aws

ユーザーガイド

AWS CodePipeline



API バージョン 2015-07-09

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

AWS CodePipeline: 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

CodePipeline とは何ですか。 1
継続的デリバリーと継続的インテグレーション 1
CodePipeline で何ができますか。 2
CodePipeline のクイックルック 2
CodePipeline を使い始めるにはどうすればよいですか。 3
概念 4
Pipelines 4
パイプライン実行 6
ステージオペレーション 7
アクション実行 8
実行タイプ 8
アクションタイプ 8
アーティファクト 9
ソースリビジョン 9
トリガ 9
[変数] 10
条件 10
Rules 11
DevOps パイプラインの例 11
パイプライン実行の仕組み 13
パイプライン実行の開始方法 14
パイプライン実行でソースリビジョンを処理する方法 14
ソースオーバーライドと EventBridge 入力トランスフォーマーの連携方法 15
パイプライン実行の停止方法 16
SUPERSEDED モードでの実行の処理方法 20
QUEUED モードでの実行の処理方法 21
PARALLEL モードでの実行の処理方法 23
パイプラインのフローを管理する 23
入力および出力アーティファクト 26
ステージ条件はどのように機能しますか? 29
ステージ条件のルール 32
パイプラインのタイプ 33
適切なパイプラインのタイプの選択 33
開始方法 39

にサインアップする AWS アカウント 39
CodePipeline への管理アクセスに マネージドポリシーを適用する 40
のインストール AWS CLI 41
CodePipeline のコンソールを開く 42
次の手順 42
製品およびサービスの統合 43
CodePipeline アクションタイプとの統合 43
ソースアクションの統合 43
ビルドアクションの統合 51
テストアクションの統合 53
デプロイアクションの統合 55
Amazon Simple Notification Service との承認アクションの統合 62
呼び出しアクションの統合 62
CodePipeline との一般的統合 64
コミュニティからの例 67
ブログ記事 67
チュートリアル 71
チュートリアル: CodePipeline を使用して Amazon EC2 インスタンスにデプロイする 72
前提条件 73
ステップ 1: Amazon EC2 Linux インスタンスを作成する 73
ステップ 2: EC2 インスタンスロールにアーティファクトバケットのアクセス許可を追加する 75
ステップ 3: リポジトリにスクリプトファイルを追加する 76
ステップ 4: パイプラインを作成する 77
ステップ 5: パイプラインをテストする 81
チュートリアル: CodePipeline (V2 タイプ) を使用して Docker イメージを構築し Amazon ECR にプッシュする 82
前提条件 83
ステップ 1: ソースリポジトリに Dockerfile を追加する 83
ステップ 2: imagedefinitions.json ファイルをソースリポジトリに追加する 85
ステップ 3: パイプラインを作成する 86
ステップ 4: パイプラインのテスト 91
チュートリアル: CodePipeline を使用して Amazon EKS にデプロイする 92
前提条件 92
ステップ 1: (オプション) Amazon EKS でクラスターを作成する 93
ステップ 2: Amazon EKS でプライベートクラスターを設定する 96

ステップ 3: IAM の CodePipeline サービスロールポリシーを更新する 96
ステップ 4: CodePipeline サービスロールのアクセスエントリを作成する 99
ステップ 5: ソースリポジトリを作成し、Helm chart 設定ファイルを追加する 100
ステップ 6: パイプラインを作成する 101
チュートリアル: コンピューティングを使用してコマンドを実行するパイプラインを作成する (V2 タイプ) 103
前提条件 104
ステップ 1: ソースファイルを作成して GitHub リポジトリにプッシュする 104
ステップ 2: パイプラインを作成する 105
ステップ 3: パイプラインを実行してビルドコマンドを検証する 108
チュートリアル: Git タグを使用してパイプラインを開始する 109
前提条件 110
ステップ 1: CloudShell を開いてリポジトリを複製する 111
ステップ 2: Git タグでトリガーするパイプラインを作成する 111
ステップ 3: リリースに対するコミットにタグを付ける 115
ステップ 4: 変更をリリースしてログを表示する 116
チュートリアル: プルリクエストのブランチ名をフィルタリングしてパイプラインを開始する (V2 タイプ) 117
前提条件 118
ステップ 1: 指定したブランチのプルリクエストに応じて開始するパイプラインを作成する 118
ステップ 2: GitHub.com でプルリクエストを作成してマージし、パイプライン実行を開始する 121
チュートリアル: パイプラインレベルの変数を使用する 122
前提条件 123
ステップ 1: パイプラインを作成してプロジェクトをビルドする 123
ステップ 2: 変更をリリースしてログを表示する 126
チュートリアル: シンプルなパイプラインを作成する (S3 バケット) 127
S3 バケットを作成する 128
Windows Server の Amazon EC2 インスタンスを作成し、CodeDeploy エージェントをインストールします。 130
CodeDeploy でアプリケーションを作成する 133
最初のパイプラインを作成する 135
別のステージを追加する 138
ステージ間の移行を有効または無効にする 145
リソースをクリーンアップする 146

チュートリアル: シンプルなパイプラインを作成する (CodeCommit リポジトリ) 147
新しい CodeCommit リポジトリの作成 148
コードのダウンロード、コミット、プッシュする 149
EC2 Linux インスタンスを作成して CodeDeploy エージェントをインストールする 152
CodeDeploy でアプリケーションを作成する 154
最初のパイプラインを作成する 155
CodeCommit リポジトリ内のコードを更新する 158
リソースをクリーンアップする 160
詳細情報 160
チュートリアル: 4 ステージのパイプラインを作成する 161
前提条件を満たす 163
パイプラインを作成する 167
ステージを追加する 169
リソースをクリーンアップする 172
チュートリアル: CloudWatch Events ルールをセットアップし、パイプラインの状態の変更の E メール通知を送信します。 173
Amazon SNS を使用してEメール通知を設定します。 174
CodePipeline の CloudWatch Events 通知ルールを作成します。 175
リソースをクリーンアップする 177
チュートリアル: を使用して Android アプリを構築およびテストする AWS Device Farm 177
Device Farm テストを使用するように CodePipeline を設定します。 178
チュートリアル: で iOS アプリをテストする AWS Device Farm 183
Device Farm テスト(例 Amazon S3) を使用するように CodePipeline を設定する 184
チュートリアル: Service Catalog にデプロイするパイプラインを作成する 189
オプション 1: 設定ファイルを使用しないで Service Catalog にデプロイする 191
オプション 2: 設定ファイルを使用して Service Catalog にデプロイする 196
チュートリアル: を使用してパイプラインを作成する AWS CloudFormation 201
例 1: を使用して AWS CodeCommit パイプラインを作成する AWS CloudFormation 201
例 2: を使用して Amazon S3 パイプラインを作成する AWS CloudFormation 204
チュートリアル: AWS CloudFormation デプロイアクションの変数を使用するパイプラインを作成する 207
前提条件: AWS CloudFormation サービスロールと CodeCommit リポジトリを作成する 208
ステップ 1: サンプル AWS CloudFormation テンプレートをダウンロード、編集、アップロードする 209
ステップ 2: パイプラインを作成する 210
ステップ 3: CloudFormation デプロイアクションを追加して変更セットを作成する 213

ステップ 4: 手動承認アクションを追加する 214
ステップ 5: 変更セットを実行するための CloudFormation デプロイアクションを追加する 214
ステップ 6: スタックを削除するための CloudFormation デプロイアクションを追加する 215
チュートリアル: CodePipeline を使用した Amazon ECS 標準デプロイ 216
前提条件 217
ステップ 1: ビルド仕様ファイルをソースリポジトリに追加する 220
ステップ 2: 継続的デプロイパイプラインを作成する 222
ステップ 3: CodeBuild ロールに Amazon ECR 権限を追加する 224
ステップ 4: パイプラインのテスト 224
チュートリアル: Amazon ECR ソース、ECS - CodeDeploy 間のデプロイでパイプラインを作成する 225
前提条件 227
ステップ 1: イメージを作成して Amazon ECR リポジトリにプッシュする 227
ステップ 2: タスク定義ソースファイルと AppSpec ソースファイルを作成して、CodeCommit リポジトリにプッシュする 228
ステップ 3: Application Load Balancer とターゲットグループを作成する 232
ステップ 4: Amazon ECS クラスターとサービスを作成する 235
ステップ 5: CodeDeploy アプリケーションとデプロイグループ (ECS コンピューティングプラットフォーム) を作成する 238
ステップ 6: パイプラインを作成する 239
ステップ 7: パイプラインに変更を加えてデプロイを確認する 244
チュートリアル: Amazon Alexa Skill をデプロイするパイプラインを作成する 244
前提条件 244
ステップ 1: Alexa デベロッパーサービス LWA セキュリティプロファイルを作成する 245
ステップ 2 : Alexa スキルのソースファイルを作成して CodeCommit リポジトリにプッシュします。 245
ステップ 3: ASK CLI コマンドを使用して更新トークンを作成する 247
ステップ 4: パイプラインを作成する 248
ステップ 5: 任意のソースファイルに変更を加えてデプロイを確認する 250
チュートリアル: Amazon S3 をデプロイプロバイダとして使用するパイプラインを作成する . 250
オプション 1: 静的ウェブサイトファイルを Amazon S3 にデプロイする 252
オプション 2 : 構築されたアーカイブファイルを S3 ソースバケットから Amazon S3 にデプロイする 257
チュートリアル: にアプリケーションを公開する AWS Serverless Application Repository 262
[開始する前に] 263

ステップ 1: buildspec.yml ファイルを作成する 264
ステップ 2: パイプラインを作成して設定する 264
ステップ 3: 発行アプリケーションをデプロイする 266
ステップ 4: 発行アクションを作成する 267
チュートリアル: CodePipeline を使用した Lambda 関数のデプロイ 268
前提条件 268
ステップ 1: サンプル Lambda 関数を作成する 269
ステップ 2: 関数ファイルをリポジトリにアップロードする 270
ステップ 3: パイプラインを作成する 270
ステップ 4: パイプラインのテスト 273
詳細情報 274
チュートリアル: Lambda 呼び出しアクションで変数を使用する 274
前提条件 275
ステップ 1: Lambda 関数を作成する 275
ステップ 2: Lambda 呼び出しアクションと手動承認アクションをパイプラインに追加する 278
チュートリアル: AWS Step Functions 呼び出しアクションを使用する 279
前提条件: シンプルなパイプラインを作成または選択する 280
ステップ 1: サンプルステートマシンを作成する 281
ステップ 2: パイプラインにステップ関数呼び出しアクションを追加する 281
チュートリアル: AppConfig をデプロイプロバイダとして使用するパイプラインを作成します。 282
前提条件 283
ステップ 1: AWS AppConfig リソースを作成する 283
ステップ 2 : ファイルを S3 ソースバケットにアップロードします。 284
ステップ 3: パイプラインを作成する 285
ステップ 4 : 任意のソースファイルに変更を加えてデプロイを確認します。 286
チュートリアル: CodeCommit パイプラインソースで完全なクローンを使用する 286
前提条件 288
ステップ 1: README ファイルを作成する 288
ステップ 2: パイプラインを作成してプロジェクトをビルドする 288
ステップ 3: 接続を使用するように CodeBuild サービスロールポリシーを更新する 292
ステップ 4: ビルド出力でリポジトリコマンドを表示する 293
チュートリアル:CodeCommit パイプラインソースでフルクローンを使用する 293
前提条件 294
ステップ 1 : README ファイルを作成する 294

ステップ 2: パイプラインを作成してプロジェクトをビルドする 295
ステップ 3: CodeBuild サービスロールポリシーを更新してリポジトリをクローンする 298
ステップ 4: 構築出力でリポジトリコマンドを表示する 298
チュートリアル: AWS CloudFormation StackSets デプロイアクションを使用してパイプラインを作成する 298
前提条件 299
ステップ 1: サンプル AWS CloudFormation テンプレートとパラメータファイルをアップロードする 300
ステップ 2: パイプラインを作成する 302
ステップ 3: 初期デプロイを表示する 305
ステップ 4: CloudFormationsStackInstances アクションを追加する 305
ステップ 5: デプロイのスタックセットリソースを表示する 306
ステップ 6: スタックセットを更新する 307
パイプラインの変数チェックルールを入力条件として作成する 307
前提条件 308
ステップ 1: サンプルのソースファイルを作成して GitHub リポジトリに追加する 309
ステップ 2: パイプラインを作成する 309
ステップ 2: ビルドステージを編集して条件とルールを追加する 312
ステップ 3: パイプラインを実行し、解決された変数を表示する 314
ベストプラクティスとユースケース 317
CodePipeline の使用方法例 317
Amazon S3 で CodePipeline を使用する AWS CodeCommit、および Amazon S3 AWS CodeDeploy 317
サードパーティーアクションプロバイダー (GitHub や Jenkins) で CodePipeline を使用する 318
CodePipeline を使用して、CodeBuild でコードをコンパイル、ビルド、テストする 319
CodePipeline で Amazon ECS を使用してクラウドにコンテナベースのアプリケーションを継続的に配信する 319
Elastic Beanstalk で CodePipeline を使用してクラウドにウェブアプリケーションを継続的にデリバリーする 319
で CodePipeline を使用して Lambda ベースのアプリケーションとサーバーレスアプリケーションの AWS Lambda 継続的な配信を行う 319
CloudFormation テンプレートで CodePipeline を使用してクラウドに継続的に配信する 320
Amazon VPC で CodePipeline を使用する 321
利用可能な状況 321
CodePipeline 用の VPC エンドポイントポリシーを作成する 322

