスを認証および登録する方法について説明します AWS Security Token Service。この方法により、各インスタンスで静的 IAM ユーザーの認証情報だけを使用するより、オンプレミスインスタンスの大規模フリートをサポートするためのより良いサポートが提供されます。「Working with On-Premises Instances」はこの新しいサポートを反映して更新されました。	2016 年 28 月 12 日

変更	説明	変更日
トピックの更新	CodeDeploy が欧州 (ロンドン) リージョン (eu-west-2) リージョンで利用可能になりました。CodeDeploy エージェントのセットアップの手順が含まれているいくつかのトピックは、この新しいリージョンの可用性を反映するように更新されています。	2016 年 12 月 13 日
トピックの更新	CodeDeploy がカナダ (中部) リージョン (ca-central-1) で利用可能になりました。CodeDeploy エージェントのセットアップの手順が含まれているいくつかのトピックは、この新しいリージョンの可用性を反映するように更新されています。	2016 年 12 月 8 日
トピックの更新	CodeDeploy が米国東部 (オハイオ) リージョン (us-east-2) で利用可能になりました。CodeDeploy エージェントのセットアップの手順が含まれているいくつかのトピックは、この新しいリージョンの可用性を反映するように更新されています。	2016 年 10 月 17 日
新しいトピック	新しいセクション「認証とアクセスコントロール」は、認証情報の使用を通してリソースへのアクセスをセキュリティで保護するのに役立つ「AWS Identity and Access Management (IAM)」および CodeDeploy の使用についての包括的な情報を提供します。これらの認証情報は、Amazon S3 バケットからのアプリケーションリビジョンの取得や Amazon EC2 インスタンスのタグの読み取りなど、 AWS リソースへのアクセスに必要なアクセス許可を提供します。	2016 年 10 月 11 日

変更	説明	変更日
トピックの更新	「Windows サーバーで CodeDeploy エージェントを更新する」は、Windows Server 用の新しい CodeDeploy エージェントアップデーターの可用性を反映するよう更新されています。Windows Server インスタンスにインストールすると、アップデーターは新しいバージョンを定期的に確認します。新しいバージョンが検出された場合、アップデーターは、最新バージョンをインストールする前に、インストールされている場合はエージェントの現在のバージョンをアンインストールします。	2016 年 10 月 4 日
トピックの更新	CodeDeploy は Amazon CloudWatch アラームと統合し、アラームしきい値で指定されているとおり、連続した期間の数、指定したアラームの状態に変更がある場合、デプロイを停止できるようにします。 CodeDeploy は、デプロイの失敗やアクティブ化されたアラームのような特定の条件が満たされている場合、デプロイの自動的なロールバックをサポートしています。 いくつかのトピックは CodeDeploy でアプリケーションを作成する、CodeDeploy でデプロイグループを作成する、CodeDeploy を使用して、デプロイグループの設定を変更します。、Deployments、CodeDeploy を使用した再デプロイおよびデプロイのロールバック、CodeDeploy との製品とサービスの統合 を含め、新しいトピック CodeDeploy での CloudWatch アラームを使用したデプロイのモニタリング と共にこれらの変更を反映するよう更新されています。	2016 年 9 月 15 日

変更	説明	変更日
新しく更新されたトピック	CodeDeploy が Amazon CloudWatch Events との統合を提供するようになりました。CodeDeploy デプロイグループに属するデプロイの状態またはインスタンスの状態の変更が検出された場合に、CloudWatch Events を使用して 1 つまたは複数のアクションを開始できるようになりました。AWS Lambda 関数を呼び出すアクション、Kinesis ストリームまたは Amazon SNS トピックに発行するアクション、Amazon SQS キューにメッセージをプッシュするアクション、または CloudWatch アラームアクションをトリガーするアクションを組み込むことができます。詳細については、「Amazon CloudWatch Events を使用したデプロイのモニタリング」を参照してください。	2016 年 9 月 9 日
トピックの更新	「Integrating CodeDeploy with Elastic Load Balancing」トピック および「他の AWS サービスとの統合」は、追加の負荷分散オプションを反映するように更新されました。CodeDeploy で、Elastic Load Balancing の Classic Load Balancer と Application Load Balancer がサポートされるようになりました。	2016 年 8 月 11 日
トピックの更新	CodeDeploy がアジアパシフィック (ムンバイ) リージョン (ap-south-1) で利用可能になりました。CodeDeploy エージェントのセットアップの手順が含まれているいくつかのトピックは、この新しいリージョンの可用性を反映するように更新されています。	2016 年 6 月 27 日