VPC 設定のトラブルシューティング 323
ステージとアクションを使用して CI/CD パイプラインを定義する 324
パイプライン、ステージ、アクションを作成する 325
カスタムパイプラインを作成する (コンソール) 326
パイプラインを作成する (CLI) 339
静的テンプレートからパイプラインを作成する 345
パイプラインを編集する 352
パイプラインを編集する (コンソール) 353
パイプラインを編集する (AWS CLI) 356
パイプラインと詳細を表示する 361
パイプラインを表示する (コンソール) 361
パイプラインのアクションの詳細を表示する (コンソール) 366
パイプラインの ARN とサービスロール ARN (コンソール) を表示します。 369
パイプラインの詳細と履歴を表示する (CLI) 370
実行履歴でステージ条件のルール結果を表示する 371
パイプラインを削除します。 374
パイプラインを削除する (コンソール) 374
パイプラインを削除する (CLI) 374
他のアカウントのリソースを使用するパイプラインを作成する 376
前提条件: AWS KMS 暗号化キーを作成する 378
ステップ 1: アカウントポリシーおよびロールをセットアップする 379
ステップ 2: パイプラインを編集する 386
ポーリングパイプラインをイベントベースの変更検出の使用に移行する 389
ポーリングパイプラインを移行する方法 389
アカウント内のポーリングパイプラインの表示 391
CodeCommit ソースを使用してポーリングパイプラインを移行する 396
イベントに対応した S3 ソースを使用してポーリングパイプラインを移行する 418
S3 ソースと CloudTrail 証跡を使用してポーリングパイプラインを移行する 438
GitHub (OAuth アプリ経由) ソースアクションのポーリングパイプラインを接続に移行する 474
GitHub (OAuth アプリ経由) ソースアクションのポーリングパイプラインを Webhook に移行する 478
CodePipeline サービスロールを作成する 495
CodePipeline サービスロールを作成する (コンソール) 496
CodePipeline サービスロールを作成する (CLI) 496
リソースのタグ付け 499

パイプラインにタグ付けする 500
パイプラインにタグ付けする (コンソール) 500
パイプラインにタグ付けする (CLI) 502
通知ルールの作成 504
..... 508
ソースアクションを使用してファーストパーティーソースプロバイダーに接続する 511
Amazon ECR ソースアクションと EventBridge 511
Amazon ECR ソースに対する EventBridge ルールを作成する (コンソール) 512
Amazon ECR ソースに対する EventBridge ルールを作成する (CLI) 515
Amazon ECR ソースの EventBridge ルールを作成する (CloudFormation テンプレート) 518
EventBridge イベントを使用する Amazon S3 ソースアクションへの接続 523
イベントに対して有効な S3 ソースを使用してパイプラインを作成する (CLI) 524
イベントに対して S3 ソースを有効にしてパイプラインを作成する (CloudFormation テンプレート) 529
EventBridge と を使用する Amazon S3 ソースアクションへの接続 AWS CloudTrail 543
Amazon S3 ソースに対する EventBridge ルールを作成する (コンソール) 545
Amazon S3 ソースに対する EventBridge ルールを作成する (CLI) 548
Amazon S3 ソースの EventBridge ルールを作成する (CloudFormation テンプレート)
Amazon S3 554
CodeCommit ソースアクションと EventBridge 567
CodeCommit ソースに対する EventBridge ルールを作成する (コンソール) 568
CodeCommit ソースに対する EventBridge ルールを作成する (CLI) 571
CodeCommit ソースの EventBridge ルールを作成する (CloudFormation テンプレート) 576
を使用してパイプラインにサードパーティーのソースプロバイダーを追加する AWS
CodeConnections 584
Azure DevOps 接続 584
Azure DevOps への接続を作成する (コンソール) 586
Azure DevOps への接続を作成する (CLI) 590
Bitbucket Cloud への接続 592
Bitbucket Cloud への接続を作成する (コンソール) 594
Bitbucket Cloud への接続を作成する (CLI) 597
GitHub コネクション 599
GitHub (コンソール) への接続を作成する 601
GitHub (CLI) への接続を作成する 604
GitHub Enterprise Server 接続 606
GitHub Enterprise Server への接続を作成する (コンソール) 607

GitHub Enterprise Server (CLI) への接続を作成します。 611
GitLab.com への接続 614
GitLab.com への接続を作成する (コンソール) 616
GitLab.com への接続を作成する (CLI) 620
GitLab セルフマネージドの接続 622
GitLab セルフマネージドへの接続を作成する (コンソール) 624
GitLab セルフマネージドへのホストと接続を作成する (CLI) 628
別のアカウントと共有されている接続を使用する 630
トリガーとフィルタリングを使用してパイプラインを自動的に開始する 632
トリガーフィルターに関する考慮事項 634
プロバイダーによるトリガーのプルリクエストイベント 635
トリガーフィルターの例 636
フィルターを使用せずにコードプッシュにトリガーを追加する 645
コードプッシュまたはプルリクエストイベントタイプを使用してトリガーを追加する 645
プッシュリクエストイベントタイプとプルリクエストイベントタイプのフィルターを追加する (コンソール) 647
プッシュリクエストイベントタイプとプルリクエストイベントタイプのフィルターを追加する (CLI) 649
プッシュおよびプルリクエストイベントタイプのフィルターを追加する (CloudFormation テンプレート) 652
トリガーを追加して変更検出をオフにする 653
パイプラインを手動で開始および停止する 655
CodePipeline でパイプラインを編集する 655
パイプラインを手動で開始する 657
スケジュールに基づいたパイプラインの開始 658
ソースリビジョンオーバーライドでパイプラインを開始する 662
パイプライン実行を停止する 665
パイプライン実行を停止する (コンソール) 666
インバウンド実行を停止します (コンソール)。 669
パイプライン実行を停止する (CLI) 669
インバウンド実行 (CLI) を停止します。 671
パイプライン実行の履歴を表示し、モードを設定する 673
実行を表示する 673
パイプライン実行の履歴を表示する (コンソール) 673
実行のステータスを表示する (コンソール) 675
インバウンドの実行(コンソール)を表示します。 677

パイプライン実行ソースのリビジョンを表示する (コンソール) 678
アクション実行を表示する (コンソール) 680
アクションアーティファクトとアーティファクトストア情報を表示する (コンソール) 681
パイプラインの詳細と履歴を表示する (CLI) 681
パイプライン実行モードを設定または変更する 693
実行モードの表示に関する考慮事項 694
実行モード間を切り替える場合の考慮事項 697
パイプライン実行モードを設定または変更する (コンソール) 698
パイプライン実行モードを設定する (CLI) 699
ステージをロールバックまたは再試行する 703
失敗したステージまたは失敗したアクションのステージ再試行の設定 703
ステージ再試行に関する考慮事項 704
失敗したステージを手動で再試行する 704
ステージ障害時の自動再試行を設定する 708
ステージロールバックの設定 714
ロールバックに関する考慮事項 714
ステージを手動でロールバックする 714
ステージの自動ロールバックを設定する 720
実行リストでロールバックステータスを表示する 724
ロールバックステータスの詳細を表示する 727
ステージの条件を設定する 732
ステージ条件のユースケース 733
ステージ条件に設定する結果に関する考慮事項 733
ステージ条件に設定するルールに関する考慮事項 734
入力条件の作成 734
入力条件の作成 - CloudWatchAlarm ルールの例 (コンソール) 735
スキップ結果と VariableCheck ルールを使用した入力条件の作成 (コンソール) 736
入力条件の作成 (CLI) 738
入力条件の作成 (CFN) 740
失敗時の条件の作成 741
失敗時の条件の作成 (コンソール) 741
再試行結果の例を使用した OnFailure 条件の作成 (コンソール) 742
失敗時の条件の作成 (CLI) 743
失敗時の条件の作成 (CFN) 745
成功時の条件の作成 746
成功時の条件の作成 (コンソール) 746

成功時の条件の作成 (CLI) ... 748
成功時の条件を作成する (CFN) ... 750
ステージ条件の削除 ... 751
ステージ条件の上書き ... 752
アクションタイプ、カスタムアクション、および承認アクションを使用する ... 754
アクションタイプの使用 ... 754
アクションタイプをリクエストする ... 756
使用可能なアクションタイプをパイプラインに追加する (コンソール) ... 762
アクションタイプを表示する ... 764
アクションタイプを更新する ... 765
パイプラインのカスタムアクションを作成する ... 767
カスタムアクションを作成する ... 769
カスタムアクションのジョブワーカーを作成する ... 773
パイプラインにカスタムアクションを追加する ... 781
CodePipeline でカスタムアクションにタグ付けする ... 784
カスタムアクションにタグを追加する ... 784
カスタムアクションのタグを表示する ... 785
カスタムアクションのタグを編集する ... 785
カスタムアクションからタグを削除する ... 785
パイプラインで Lambda 関数を呼び出す ... 786
ステップ 1: パイプラインを作成する ... 788
ステップ 2 : Lambda 関数を作成する ... 789
ステップ 3: CodePipeline コンソールでパイプラインに Lambda 関数を追加する ... 794
ステップ 4 : Lambda 関数でパイプラインをテストする ... 795
ステップ 5: 次のステップ ... 795
JSON イベントの例 ... 796
追加のサンプル関数 ... 798
手動の承認アクションをステージに追加する ... 811
手動の承認アクションに関する設定オプション ... 811
承認アクションのセットアップおよびワークフローの概要 ... 812
CodePipeline で IAM ユーザーに承認アクセス許可を付与する ... 813
サービスロールへの Amazon SNS アクセス許可の付与 ... 816
手動の承認アクションを追加する ... 818
承認アクションを承認または拒否する ... 822
手動の承認通知の JSON データ形式 ... 826
クロスリージョンアクションをパイプラインに追加する ... 827

パイプライン内のクロスリージョンアクションを管理する (コンソール) 829
クロスリージョンアクションをパイプラインに追加する (CLI) 832
クロスリージョンアクションをパイプラインに追加する (CloudFormation) 837
変数の操作 840
変数のアクションを設定する 841
出力変数を表示する 845
例: 手動承認で変数を使用する 848
例:CodeBuild 環境変数で BranchName 変数を使用する 848
ステージ移行の操作 851
移行を無効化または有効化する (コンソール) 851
移行を無効化または有効化する (CLI) 853
パイプラインのモニタリング 855
CodePipeline イベントのモニタリング 856
詳細タイプ 857
パイプラインレベルのイベント 860
ステージレベルのイベント 868
アクションレベルのイベント 872
パイプラインイベントで通知を送信するルールを作成する 880
イベントのプレースホルダーバケットに関するリファレンス 884
イベントのプレースホルダーバケット名 (リージョン別) 885
AWS CloudTrail を使用した API コールのログ記録 888
CloudTrail での CodePipeline 情報 889
CodePipeline ログファイルエントリについて 890
CodePipeline CloudWatch メトリクス 892
PipelineDuration 893
FailedPipelineExecutions 893
トラブルシューティング 894
パイプラインエラー: で設定されたパイプラインは、「デプロイに失敗しました」というエラーメッセージ AWS Elastic Beanstalk を返します。提供されたロールに十分なアクセス権限がありません: サービス: AmazonElasticLoadBalancing」 895
デプロイエラー: AWS Elastic Beanstalk デプロイアクションで設定されたパイプラインは、DescribeEvents」アクセス許可がない場合に失敗するのではなくハングします 896
パイプラインのエラー: ソースアクションは次のようなアクセス許可の不足メッセージを返します。「CodeCommit リポジトリ `repository-name` にアクセスできませんでした。」 リポジトリにアクセスするための十分な権限がパイプラインの IAM ロールにあることを確認してください。」 896

パイプラインのエラー: Jenkins のビルドまたはテストアクションが長期間実行された後、認証情報やアクセス許可の不足のため失敗します。 897
パイプラインエラー: 別の AWS リージョンで作成されたバケットを使用してある AWS リージョンで作成されたパイプラインは、JobFailed」というコードのInternalError」を返します。 897
デプロイエラー: WAR ファイルを含む ZIP ファイルは正常にデプロイされますが AWS Elastic Beanstalk、アプリケーション URL は 404 が見つかりませんという エラーを報告します 896
パイプラインのアーティファクトフォルダ名が切り詰められているように見えます 898
Bitbucket、GitHub、GitHub Enterprise Server、または GitLab.com に接続するための CodeBuild GitClone アクセス許可を追加します。 899
CodeBuild GitClone のアクセス権限を CodeCommit ソースアクションに追加します。 901
パイプラインのエラー: CodeDeployToECS アクションがあるデプロイから、「<source artifact name> からタスク定義アーティファクトファイルを読み取ろうとしたときに例外が発生しました」というエラーメッセージが返されます。 902
GitHub (OAuth アプリ経由) ソースアクション: リポジトリリストに異なるリポジトリが表示されます。 902
GitHub (GitHub アプリ経由) ソースアクション: リポジトリの接続を完了できません。 903
Amazon S3 エラー: CodePipeline サービスロール <ARN> により、S3 バケット <BucketName> に対する S3 アクセスが拒否されました。 903
Amazon S3 、Amazon ECR、または CodeCommit ソースを使用した パイプラインは自動的に起動されなくなりました。 906
GitHub への接続時の Connections エラー:「問題が発生しました。ブラウザで Cookie が有効になっていることを確認してください」または「組織の所有者は GitHub アプリケーションをインストールする必要があります」 907
実行モードが QUEUED モードまたは PARALLEL モードに変更されたパイプラインは、実行制限に達すると失敗します 908
PARALLEL モードのパイプラインは、QUEUED モードまたは SUPERSEDED モードに変更して編集したときに、パイプライン定義が古いままになります。 908
PARALLEL モードから変更したパイプラインに、以前の実行モードが表示されます。 908
ファイルパスによるトリガーフィルタリングを使用する接続を持つパイプラインは、ブランチの作成時に開始しない可能性があります 909
ファイルパスによるトリガーフィルタリングを使用する接続を持つパイプラインは、ファイル制限に達したときに開始しない場合があります 909
PARALLEL モードの CodeCommit または S3 ソースリビジョンは、EventBridge イベントと一致しない可能性があります 910
EC2 デプロイアクションがエラーメッセージ `No such file` で失敗する 910

EKS デプロイアクションがエラーメッセージ cluster unreachable で失敗する 911
複数のソースアクションが同じリポジトリを参照している場合、パイプラインはすべてのブランチに対してトリガーされません 912
別の問題があるため問い合わせ先を教えてください。 912
セキュリティ 914
データ保護 915
ネットワーク間のトラフィックのプライバシー 916
保管中の暗号化 916
転送中の暗号化 917
暗号化キーの管理 917
CodePipeline 用に Amazon S3 に保存したアーティファクトのサーバー側の暗号化を設定する 917
AWS Secrets Manager を使用してデータベースパスワードまたはサードパーティー API キーを追跡する 920
ID とアクセス管理 921
オーディエンス 922
アイデンティティを使用した認証 922
ポリシーを使用したアクセス管理 923
が IAM と AWS CodePipeline 連携する方法 925
アイデンティティベースのポリシーの例 930
リソースベースのポリシーの例 959
トラブルシューティング 960
アクセス許可に関するリファレンス 963
CodePipeline サービスロールを管理する 973
インシデント対応 980
コンプライアンス検証 981
耐障害性 981
インフラストラクチャセキュリティ 982
セキュリティのベストプラクティス 982
パイプライン構造リファレンス 984
パイプライン宣言 988
name 990
roleArn 991
artifactStore または artifactStores 991
stages 992
version 993

executionMode ... 993
pipelineType ... 993
variables ... 994
triggers ... 994
metadata ... 998
ステージ宣言 ... 999
name ... 1002
actions ... 1002
conditions ... 1002
rules ... 1003
アクションの宣言 ... 1003
name ... 1007
region ... 1007
roleArn ... 1007
namespace ... 1008
actionTypeId ... 1008
InputArtifacts ... 1009
outputArtifacts ... 1010
configuration (アクションプロバイダー別) ... 1011
runOrder ... 1013
CodePipeline の有効なアクションプロバイダー ... 1014
PollForSourceChanges パラメータの有効な設定 ... 1019
アクションタイプ別の有効な入力/出力アーティファクトの数 ... 1021
プロバイダータイプ別の有効な設定パラメータ ... 1023
アクション構造リファレンス ... 1028
Amazon EC2 アクションリファレンス ... 1029
アクションタイプ ... 1030
設定パラメータ ... 1030
入力アーティファクト ... 1034
出力アーティファクト ... 1035
EC2 デプロイアクションのサービスロールポリシーのアクセス許可 ... 1035
仕様ファイルのリファレンスをデプロイする ... 1037
アクションの宣言 ... 1039
デプロイ仕様を使用したアクション宣言の例 ... 1040
関連情報 ... 1042
Amazon ECR ソースアクションリファレンス ... 1042

アクションタイプ 1043
設定パラメータ 1043
入力アーティファクト 1044
出力アーティファクト 1044
出力変数 1044
サービスロールのアクセス許可: Amazon ECR アクション 1045
アクションの宣言 (Amazon ECR の例) 1045
関連情報 1046
ECRBuildAndPublish ビルドアクションリファレンス 1047
アクションタイプ 1048
設定パラメータ 1048
入力アーティファクト 1049
出力アーティファクト 1049
出力変数 1049
サービスロールのアクセス許可: ECRBuildAndPublish アクション 1050
アクションの宣言 1051
関連情報 1052
Amazon ECS および CodeDeploy ブルー/グリーンデプロイアクションリファレンス 1053
アクションタイプ 1054
設定パラメータ 1054
入力アーティファクト 1056
出力アーティファクト 1057
サービスロールのアクセス許可: CodeDeployToECS アクション 1057
アクションの宣言 1059
関連情報 1060
Amazon Elastic Container Service デプロイアクションリファレンス 1061
アクションタイプ 1062
設定パラメータ 1063
入力アーティファクト 1063
出力アーティファクト 1064
サービスロールのアクセス許可: Amazon ECS 標準アクション 1064
アクションの宣言 1066
関連情報 1067
Amazon Elastic Kubernetes Service の EKS デプロイアクションリファレンス 1068
アクションタイプ 1069
設定パラメータ 1069

入力アーティファクト 1070
出力アーティファクト 1071
環境変数 1071
出力変数 1071
サービスロールのポリシーのアクセス許可 1071
アクションの宣言 1074
関連情報 1075
AWS Lambda デプロイアクションリファレンス 1075
アクションタイプ 1076
設定パラメータ 1076
入力アーティファクト 1079
出力アーティファクト 1079
出力変数 1079
Lambda デプロイアクションのサービスロールポリシーのアクセス許可 1079
アクションの宣言 1081
関連情報 1082
Amazon S3 デプロイアクションリファレンス 1082
アクションタイプ 1083
設定パラメータ 1083
入力アーティファクト 1085
出力アーティファクト 1085
サービスロールのアクセス許可: S3 デプロイアクション 1085
アクション設定の例 1086
関連情報 1089
Amazon S3 ソースアクションリファレンス 1089
アクションタイプ 1090
設定パラメータ 1090
入力アーティファクト 1092
出力アーティファクト 1092
出力変数 1093
サービスロールのアクセス許可: S3 ソースアクション 1093
アクションの宣言 1094
関連情報 1095
AWS AppConfig デプロイアクションリファレンス 1095
アクションタイプ 1096
設定パラメータ 1096

入力アーティファクト 1097
出力アーティファクト 1097
サービスロールのアクセス許可: AppConfig アクション 1097
アクション設定の例 1098
関連情報 1099
CloudFormation デプロイアクションリファレンス 1099
アクションタイプ 1100
設定パラメータ 1100
入力アーティファクト 1105
出力アーティファクト 1105
出力変数 1106
サービスロールのアクセス許可: CloudFormation アクション 1106
アクションの宣言 1108
関連情報 1109
CloudFormation StackSets 1109
Stack CloudFormation StackSets アクションの仕組み 1111
パイプラインで StackSets アクションを構成する方法 1113
CloudFormationStackSet アクション 1114
CloudFormationStackInstances アクション 1128
サービスロールのアクセス許可: CloudFormationStackSet アクション 1138
サービスロールのアクセス許可: CloudFormationStackInstances アクション 1138
スタックセットオペレーションのアクセス許可モデル 1138
テンプレートパラメータのデータタイプ 1139
関連情報 1109
AWS CodeBuild ビルドおよびテストアクションリファレンス 1141
アクションタイプ 1142
設定パラメータ 1142
入力アーティファクト 1145
出力アーティファクト 1146
出力変数 1147
サービスロールのアクセス許可: CodeBuild アクション 1147
アクション宣言 (CodeBuild の例) 1148
関連情報 1149
AWS CodePipeline アクションリファレンスを呼び出す 1150
アクションタイプ 1150
設定パラメータ 1151

入力アーティファクト 1153
出力アーティファクト 1154
CodePipeline 呼び出しアクションのサービスロールポリシーのアクセス許可 1154
アクションの宣言 1154
関連情報 1155
AWS CodeCommit ソースアクションリファレンス 1155
アクションタイプ 1156
設定パラメータ 1157
入力アーティファクト 1158
出力アーティファクト 1158
出力変数 1159
サービスロールのアクセス許可: CodeCommit アクション 1160
アクション設定の例 1160
関連情報 1163
AWS CodeDeploy デプロイアクションリファレンス 1163
アクションタイプ 1164
設定パラメータ 1164
入力アーティファクト 1164
出力アーティファクト 1164
サービスロールのアクセス許可: AWS CodeDeploy アクション 1165
アクションの宣言 1166
関連情報 1167
CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアクションの場合) 1168
アクションタイプ 1172
設定パラメータ 1172
入力アーティファクト 1173
出力アーティファクト 1173
出力変数 1174
サービスロールのアクセス許可: AWS CodeConnections アクション 1175
アクションの宣言 1175
インストールアプリケーションのインストールと接続の作成 1177
関連情報 1177
コマンドアクションリファレンス 1178
コマンドアクションに関する考慮事項 1179
サービスロールのポリシーのアクセス許可 1180

アクションタイプ 1181
設定パラメータ 1181
入力アーティファクト 1186
出力アーティファクト 1186
環境変数 1186
サービスロールのアクセス許可: コマンドアクション 1188
アクションの宣言 (例) 1188
関連情報 1190
AWS Device Farm テストアクションリファレンス 1190
アクションタイプ 1190
設定パラメータ 1191
入力アーティファクト 1195
出力アーティファクト 1195
サービスロールのアクセス許可: AWS Device Farm アクション 1195
アクションの宣言 1196
関連情報 1197
Elastic Beanstalk のデプロイアクションリファレンス 1198
アクションタイプ 1198
設定パラメータ 1198
入力アーティファクト 1199
出力アーティファクト 1199
サービスロールのアクセス許可: ElasticBeanstalk のデプロイアクション 1199
アクションの宣言 1200
関連情報 1201
Amazon Inspector InspectorScan 呼び出しアクションリファレンス 1201
アクションタイプ ID 1202
設定パラメータ 1203
入力アーティファクト 1205
出力アーティファクト 1205
出力変数 1205
サービスロールのアクセス許可: InspectorScan アクション 1205
アクションの宣言 1206
関連情報 1207
AWS Lambda アクションリファレンスを呼び出す 1207
アクションタイプ 1208
設定パラメータ 1208

入力アーティファクト 1209
出力アーティファクト 1209
出力変数 1209
アクション設定の例 1209
JSON イベントの例 1210
関連情報 1213
AWS OpsWorks デプロイアクションリファレンス 1213
アクションタイプ 1213
設定パラメータ 1213
入力アーティファクト 1214
出力アーティファクト 1214
サービスロールのアクセス許可: AWS OpsWorks アクション 1214
アクション設定の例 1215
関連情報 1216
AWS Service Catalog デプロイアクションリファレンス 1216
アクションタイプ 1216
設定パラメータ 1216
入力アーティファクト 1217
出力アーティファクト 1217
サービスロールのアクセス許可: Service Catalog アクション 1217
設定ファイルのタイプ別のアクション設定の例 1218
関連情報 1219
AWS Step Functions 1219
アクションタイプ 1213
設定パラメータ 1220
入力アーティファクト 1222
出力アーティファクト 1222
出力変数 1222
サービスロールのアクセス許可: StepFunctions アクション 1222
アクション設定の例 1223
行動 1226
関連情報 1099
ルール構造リファレンス 1229
CloudwatchAlarm 1229
ルールタイプ 1230
設定パラメータ 1230