変更	説明	変更日
トピックの更新	CodeDeploy が、アジアパシフィック (ソウル) (ap-northeast-2) リージョンで利用可能になりました。CodeDeploy エージェントのセットアップの手順が含まれているいくつかのトピックは、この新しいリージョンの可用性を反映するように更新されています。 目次はインスタンス、デプロイ設定、アプリケーション、デプロイグループ、リビジョン、およびデプロイのセクションを含めるように再編成されました。新しいセクションが CodeDeploy のチュートリアルに追加されています。より使いやすくするため、CodeDeploy AppSpec ファイルのリファレンス および CodeDeploy のトラブルシューティング を含むいくつかの長いトピックは、短いトピックに分割されています。CodeDeploy エージェントの設定情報は新しいトピック「CodeDeploy エージェント設定リファレンス」に移動しました。	2016 年 6 月 15 日
新しく更新されたトピック	「のエラーコード AWS CodeDeploy」では、CodeDeploy デプロイが失敗したときに表示される可能性のある一部のエラーメッセージについての情報を提供します。CodeDeploy のトラブルシューティング の以下のセクションは、デプロイの問題の解決により役立つよう更新されました。 • Amazon EC2 Auto Scaling グループの EC2 インスタンスが起動に失敗し、「ハートビートのタイムアウト」というエラーが表示される • 複数のデプロイグループを 1 つの Amazon EC2 Auto Scaling グループに関連付けることは避ける	2016 年 4 月 20 日

変更	説明	変更日
トピックの更新	CodeDeploy が南米 (サンパウロ) (sa-east-1) リージョンで利用可能になりました。CodeDeploy エージェントのセットアップの手順が含まれているいくつかのトピックは、この新しいリージョンの可用性を反映するように更新されています。 「CodeDeploy エージェントの使用」は、新しい :max_revisions: を反映するように更新されました。これは、CodeDeploy エージェントがアーカイブするデプロイグループのアプリケーションリビジョンの数を指定するために使用します。	2016 年 3 月 10 日
新しく更新されたトピック	CodeDeploy は、デプロイグループへのトリガーの追加をサポートし、デプロイグループのデプロイまたはインスタンスに関連するイベントに関する通知を受信するようになりました。これらの通知は、トリガーのアクションの一部にした Amazon Simple Notification Service トピックをサブスクライブする受信者に送信されます。カスタマイズされた通知のワークフローでトリガーが発生した場合に作成される JSON データも使用できます。詳細については、「Monitoring Deployments with Amazon SNS Event Notifications」を参照してください。 手順は [Application details] ページの再設計を反映するように更新されています。 CodeDeploy のトラブルシューティング の デプロイ中にインスタンスを削除した場合、デプロイは最大 1 時間は失敗しません。 セクションが更新されました。 「CodeDeploy のクォータ」は、改訂された 1 つのアプリケーションに関連付けられるデプロイグループの数の制限、最小の正常なインスタンス設定の許可された値、AWS SDK for Rubyの必要なバージョンを反映するように更新されました。	2016 年 2 月 17 日