ルール設定の例 1230
関連情報 1231
CodeBuild ルール 1232
サービスロールのポリシーのアクセス許可 1232
ルールタイプ 1233
設定パラメータ 1233
ルール設定の例 1235
関連情報 1236
コマンド 1236
コマンドルールに関する考慮事項 1236
サービスロールのポリシーのアクセス許可 1232
ルールタイプ 1233
設定パラメータ 1233
ルール設定の例 1235
関連情報 1236
DeploymentWindow 1241
ルールタイプ 1242
設定パラメータ 1242
ルール設定の例 1244
関連情報 1245
LambdaInvoke 1245
ルールタイプ 1230
設定パラメータ 1246
ルール設定の例 1246
関連情報 1247
VariableCheck 1247
ルールタイプ 1248
設定パラメータ 1248
ルール設定の例 1250
関連情報 1251
統合モデルのリファレンス 1252
インテグレータとサードパーティーのアクションタイプがどのように機能するか 1252
概念 1253
サポートされている統合モデル 1255
Lambda 統合モデル 1256
Lambda 関数を更新して CodePipeline からの入力を処理します。 1256

Lambda 関数の結果を CodePipeline に返す 1261
継続トークンを使用して、非同期プロセスの結果を待つ 1263
ランタイム時にインテグレーターの Lambda 関数を呼び出す権限を CodePipeline に提供します。 1263
ジョブワーカー統合モデル 1263
ジョブワーカー用にアクセス許可管理戦略を選択して設定する 1264
イメージ定義ファイルのリファレンス 1267
Amazon ECS 標準デプロイアクション用の imagedefinitions.json ファイル 1267
Amazon ECS Blue/Green デプロイアクション用の imageDetail.json ファイル 1270
変数リファレンス 1275
概念 1276
[変数] 1276
名前空間 1277
変数のユースケース 1278
変数の設定 1279
パイプラインレベルの変数を設定する 1279
アクションレベルの変数の設定 1280
変数の解決 1282
変数のルール 1283
パイプラインアクションで使用できる変数 1284
定義された変数キーを持つアクション 1284
ユーザー設定の変数キーを使用したアクション 1290
構文での glob パターンの使用 1292
ポーリングパイプラインを推奨される変更検出方法に更新する 1294
GitHub (OAuth アプリ経由) ソースアクションを GitHub (GitHub アプリ経由) ソースアクションに更新する 1295
ステップ 1: (OAuth アプリ経由) GitHub アクションを置き換える 1296
ステップ 2 ：GitHub への接続を作成する 1297
ステップ 3: GitHub のソースアクションを保存する 1298
クォータ 1300
付録 A: GitHub (OAuth アプリ経由) ソースアクション 1316
GitHub (OAuth アプリ経由) ソースアクションの追加 1317
GitHub (OAuth アプリ経由) ソースアクションリファレンス 1318
アクションタイプ 1319
設定パラメータ 1319
入力アーティファクト 1321

出力アーティファクト 1321
出力変数 1321
アクションの宣言 (GitHub の例) 1322
GitHub (OAuth) への接続 1323
関連情報 1324
ドキュメント履歴 1325
以前の更新 1360
CodePipeline 機能リファレンス 1372
..... mccclxxv

とは AWS CodePipeline

AWS CodePipeline は、ソフトウェアのリリースに必要なステップをモデル化、視覚化、自動化するために使用できる継続的な配信サービスです。ソフトウェアリリースプロセスのさまざまなステージを素早くモデル化して設定できます。CodePipeline はソフトウェアの変更を継続的にリリースするために必要なステップを自動化します。CodePipeline の料金については、コスト を参照してください。

トピック

- 継続的デリバリーと継続的インテグレーション
- CodePipeline で何ができますか。
- CodePipeline のクイックルック
- CodePipeline を使い始めるにはどうすればよいですか。
- CodePipeline の概念
- DevOps パイプラインの例
- パイプライン実行の仕組み
- 入力および出力アーティファクト
- ステージ条件はどのように機能しますか?
- パイプラインのタイプ
- 適切なパイプラインのタイプの選択

継続的デリバリーと継続的インテグレーション

CodePipeline は、ソフトウェアの構築、テスト、製品へのデプロイを自動化する 継続的デリバリー サービス です。

「継続的な配信」はリリースプロセスが自動化されるソフトウェア開発方法です。すべてのソフトウェア変更は自動的に構築され、テストされ、本番稼働用にデプロイされます。最終的な本番稼働に進む前に、個人、自動テスト、またはビジネスルールが最終的なプッシュがいつ行われるかを決定します。正常なすべてのソフトウェア変更は、継続的な配信ですぐに本番稼働にリリースできますが、すべての変更をすぐにリリースする必要はありません。

継続的統合 は、チームのメンバーがバージョン管理システムを使用し、頻繁にマスターブランチなどの同じ場所に作業を統合するソフトウェア開発のプラクティスです。各変更は、可能な限り迅速に

統合エラーを検出するために構築され、検証されます。継続的な統合は、ソフトウェアのリリースプロセス全体を本番稼働まで自動化する継続的な配信と比較して、コードの自動構築とテストに重点を置いています。

詳細については、「継続的インテグレーションと継続的デリバリーの実践 AWS: DevOps によるソフトウェアデリバリーの加速」を参照してください。

CodePipeline コンソール、 AWS Command Line Interface (AWS CLI)、 AWS SDKs、またはこれらの任意の組み合わせを使用して、パイプラインを作成および管理できます。

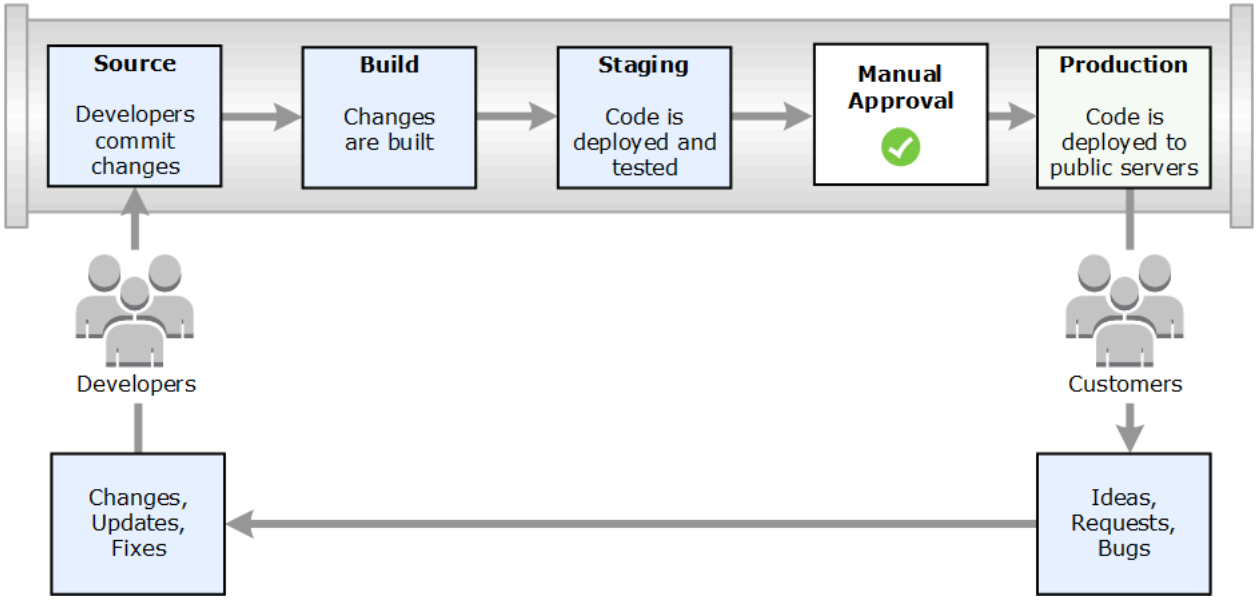
CodePipeline で何ができますか。

CodePipeline を使用すると、アプリケーションをクラウドで自動的に構築、テスト、およびデプロイするのに役立ちます。具体的な内容は以下のとおりです：

- リリースプロセスの自動化: CodePipeline は、ソースリポジトリから構築、テスト、デプロイまで、リリースプロセスを端末間で完全に自動化します。ソースステージ以外の任意のステージで手動承認アクションを含めることで、パイプラインを通して変更が移動しないようにすることができます。選択したシステムで、1 つのインスタンスまたは複数のインスタンス間で、任意の方法を使用して、必要に応じてリリースできます。
- 一貫性のあるリリースプロセスを確立する：コードを変更するたびに一貫性のある一連のステップを定義します。CodePipeline はお客様の基準に従ってリリースの各ステージを実行します。
- 品質を向上しながら配信を高速化: リリースプロセスを自動化して、開発者がコードを段階的にテストおよびリリースし、新しい機能のリリースを顧客に迅速に提供できるようにすることができます。
- お好みのツールを使用: 既存のソース、ビルド、およびデプロイツールをパイプラインに組み込むことができます。CodePipeline で現在サポートされている AWS のサービス およびサードパーティーツールの完全なリストについては、「」を参照してくださいCodePipeline との製品とサービスの統合。
- 進捗状況を一目で確認: パイプラインのリアルタイムステータスの確認、アラート詳細の確認、失敗したステージまたはアクションの再試行、各ステージで最新のパイプライン実行で使用されたソースリビジョンの詳細の表示、手動でのパイプラインの再実行が可能です。
- パイプライン履歴の詳細を表示: 開始時刻と終了時刻、継続時間、実行 ID など、パイプラインの実行の詳細を表示できます。

CodePipeline のクイックルック

次の図表は、CodePipeline を使用したリリースプロセスの例を示しています。



この例では、デベロッパーがソースリポジトリに変更をコミットすると、CodePipeline は自動的に変更を検出します。これらの変更が作成され、テストが設定されている場合は、それらのテストが実行されます。テストが完了すると、ビルドされたコードがテスト用のステージングサーバーにデプロイされます。ステージングサーバーから、CodePipeline は統合やロードなどの色々なテストを実行します。これらのテストが正常に完了し、パイプラインに追加された手動承認アクションが承認された後、CodePipeline はテスト済みと承認済みコードを製品インスタンスにデプロイします。

CodePipeline は、CodeDeploy、 AWS Elastic Beanstalk、または を使用してアプリケーションを EC2 インスタンスにデプロイできます AWS OpsWorks Stacks。CodePipeline は、Amazon ECS を使用してコンテナベースのアプリケーションをサービスにデプロイすることもできます。デベロッパーは、CodePipeline で提供される統合ポイントを使用して、構築サービス、テストプロバイダー、その他のデプロイターゲットやシステムなど、他のツールやサービスをプラグインすることもできます。

パイプラインは、リリースプロセスが必要とするのと同じくらいシンプルでも複雑でもかまいません。

CodePipeline を使い始めるにはどうすればよいですか。

CodePipeline の使用を開始するには

1. CodePipeline の概念 セクションを読んで、CodePipeline がどのように機能するかを 学びます。
2. CodePipeline の使用開始 のステップに従って、CodePipeline の使用のための 準備をします。

3. CodePipeline チュートリアル チュートリアルのステップに従って、CodePipeline を 試してください。
4. パイプライン、ステージ、アクションを作成する の手順に従って、新規または既存のプロジェクトに CodePipeline を 使用します。

CodePipeline の概念

で使用される概念と用語を理解すれば、自動リリースプロセスのモデリングと設定が簡単になります AWS CodePipeline。ここでは、 CodePipeline を使用する際に知っておかなければならないいくつかの概念を次に示します。

DevOps パイプラインの例については、「DevOps パイプラインの例」を参照してください。

CodePipeline では、次の用語が使用されます：

トピック

- Pipelines
- パイプライン実行
- ステージオペレーション
- アクション実行
- 実行タイプ
- アクションタイプ
- アーティファクト
- ソースリビジョン
- トリガ
- [変数]
- 条件
- Rules

Pipelines

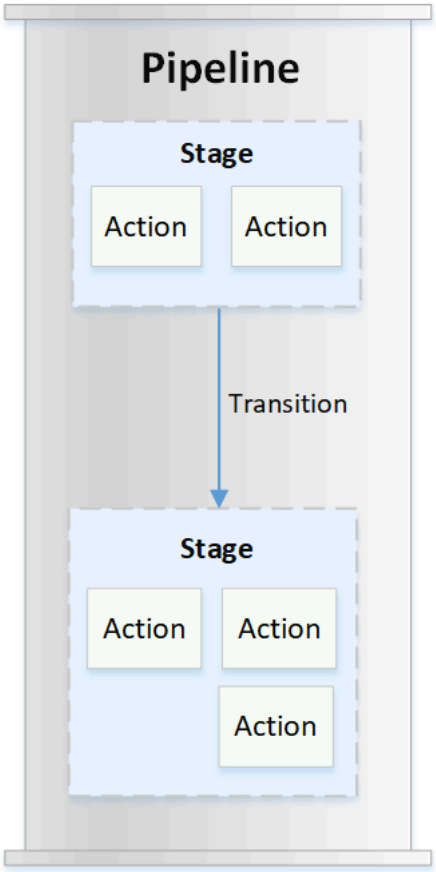
パイプラインは、ソフトウェアの変更がリリースプロセスをどのように通過するかを記述するワークフロー構造です。各パイプラインは一連のステージで構成されています。

Stages

ステージは、環境を分離し、その環境での同時変更の数を制限するために使用できる論理ユニットです。各ステージには、アプリケーションアーティファクトに対して実行されるアクションが含まれます。ソースコードはアーティファクトの例です。ステージは、ソースコードが構築され、テストが実行されるビルドステージである場合もあれば、コードをランタイム環境にデプロイするデプロイステージの場合もあります。各ステージは、連続または並列のアクションで構成されています。

Transitions

トランジション は、パイプライン実行がパイプラインの次のステージに移動するポイントです。ステージのインバウンドトランジションを無効にして、実行がそのステージに入らないようにし、そのトランジションを有効にして実行を継続することができます。無効なトランジションで複数の実行が到着した場合、トランジションが有効になると、最新の実行だけが次のステージに進みます。つまり、トランジションが無効になっている間は、より新しい実行が待機中の実行よりも優先され、トランジションが有効になった後は継続する実行が優先されます。



アクション

アクションは、アプリケーションコードに対して実行される一連の操作であり、アクションがパイプライン内で指定されたポイントで実行されるように設定されます。これには、コード変更によるソースアクション、インスタンスにアプリケーションをデプロイするためのアクションなどが含まれます。たとえば、デプロイステージには、 AWS Lambdaや Amazon EC2 などのコンピューティングサービスにコードをデプロイするデプロイアクションが含まれている場合があります。

有効な CodePipeline アクションタイプは次のとおりです。`source`、`build`、`test`、`deploy`、`approval`、および `invoke`。アクションプロバイダーのリストについては、「CodePipeline の有効なアクションプロバイダー」を参照してください。

アクションは、直列または並列で実行できます。ステージ内のシリアルアクションとパラレルアクションの詳細については、アクション構造の要件 の `runOrder` の情報を参照してください。

パイプライン実行

実行は、パイプラインによってリリースされる一連の変更です。各パイプライン実行は一意であり、独自の ID を持ちます。実行は、マージされたコミットや最新のコミットの手動リリースなど、一連の変更に対応します。2 つの実行では、同じ変更セットを異なる時間に解放できます。

パイプラインは同時に複数の実行を処理できますが、パイプラインステージは一度に 1 つの実行のみを処理します。これを行うために、ステージは実行を処理している間ロックされます。2 つのパイプライン実行は、同時に同じステージを占めることはできません。占有ステージに入るのを待っている実行は、インバウンド実行です。インバウンドの実行は、失敗したり、置き換えたり、手動で停止したりする可能性があります。インバウンドの実行の詳細については、「インバウンド実行の仕組み」を参照してください。

パイプラインの実行は、パイプラインのステージを順番に通過します。パイプラインの有効なステータスは、`InProgress`、`Stopping`、`Stopped`、`Succeeded`、`Superseded`、`Failed` です。

詳細については、「PipelineExecution」を参照してください。

停止された実行

パイプライン実行を手動で停止して、進行中のパイプライン実行がパイプラインを介して続行されないようにすることができます。手動で停止した場合、完全に停止するまでパイプライン実行には `Stopping` ステータスが表示されます。次に、`Stopped` ステータスが表示されます。`Stopped` パイプラインの実行を再試行できます。

パイプラインの実行を停止する方法は 2 つあります。

- [Stop and wait (停止して待機)]
- [Stop and abandon (停止して中止)]

実行を停止するユースケースおよびこれらのオプションのシーケンスの詳細については、「パイプライン実行の停止方法」を参照してください 。

失敗した実行

実行が失敗した場合、実行は停止し、パイプラインを完全に通過しません。ステータスは に変わりFAILED、CodePipeline はステージをロック解除します。より最近の実行が、追いついてロック解除されたステージに入り、ステージをロックすることができます。失敗した実行が置き換えられているか、再試行可能でない場合を除き、失敗した実行を再試行できます。失敗したステージを以前の成功した実行にロールバックできます。

実行モード

パイプラインを介して最新の変更セットを配信するため、より新しい実行が、パイプラインを経由してすでに実行されている最新ではない実行をパスし、置き換えます。これが発生すると、古い実行は新しい実行に置き換えられます。実行は、ステージ間のポイントである特定の時点で、より最新の実行に置き換えることができます。SUPERSEDED はデフォルトの実行モードです。

SUPERSEDED モードでは、ロックされたステージに入るまで実行が待機している間に、より新しい実行が追いつき、待機中の実行に置き換わる場合があります。より新しい実行はステージのロックが解除されるまで待機し、置き換えられた実行は SUPERSEDED ステータスで停止します。パイプライン実行が置き換えられると、実行は停止し、パイプラインを完全に通過しません。このステージで置き換えられた後に、置き換えられた実行を再試行することはできません。その他の使用可能な実行モードは、PARALLEL モードまたは QUEUED モードです。

実行モードとロックされたステージの詳細については、「SUPERSEDED モードでの実行の処理方法」を参照してください。

ステージオペレーション

パイプライン実行がステージを通過する場合、ステージは、ステージ内のすべてのアクションを完了するプロセスに入ります。ステージオペレーションの仕組みとロックされたステージの詳細については、「SUPERSEDED モードでの実行の処理方法」を参照してください。

ステージの有効なステータスは次のとおりで
す。InProgress、Stopping、Stopped、Succeeded、および Failed。失敗したステージは、

再試行不可能でない限り、再試行できます。詳細については、「ステージ実行」を参照してください。ステージは、指定した以前の成功した実行にロールバックできます。ステージは、「ステージロールバックの設定」で説明しているように、失敗時に自動的にロールバックするように設定できます。詳細については、「RollbackStage」を参照してください。

アクション実行

アクションの実行は、指定されたアーティファクトに対して動作する設定済みアクションを完了するプロセスです。これらは、入力アーティファクト、出力アーティファクト、またはその両方です。たとえば、ビルドアクションでは、アプリケーションのソースコードのコンパイルなど、入力アーティファクトに対してビルドコマンドを実行できます。アクション実行の詳細には、アクション実行ID、関連するパイプライン実行ソーストリガー、アクションの入出力アーティファクトが含まれます。

アクションの有効なステータスは、 `InProgress` 、 `Abandoned` 、 `Succeeded`、または `Failed` です。詳細については、「アクション実行」を参照してください。

実行タイプ

パイプラインまたはステージの実行は、標準実行またはロールバック実行のいずれかになります。

標準タイプの場合、実行には一意の ID があり、完全なパイプライン実行になります。パイプラインのロールバックには、ロールバックされるステージと、ロールバックする先のターゲット実行としての以前の成功したステージ実行があります。ターゲットのパイプライン実行は、ステージを再実行するためのソースリビジョンと変数を取得するために使用します。

アクションタイプ

アクションタイプ とは、 CodePipeline で選択できる事前設定済みのアクションです。アクションタイプは、その所有者、プロバイダー、バージョン、およびカテゴリによって定義されます。アクションタイプには、パイプライン内のアクションタスクを完了するために使用されるカスタマイズされたパラメータが用意されています。

アクションタイプに基づいてパイプラインに統合できる AWS のサービス およびサードパーティー製品およびサービスの詳細については、「」を参照してくださいCodePipeline アクションタイプとの統合。

CodePipeline のアクションタイプでサポートされている統合モデルの詳細については、統合モデルのリファレンス を参照してください。

サードパーティープロバイダーが CodePipeline でアクションタイプを設定および管理する方法については、 アクションタイプの使用 を参照してください。

アーティファクト

アーティファクトとは、パイプラインアクションによって処理されるアプリケーションのソースコード、構築されたアプリケーション、依存関係、定義ファイル、テンプレートなどのデータの集合を指します。アーティファクトは、いくつかのアクションによって生成され、他のアクションによって消費されます。パイプラインでは、アーティファクトは、アクション (入力アーティファクト) によって処理されるファイルのセットまたは完了したアクションの更新された出力 (出力アーティファクト) です。

アクションは、パイプラインアーティファクトバケットを使用してさらに処理するために、出力を別のアクションに渡します。CodePipeline はアーティファクトストアにアーティファクトをコピーし、このアーティファクトはそこでアクションによりピックアップされます。アーティファクトの詳細については、「入力および出力アーティファクト」を参照してください。

ソースリビジョン

ソースコードを変更すると、新しいバージョンが作成されます。ソースリビジョンは、パイプライン実行をトリガーするソース変更のバージョンです。実行はソースリビジョンを処理します。GitHub および CodeCommit リポジトリの場合は、コミットメッセージです。S3 バケットまたはアクションの場合、これはオブジェクトバージョンです。

指定したソースリビジョン (コミットなど) でパイプライン実行を開始できます。実行は指定されたリビジョンを処理し、実行に使用されたはずのリビジョンをオーバーライドします。詳細については、「ソースリビジョンオーバーライドでパイプラインを開始する」を参照してください。

トリガ

トリガーは、パイプラインを開始するイベントです。パイプラインの手動開始などの一部のトリガーは、パイプライン内のすべてのソースアクションプロバイダーで使用できます。パイプラインのソースプロバイダーに依存するトリガーもあります。例えば、CloudWatch イベントは Amazon CloudWatch のイベントリソースで設定され、イベントルールでターゲットとしてパイプラインの ARN が追加されている必要があります。Amazon CloudWatch Events は、CodeCommit または S3 ソースアクションを使用したパイプラインの自動変更検出に推奨されるトリガーです。Webhook は、サードパーティーのリポジトリイベント用に設定されるトリガーの一種です。例えば、WebhookV2 は、Git タグを使用して GitHub.com、GitHub Enterprise Server、GitLab.com、GitLab self-managed、Bitbucket Cloud などのサードパーティのソースプロバ

イダーのパイプラインを開始できるようにするトリガータイプです。パイプライン設定では、プッシュリクエストやプルリクエストなどのトリガーのフィルターを指定できます。Git タグ、ブランチ、またはファイルパスでコードプッシュイベントをフィルタリングできます。イベント (オープン、更新、クローズ) 、ブランチ、またはファイルパスでプルリクエストイベントをフィルタリングできます。

トリガーについての詳細は、「CodePipeline でパイプラインを編集する」を参照してください。Git タグをパイプラインのトリガーとして使用する手順を説明するチュートリアルについては、「チュートリアル: Git タグを使用してパイプラインを開始する」を参照してください。

> **Important**
>
> 30 日以上非アクティブなパイプラインでは、パイプラインのポーリングが無効になります。詳細については、パイプイライン構造リファレンスの「pollingDisabledAt」を参照してください。パイプラインをポーリングからイベントベースの変更検出に移行する手順については、「変更検出方法」を参照してください。

[変数]

変数は、パイプライン内のアクションを動的に設定するために使用できる値です。変数はパイプラインレベルで宣言したり、パイプライン内のアクションによって出力したりできます。変数値はパイプラインの実行時に解決され、実行履歴で確認できます。パイプラインレベルで宣言した変数は、パイプライン設定でデフォルト値を定義するか、特定の実行に応じて上書きすることができます。アクションによって出力される変数の場合、値はアクションが正常に完了した後に使用できます。詳細については、「変数リファレンス」を参照してください。

条件

条件内には、評価される一連のルールがあります。条件内のすべてのルールが成功すると、条件は満たされます。満たされない場合は、指定した結果 (ステージを失敗させるなど) を適用するように条件を設定できます。条件はゲートとも呼ばれます。これは、実行がステージに入り、ステージを通過し、通過後にステージを終了するタイミングを指定できるためです。これは、ゲートを閉じて交通の流れを溜め、次にゲートを開いてエリアへの交通の流れを許可することと似ています。条件タイプの結果には、ステージを失敗させることやロールバックすることが含まれます。条件は、これらのアクションがパイプラインステージでいつ発生するかを指定するのに役立ちます。条件は、ランタイムに上書きできます。