変更	説明	変更日
新しく更新されたトピック	CodeDeploy が米国西部 (北カリフォルニア) リージョン (us-west-1) で利用可能になりました。CodeDeploy エージェントの設定手順を含むいくつかのトピックが、この新しいリージョンの追加を反映するように更新されました。 「CodeDeploy リポジトリタイプを選択する」は、CodeDeploy にサポートされているリポジトリタイプを一覧表示し、説明しています。この新しいトピックは、他のリポジトリタイプのサポートの導入に合わせて更新されます。 「CodeDeploy エージェントのオペレーションの管理」は、エージェントのサポートされたバージョンに関する情報と同様、CodeDeploy エージェントの現在のバージョンを報告するためにインスタンスに追加された新しい `.version` ファイルに関する情報が更新されました。 JSON、および YAML の例を含め、コードサンプルを強調する構文がユーザーガイドに追加されています。 CodeDeploy 用のアプリケーション仕様ファイルをリビジョンに追加 は、詳しい手順として再編成されました。	2016 年 1 月 20 日
新しいトピック	別の AWS アカウントにアプリケーションをデプロイする では、組織の別のアカウントに属するデプロイを開始するためのセットアップ要件とプロセスについて説明します。他のアカウントの完全な認証情報セットは必要ありません。これにより、開発およびテスト環境に関連付けたものや本稼働環境に関連付けたものなど、複数のアカウントをさまざまな目的で使用する組織にとって非常に便利です。	2015 年 12 月 30 日
トピックの更新	CodeDeploy との製品とサービスの統合 トピックは再設計されています。CodeDeploy 統合に関連するブログ投稿や動画の例のリストがあるコミュニティからの統合の例についてのセクションが含まれます。	2015 年 12 月 16 日

変更	説明	変更日
トピックの更新	CodeDeploy がアジアパシフィック (シンガポール) リージョン (ap-southeast-1) で利用可能になりました。CodeDeploy エージェントのセットアップの手順が含まれているいくつかのトピックは、この新しいリージョンの可用性を反映するように更新されています。	2015 年 12 月 9 日
トピックの更新	「CodeDeploy エージェントの使用」は、CodeDeploy エージェント設定ファイルの新しい `:proxy_uri:` オプションを反映するように更新されています。CodeDeploy AppSpec ファイルのリファレンス では、フックスクリプトがデプロイライフサイクルイベント中にアクセスできる新しい環境変数 `DEPLOYMENT_GROUP_ID` の使用に関する情報が更新されました。	2015 年 12 月 1 日
トピックの更新	「ステップ 2: CodeDeployのサービスのロールを作成する」は、CodeDeploy のサービスロールの作成の新しい手順を反映し、他の改善を組み込むために更新されています。	2015 年 11 月 13 日
トピックの更新	CodeDeploy が欧州 (フランクフルト) リージョン (eu-central-1) で利用可能になりました。CodeDeploy エージェントのセットアップの手順が含まれているいくつかのトピックは、この新しいリージョンの可用性を反映するように更新されています。 CodeDeploy のトラブルシューティング トピックでは、インスタンスの時間設定の正確さを確認することに関する情報が更新されました。	2015 年 10 月 19 日
新しいトピック	CloudFormation CodeDeploy リファレンスの テンプレート は、CodeDeploy アクションの新しい CloudFormation サポートを反映するために公開されました。 Primary Components トピックを作成し、ターゲットリビジョンの定義を紹介します。	2015 年 10 月 1 日

変更	説明	変更日
トピックの更新	CodeDeploy でデプロイグループを作成する は、ワイルドカード検索を使用してデプロイグループのインスタンスを探す機能を反映するように更新されています。 Instance Health は、正常なインスタンスの最小数の概念を明確にするために更新されています。	2015 年 8 月 31 日
トピックの更新	CodeDeploy が、アジアパシフィック (東京) (ap-northeast-1) リージョンで利用可能になりました。CodeDeploy エージェントのセットアップの手順が含まれているいくつかのトピックは、この新しいリージョンの可用性を反映するように更新されています。	2015 年 8 月 19 日
トピックの更新	CodeDeploy は、Red Hat Enterprise Linux (RHEL) オンプレミスインスタンスおよび Amazon EC2 インスタンスへのデプロイをサポートするようになりました。詳細については、以下の各トピックを参照してください。 • CodeDeploy エージェントで対応するオペレーティングシステム • CodeDeploy のためにインスタンスを用いた操作 • チュートリアル: Amazon EC2 インスタンスに WordPress をデプロイする (Amazon Linux または Red Hat エンタープライズ Linux および Linux、macOS、または Unix) • チュートリアル: CodeDeploy (Windows サーバー、Ubuntu サーバー、または Red Hat エンタープライズ Linux) を使用してオンプレミスインスタンスにアプリケーションをデプロイします。	2015 年 6 月 23 日