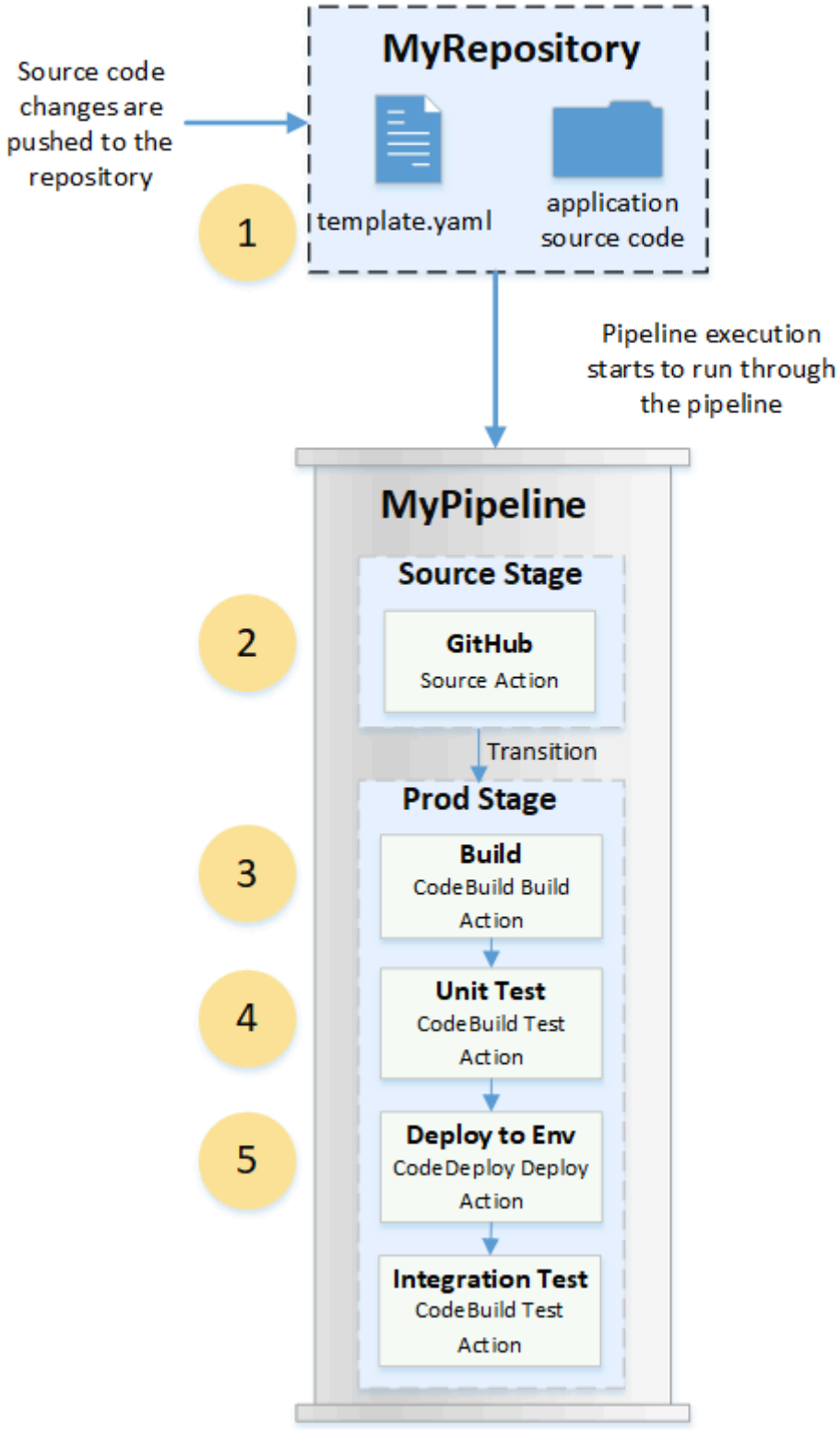
条件には 3 つのタイプがあります。入力条件は、「条件のルールが満たされた場合、ステージに入る」かどうかという質問に答えます。実行がステージに入ると、ステージはロックされ、次にルールが実行されます。失敗時の条件では、ステージが失敗すると、ルールが適用され、結果として失敗したステージがロールバックされます。成功時の条件では、ステージが成功すると、続行する前に正常な実行をチェックしてアラームを確認するなどのルールが適用されます。例えば、成功時の条件では、CloudWatchAlarm ルールでデプロイ環境にアラームがあることが検出されると、正常なステージがロールバックされます。詳細については、「ステージ条件はどのように機能しますか?」を参照してください。

Rules

条件では、1 つ以上の事前設定済みのルールを使用し、チェックを実行して条件が満たされなかった場合に設定済みの結果を適用します。例えば、アラームステータスとデプロイウィンドウの時間をチェックする入力条件のすべてのルールが満たされると、すべてのチェックに合格した正常なステージがデプロイされます。詳細については、「ステージ条件はどのように機能しますか?」を参照してください。

DevOps パイプラインの例

DevOps パイプラインの例として、2 つのステージから成るパイプラインに Source という名前のソースステージと Prod という第 2 ステージがあるとします。この例では、パイプラインは最新の変更でアプリケーションを更新し、最新の結果を継続的にデプロイしています。パイプラインは、最新のアプリケーションをデプロイする前に、ウェブアプリケーションを構築およびテストします。この例では、開発者グループが MyRepository という GitHub リポジトリでウェブアプリケーションのインフラストラクチャテンプレートとソースコードを設定しています。



例えば、開発者がウェブアプリケーションのインデックスページに修正をプッシュすると、次のようになります。

1. アプリケーションのソースコードは、パイプラインの GitHub ソースアクションとして設定されたリポジトリに保持されます。デベロッパーがコミットをリポジトリにプッシュすると、CodePipeline はプッシュされた変更を検出し、パイプラインの実行が ソースステージ から開始されます。
2. GitHub ソースアクションが正常に完了します (つまり、最新の変更がダウンロードされ、その実行に固有のアーティファクトバケットに保存されます)。GitHub ソースアクションによって生成される 出力アーティファクト (リポジトリからのアプリケーションファイル) は、次のステージのアクションによって処理される入力アーティファクトとして使用されます。
3. パイプラインの実行は、ソースステージから本番ステージに移行します。Prod Stage の最初のアクションは、CodeBuild で作成され、パイプラインで構築アクションとして設定された構築プロジェクトを実行します。ビルドタスクは、ビルド環境イメージをプルし、仮想コンテナにウェブアプリケーションをビルドします。
4. Prod Stage の次のアクションは、CodeBuild で作成され、パイプラインのテストアクションとして設定された単体テストプロジェクトです。
5. ユニットテストが行われたコードは、次に本番環境にアプリケーションをデプロイする本番ステージのデプロイアクションによって処理されます。デプロイアクションが正常に完了した後、ステージの最後のアクションは、CodeBuild で作成され、パイプラインのテストアクションとして設定された統合テストプロジェクトです。テストアクションは、ウェブアプリケーション上でリンクチェッカーなどのテストツールをインストールして実行するシェルスクリプトを呼び出します。正常に完了すると、ビルドされたウェブアプリケーションと一連のテスト結果が出力として得られます。

開発者はパイプラインにアクションを追加して、変更ごとにアプリケーションをビルドおよびテストした後で、アプリケーションをデプロイまたはさらにテストできます。

詳細については、「パイプライン実行の仕組み」を参照してください。

パイプライン実行の仕組み

このセクションでは、CodePipeline が一連の変更を処理する方法の概要を説明します。CodePipeline は、パイプラインを手動で開始したときや、ソースコードを変更したときに開始する各パイプライン実行を追跡します。CodePipeline は、以下の実行モードを使用して、各実行がパイプラインを進行する方法を処理します。詳細については、「パイプライン実行モードを設定または変更する」を参照してください。

- SUPERSEDED モード: より新しい実行が、より古い実行を追い越すことができます。これがデフォルトです。
- QUEUED モード: 実行は、キューに登録した順に 1 つずつ処理されます。これにはパイプラインタイプ V2 が必要です。
- PARALLEL モード: PARALLEL モードでは、複数の実行が同時に、互いに独立して実行されます。実行は、他の実行が完了するまで待たずに開始または終了します。これにはパイプラインタイプ V2 が必要です。

> **Important**
>
> PARALLEL モードのパイプラインでは、ステージのロールバックは使用できません。同様に、ロールバック結果タイプの障害条件を PARALLEL モードのパイプラインに追加することはできません。

パイプライン実行の開始方法

ソースコードを変更したり、パイプラインを手動で開始したりするときに、実行を開始できます。また、スケジュールした Amazon CloudWatch Events ルールを使用して実行をトリガーすることもできます。例えば、パイプラインのソースアクションとして設定されているリポジトリにソースコードの変更がプッシュされると、パイプラインはその変更を検出して実行を開始します。

> **Note**
>
> パイプラインに複数のソースアクションが含まれている場合、1 つのソースアクションに対してのみ変更が検出された場合でも、すべてのアクションが再度実行されます。

パイプライン実行でソースリビジョンを処理する方法

ソースコードの変更 (ソースリビジョン) で開始するパイプライン実行ごとに、ソースリビジョンは次のように決定されます。

- パイプラインに CodeCommit ソースがある場合、コミットをプッシュした時点で、CodePipeline は HEAD をクローンします。例えば、コミットをプッシュすると、実行 1 のパイプラインが開始します。2 番目のコミットをプッシュした時点で、実行 2 のパイプラインが開始します。

> **Note**
>
> PARALLEL モードのパイプラインに CodeCommit ソースがある場合、パイプライン実行をトリガーしたコミットに関係なく、ソースアクションは常に開始時に HEAD をクローンします。詳細については、「PARALLEL モードの CodeCommit または S3 ソースリビジョンは、EventBridge イベントと一致しない可能性があります」を参照してください。

- パイプラインに S3 ソースがある場合は、S3 バケット更新の EventBridge イベントが使用されます。例えば、ソースバケットでファイルを更新するとイベントが生成され、実行 1 のパイプラインが開始します。2 番目のバケット更新のイベントが発生した時点で、実行 2 のパイプラインが開始します。

> **Note**
>
> PARALLEL モードのパイプラインに S3 ソースがある場合、実行をトリガーしたイメージタグに関係なく、ソースアクションは常に最新のイメージタグで開始します。詳細については、「PARALLEL モードの CodeCommit または S3 ソースリビジョンは、EventBridge イベントと一致しない可能性があります」を参照してください。

- パイプラインに接続ソース (Bitbucket 接続など) がある場合、コミットをプッシュした時点で、CodePipeline は HEAD をクローンします。例えば、PARALLEL モードのパイプラインの場合、コミットをプッシュすると、実行 1 のパイプラインが開始し、2 番目のパイプライン実行で 2 番目のコミットが使用されます。

ソースオーバーライドと EventBridge 入力トランスフォーマーの連携方法

オーバーライドを使用して、パイプライン実行用に指定した特定のソースリビジョン ID でパイプラインを開始できます。例えば、CodeCommit ソースからの特定のコミット ID を処理するパイプラインを開始する場合、パイプラインの開始時にコミット ID をオーバーライドとして追加できます。

ソースリビジョンには `revisionType` として 4 つタイプがあります。

- `COMMIT_ID`
- `IMAGE_DIGEST`
- `S3_OBJECT_VERSION_ID`
- `S3_OBJECT_KEY`

> **Note**
>
> COMMIT_ID タイプと IMAGE_DIGEST タイプのソースリビジョンの場合、ソースリビジョン ID は、すべてのブランチをまたいでリポジトリ内のすべてのコンテンツに適用されます。

> **Note**
>
> ソースリビジョンの S3_OBJECT_VERSION_ID タイプと S3_OBJECT_KEY タイプは、特定の ObjectKey と VersionID を持つソースを上書きするために、個別に使用することも、一緒に使用することもできます。S3_OBJECT_KEY の場合、設定パラメータ AllowOverrideForS3ObjectKey を true に設定する必要があります。S3 のソース設定パラメータの詳細については、「設定パラメータ」を参照してください。

EventBridge の入力トランスフォーマーを使用してソースオーバーライドを指定できます。入力トランスフォーマーを使用して、データを次のいずれかとして渡します。

- 入力トランスフォーマーを使用して、データを JSON パラメータとして渡すことができます。
- 入力トランスフォーマーを使用してパイプライン変数を渡すことができます。

データを JSON パラメータとして渡す例については「Amazon ECR ソースアクションと EventBridge リソース」を、S3 の場合は「EventBridge と を使用する Amazon S3 ソースアクションへの接続 AWS CloudTrail」を、CodeCommit の場合は「CodeCommit ソースアクションと EventBridge」を参照してください。

パイプライン実行の停止方法

コンソールを使用してパイプライン実行を停止するには、パイプラインの視覚化ページ、実行履歴ページ、または詳細履歴ページで [Stop execution (実行の停止)] を選択します。CLI を使用してパイプライン実行を停止するには、stop-pipeline-execution コマンドを使用します。詳細については、「CodePipeline でパイプライン実行を停止します。」を参照してください。

パイプラインの実行を停止する方法は 2 つあります。

- [Stop and wait (停止して待機)]: 進行中のアクションの実行はすべて完了でき、後続のアクションは開始されません。パイプラインの実行は、後続のステージに進みません。すでに Stopping 状態にある実行ではこのオプションは使用できません。

- [Stop and abandon (停止して中止)]: 進行中のアクションの実行はすべて中止され、完了しません。後続のアクションは開始されません。パイプラインの実行は、後続のステージに進みません。このオプションは、すでに `Stopping` 状態にある実行で使用できます。