変更	説明	変更日
トピックの更新	CodeDeployでは、デプロイのスクリプトがデプロイ中に使用できる一連の環境変数を提供されるようになりました。これらの環境変数には、現在の CodeDeploy アプリケーションの名前、デプロイグループ、デプロイライフサイクルイベント、および現在の CodeDeploy デプロイ識別子のような情報が含まれます。詳細については、『CodeDeploy AppSpec ファイルのリファレンス』の「AppSpec の「hooks」セクション」の最後を参照してください。	2015 年 5 月 29 日
トピックの更新	CodeDeploy は、同等のポリシーを手動で作成する代わりに使用できる IAM の AWS マネージドポリシーのセットを提供するようになりました。具体的には次のとおりです。 • ユーザーが CodeDeploy のみを使用してリビジョンを登録し、CodeDeploy を通してそれらをデプロイできるようにするポリシー。 • ユーザー の CodeDeploy リソースへのフルアクセスを提供するポリシー。 • ユーザーの CodeDeploy リソースへのポリシーへの読み取り専用アクセスを提供するポリシー。 • CodeDeploy が Amazon EC2 タグ、オンプレミスインスタンスタグ、または Amazon EC2 Auto Scaling グループ名によって Amazon EC2 インスタンスを識別し、アプリケーションリビジョンをそれぞれに応じてデプロイできるようにサービスロールにアタッチするポリシー。 詳細については、「認証とアクセスコントロール」の カスタマーマネージドポリシーの例 セクションを参照してください。	2015 年 5 月 29 日

変更	説明	変更日
トピックの更新	CodeDeploy が欧州 (アイルランド) リージョン (eu-west-1) およびアジアパシフィック (シドニー) リージョン (ap-southeast-2) で利用可能になりました。CodeDeploy エージェントの設定手順を含むいくつかのトピックが、これらの新しいリージョンの可用性を反映するように更新されました。	2015 年 5 月 7 日
新しいトピック	CodeDeploy は、オンプレミスインスタンスと Amazon EC2 インスタンスへのデプロイをサポートするようになりました。以下のトピックは、この新しいサポートを説明するために追加されました。 • Working with On-Premises Instances • チュートリアル: CodeDeploy (Windows サーバー、Ubuntu サーバー、または Red Hat エンタープライズ Linux) を使用してオンプレミスインスタンスにアプリケーションをデプロイします。 • Working with On-Premises Instances	2015 年 4 月 2 日
新しいトピック	「CodeDeploy リソース」が追加されました。	2015 年 4 月 2 日

変更	説明	変更日
トピックの更新	CodeDeploy のトラブルシューティング が更新されました。 • 新しい 長時間実行されているプロセスにより、デプロイが失敗することがある セクションでは、長期プロセスのデプロイの障害を特定し、対処するために実行できるステップについて説明します。 • 「Amazon EC2 Auto Scaling の一般的なトラブルシューティング」セクションは、CodeDeploy が、CodeDeploy エージェントに対して Amazon EC2 Auto Scaling タイムアウトロジックを 5 分から 1 時間に拡大したことを表示するように更新されました。 • 新しい「Amazon EC2 Auto Scaling ライフサイクルフックの不一致により、Amazon EC2 Auto Scaling グループへの自動デプロイが停止または失敗することがある」セクションでは、Amazon EC2 Auto Scaling グループに対して障害が発生した自動デプロイを特定し、対処するために実行できるステップについて説明します。	2015 年 4 月 2 日

変更	説明	変更日
トピックの更新	以下のトピックは、独自のカスタムポリシーを作成し、それらを IAM のユーザーとロールにアタッチするための新しい推奨事項を反映するように更新されました。 • CodeDeploy と共に動作するように Amazon EC2 インスタンスを構成します • ステップ 4: Amazon EC2 インスタンス用の IAM インスタンスプロファイルを作成する • ステップ 2: CodeDeployのサービスのロールを作成する CodeDeploy のトラブルシューティング に 2 つのセクションが追加されました。 • 一般的なトラブルシューティングのチェックリスト • Windows PowerShell スクリプトで、デフォルトで 64 ビットバージョンの Windows PowerShell スクリプトを使用できない 「CodeDeploy AppSpec ファイルのリファレンス」の AppSpec の「hooks」セクション セクションは、利用可能なデプロイライフサイクルイベントをより正確に説明するために更新されました。	2015 年 2 月 12 日
トピックの更新	CodeDeploy のトラブルシューティング: Amazon EC2 Auto Scaling グループの EC2 インスタンスが起動に失敗し、「ハートビートのタイムアウト」というエラーが表示される に新しいセクションが追加されました。 CodeDeploy との製品とサービスの統合 に CloudBees セクションが追加されました。	2015 年 1 月 28 日

変更	説明	変更日
トピックの更新	CodeDeploy のトラブルシューティング に以下のセクションが追加されました。 • 何らかのテキストエディタを使用して AppSpec ファイルとシェルスクリプトを作成すると、デプロイが失敗する場合がある • macOS の Finder を使用してアプリケーションリビジョンをバンドルすると、デプロイが失敗することがある • 失敗した ApplicationStop、BeforeBlockTraffic、または AfterBlockTraffic デプロイライフサイクルイベントのトラブルシューティング • 「不明なエラー: 読み取り用に開いていません」で失敗した DownloadBundle デプロイライフサイクルイベントのトラブルシューティング • Amazon EC2 Auto Scaling の一般的なトラブルシューティング	2015 年 1 月 20 日
新しいトピック	CodeDeploy との製品とサービスの統合 セクションでは、次のトピックを含めるように更新されました。 • CodeDeploy と Amazon EC2 Auto Scaling の統合 • チュートリアル: CodeDeploy を使用して、Auto Scaling グループにアプリケーションをデプロイします。 • Monitoring Deployments • Integrating CodeDeploy with Elastic Load Balancing • GitHub と CodeDeploy との統合 • チュートリアル: CodeDeploy を使用して、GitHub からアプリケーションをデプロイします。	2015 年 1 月 9 日

変更	説明	変更日
トピックの更新	• CodeDeploy を使用して CodePipeline から自動的にデプロイする セクションが GitHub と CodeDeploy との統合 に追加されました。そのリポジトリのソースコードが変更されるたびに、自動的に GitHub リポジトリからデプロイをトリガーできます。 • Amazon EC2 Auto Scaling の問題のトラブルシューティング セクションが CodeDeploy のトラブルシューティング に追加されました。この新しいセクションでは、Amazon EC2 Auto Scaling グループへのデプロイに関してよくある問題のトラブルシューティングを行う方法について説明します。 • 新しいサブセクションの「ファイル例」が、『CodeDeploy AppSpec ファイルのリファレンス』の「AppSpec の「ファイル」セクション (EC2/オンプレミスデプロイのみ)」セクションに追加されています。この新しいサブセクションには、デプロイの間に特定のファイル/フォルダを Amazon EC2 インスタンスの特定の場所にコピーするように CodeDeploy に指示する AppSpec ファイル の `files` セクションの使用方法のさまざまな例が含まれます。	2015 年 1 月 8 日
新しいトピック	「Monitoring Deployments」が追加されました。CodeDeploy は AWS CloudTrail、 AWS アカウントで CodeDeploy によって行われた、または CodeDeploy に代わって行われた API コールをキャプチャし、指定した Amazon S3 バケットにログファイルを配信するサービスである と統合されています。	2014 年 12 月 17 日
初回一般リリース	これは、「CodeDeploy ユーザーガイド」の初回リリースです。	2014 年 11 月 12 日

AWS 用語集

最新の AWS 用語については、 AWS の用語集 リファレンスのAWS 用語集を参照してください。

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。