> **Note**
>
> このオプションを使用すると、タスクが失敗する可能性またはタスクの順序が正しくなくなる可能性があります。

各オプションでは、次のように、パイプラインおよびアクション実行フェーズのシーケンスが異なります。

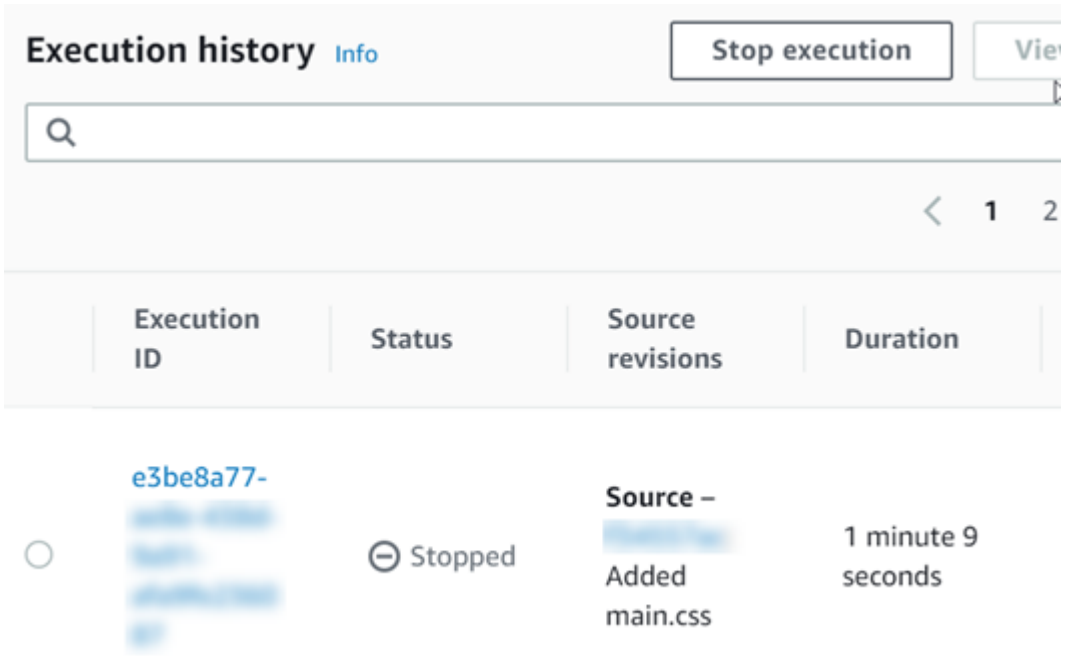
オプション 1: [Stop and wait (停止して待機)]

[Stop and wait (停止して待機)] を選択すると、実行中のアクションが完了するまで選択した実行が続行されます。例えば、次のパイプライン実行は、ビルドアクションの進行中に停止されました。

1. パイプラインビューには、成功メッセージのバナーが表示され、ビルドアクションが完了するまで続行されます。パイプラインの実行ステータスは [停止] です。

   履歴ビューでは、ビルドアクションなどの進行中のアクションの状態は、ビルドアクションが完了するまで [進行中] になります。アクションの進行中、パイプライン実行ステータスは [停止] になります。
2. 停止プロセスが完了すると、実行が停止します。ビルドアクションが正常に完了すると、そのステータスは [成功] になり、パイプライン実行のステータスは [停止]になります。後続のアクションは開始しません。[再試行] ボタンが有効になります。

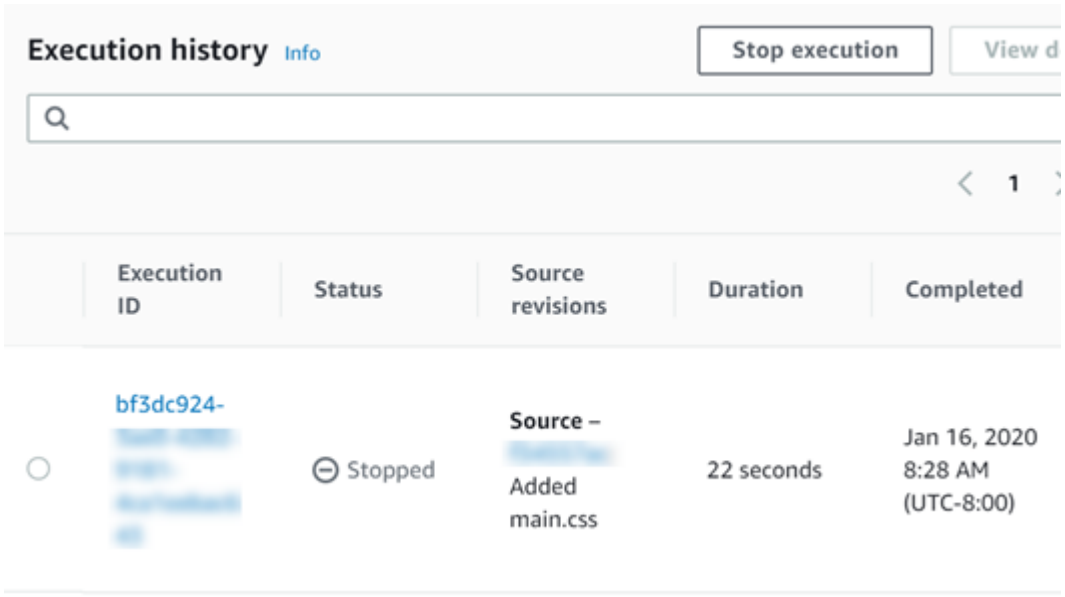
   履歴ビューでは、進行中のアクションが完了した後、実行ステータスは [停止] になります。



オプション 2: [Stop and abandon (停止して中止)]

[Stop and abandon (停止して中止)] を選択すると、選択した実行は進行中のアクションが完了するまで待機しません。アクションは中止されます。例えば、次のパイプライン実行は、ビルドアクションの進行中に停止され中止されました。

1. パイプラインビューでは、成功バナーメッセージが表示され、ビルドアクションには [進行中] ステータスが表示され、パイプライン実行には [停止] ステータスが表示されます。
2. パイプラインの実行が停止すると、ビルドアクションは [中止] の状態を示し、パイプライン実行の状態は [停止]と表示されます。後続のアクションは開始しません。[再試行] ボタンが有効になります。
3. 履歴ビューでは、実行ステータスは [停止] になります。



パイプライン実行を停止するユースケース

パイプラインの実行を停止するには、[stop the wait (待機して停止)] オプションを使用することをお勧めします。このオプションでは、パイプラインで、失敗したタスクやシーケンス外のタスクが回避されるため、より安全です。アクションが CodePipeline で中止されると、アクションプロバイダーはそのアクションに関連するすべてのタスクを続行します。 CloudFormation アクションの場合、パイプラインのデプロイアクションは中止されますが、スタックの更新が続行され、更新が失敗する可能性があります。

シーケンス外のタスクが発生する可能性のある中止されたアクションの例として、S3 デプロイアクションを使用してラージファイル (1 GB) をデプロイし、デプロイがすでに進行中にアクションを 停止して中止することを選択した場合、アクションは CodePipeline で中止されますが、Amazon S3 では続行されます。Amazon S3 は、アップロードをキャンセルするための指示を受け取りません。次に、非常に小さなファイルを使用して新しいパイプライン実行を開始すると、2 つのデプロイが進行中になります。新しい実行のファイルサイズが小さいので、古いデプロイがまだアップロードされている間に新しいデプロイが完了します。古いデプロイが完了すると、新しいファイルは古いファイルで上書きされます。

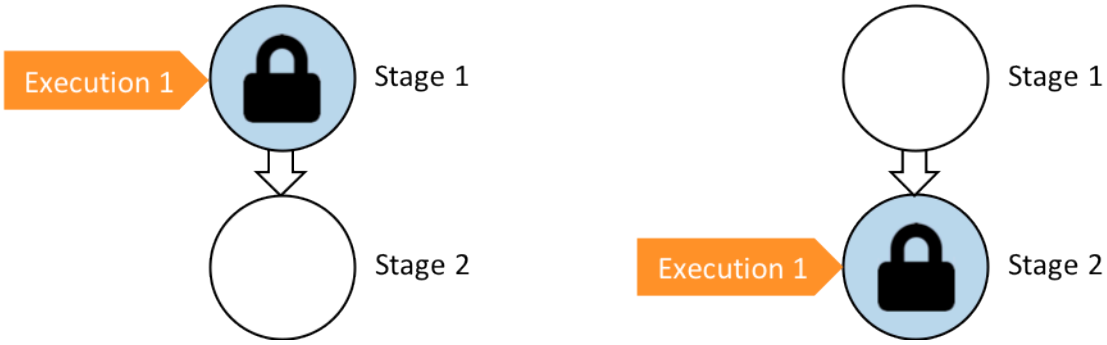
カスタムアクションがある場合は、[Stop and abandon (停止して中止)] オプションを使用できます。例えば、バグ修正のための新しい実行を開始する前に、完了する必要がない作業を含むカスタムアクションを中止できます。

SUPERSEDED モードでの実行の処理方法

実行を処理するデフォルトモードは SUPERSEDED モードです。実行は、実行によって取得され、処理される一連の変更で構成されます。パイプラインは、同時に複数の実行を処理できます。各実行は、パイプラインを介して個別に実行されます。パイプラインは各実行を順番に処理し、以前の実行を後の実行に置き換える場合があります。SUPERSEDED モードのパイプラインでは、以下のルールを使用して実行を処理します。

ルール 1: 実行の処理中はステージがロックされる

各ステージは一度に 1 つの実行しか処理できないため、進行中のステージはロックされます。実行が 1 つのステージを完了すると、パイプラインの次のステージに移行します。

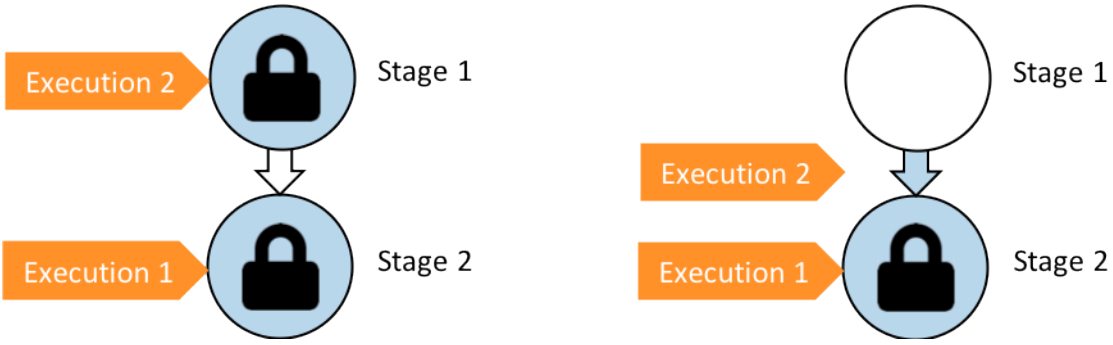


変更前: Stage 1 is locked as Execution 1 enters. 後 : Stage 2 is locked as Execution 1 enters.

ルール 2: その後の実行はステージのロックが解除されるまで待機する

ステージがロックされている間、待機中の実行はロックされたステージの前で保留されます。ステージに設定されたすべてのアクションは、ステージが完了したとみなされる前に正常に完了する必要があります。失敗すると、ステージのロックが解除されます。実行が停止すると、実行はステージ内で続行されず、ステージのロックが解除されます。

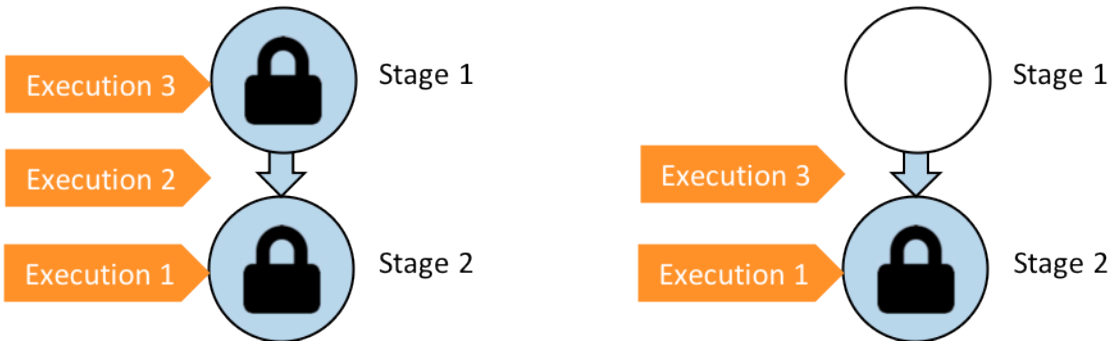
> **Note**
>
> 実行を停止する前に、ステージの前でトランジションを無効にすることをお勧めします。このようにすれば、実行が停止したためにステージのロックが解除されたときに、ステージは後続のパイプライン実行を受け付けません。



変更前: Stage 2 is locked as Execution 1 enters. 後 : Execution 2 exits Stage 1 and waits between stages.

ルール 3: 待機中の実行は、より新しい実行に置き換えられる

実行は、ステージ間でのみ置き換えられます。ロックされたステージは、ステージの完了を待つステージの前で 1 つの実行を保留します。より新しい実行は、待機中の実行を追い越し、ステージのロックが解除されるとすぐに次のステージに進みます。置き換えられた実行は続行されません。この例では、実行 2 は、ロックされたステージを待機している間に実行 3 に置き換えられています。実行 3 が次のステージに入ります。



前: 実行 2 は、実行 3 がステージ 1 に入っている間、ステージ間で待機します。後: 実行 3 はステージ 1 を終了します。実行 2 は実行 3 に置き換えられます。

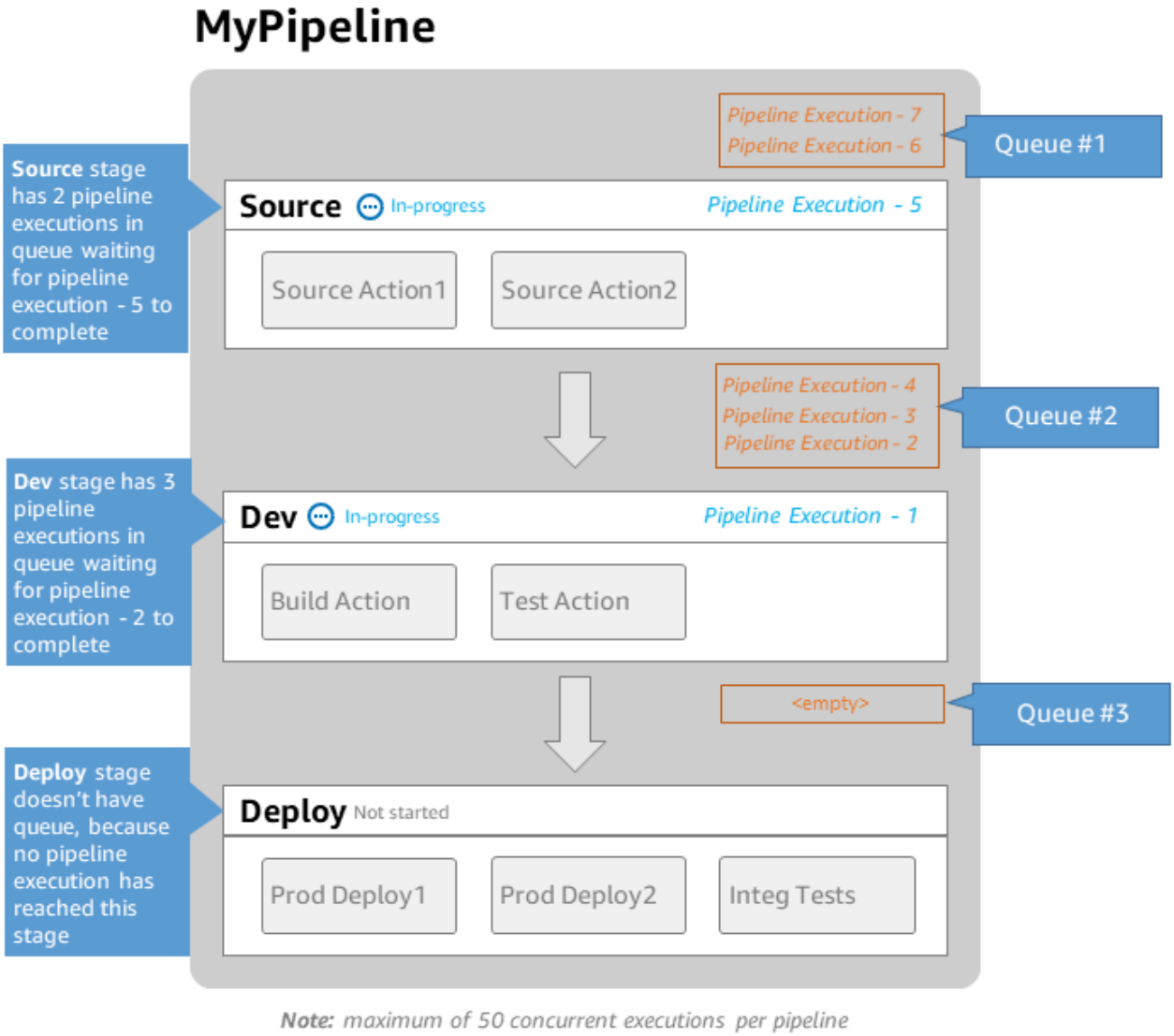
実行モードの表示と切り替えに関する考慮事項の詳細については、「パイプライン実行モードを設定または変更する」を参照してください。実行モードのクォータの詳細については、「AWS CodePipeline のクォータ」を参照してください。

QUEUED モードでの実行の処理方法

QUEUED モードのパイプラインの場合、実行の処理中はステージがロックされますが、待機中の実行は、既に開始した実行に取って代わることはありません。

待機中の実行は、ステージに到達した順にロックされたステージへのエントリポイントに集まり、待機中の実行のキューを形成します。QUEUED モードでは、同じパイプラインに複数のキューを含めることができます。キュー内の 1 つの実行がステージに入ると、ステージはロックされ、他の実行は入ることができません。この動作は SUPERSEDED モードと同じです。実行がステージを終了すると、ステージはロック解除され、次の実行を受け入れる準備が整います。

次の図は、QUEUED モードのパイプラインのステージがどのように実行を処理するかを示しています。例えば、ソースステージが実行 5 を処理する間、実行 6 および 7 はキュー #1 を形成し、ステージのエントリポイントで待機します。キュー内の次の実行は、ステージのロック解除後に処理されます。



実行モードの表示と切り替えに関する考慮事項の詳細については、「パイプライン実行モードを設定または変更する」を参照してください。実行モードのクォータの詳細については、「AWS CodePipeline のクォータ」を参照してください。

PARALLEL モードでの実行の処理方法

PARALLEL モードのパイプラインの場合、実行は互いに独立しており、開始する前に他の実行が完了するまで待機しません。キューは形成しません。パイプラインの並列実行を表示するには、実行履歴ビューを使用します。

機能ごとに独自の機能ブランチがあり、機能のデプロイ先のターゲットを他のユーザーと共有しない開発環境では、PARALLEL モードを使用します。

実行モードの表示と切り替えに関する考慮事項の詳細については、「パイプライン実行モードを設定または変更する」を参照してください。実行モードのクォータの詳細については、「AWS CodePipeline のクォータ」を参照してください。

パイプラインのフローを管理する

パイプラインの実行のフローは、次の方法で制御できます。

- トランジション。ステージへの実行の流れを制御します。トランジションは有効または無効にできます。トランジションが無効になっている場合、パイプラインの実行はステージに入ることができません。トランジションが無効になっているステージに入るのを待っているパイプライン実行は、インバウンド実行と呼ばれます。トランジションを有効にすると、インバウンド実行がステージに移動してロックされます。

  ロックされたステージを待機している実行と同様に、トランジションが無効になっている場合でも、ステージに入るのを待機している実行は新しい実行に置き換えることができます。無効なトランジションを再び有効にすると、トランジションが無効になっている間に古い実行に取って代わるものも含め、最新の実行がステージに入ります。
- 承認アクション。アクセス許可が付与されるまで (例えば、承認された ID からの手動承認を通じて)、パイプラインが次のアクションに移行するのを防ぎます。例えば、パイプラインが最終本番環境ステージに移行する時間を制御する場合は、承認アクションを使用できます。

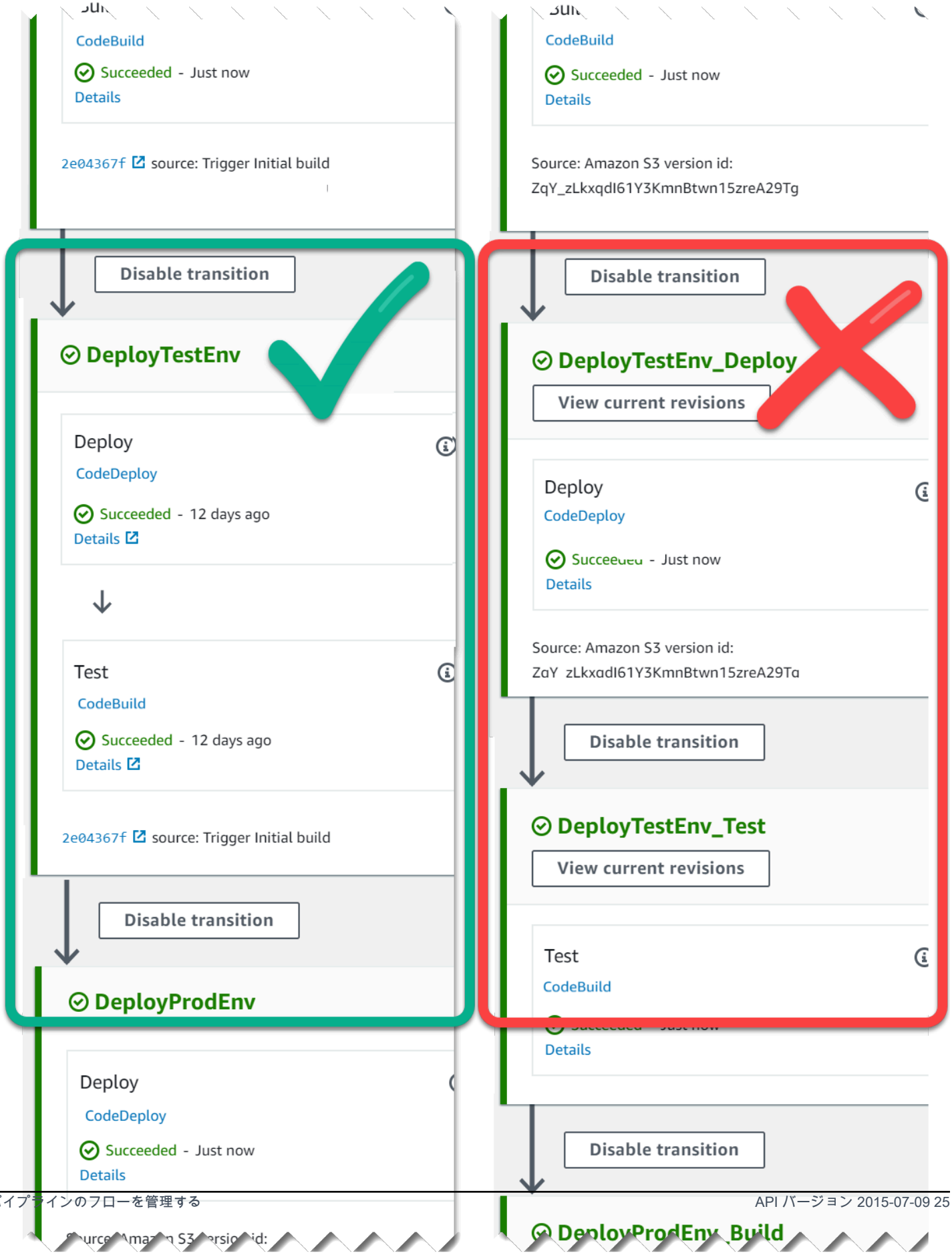
> **Note**
>
> 承認アクションのあるステージは、承認アクションが承認または却下されるか、タイムアウトするまでロックされます。タイムアウトした承認アクションは、失敗したアクションと同じ方法で処理されます。

- 失敗。ステージ内のアクションが正常に完了しなかった場合。リビジョンはステージの次のアクションまたはパイプラインの次のステージに移行しません。次の状況が発生する可能性があります。
  - 失敗したアクションを含むステージを手動で再試行します。これにより、実行が再開されます (失敗したアクションが再試行され、成功した場合は、ステージ/パイプラインで続行されます)。
  - 別の実行が失敗したステージに入り、失敗した実行に取って代わります。この時点で、失敗した実行を再試行することはできません。

推奨されるパイプライン構造

コード変更がパイプラインを通過する方法を決定するときは、ステージ内で関連するアクションをグループ化して、ステージがロックされたときにすべてのアクションが同じ実行を処理するようにすることをお勧めします。アプリケーション環境 AWS リージョンやアベイラビリティーゾーンごとにステージを作成できます。ステージが多すぎる (きめ細かすぎる) パイプラインでは、同時変更が多くなりすぎる可能性があります。一方、大きなステージでアクションが多い (粗すぎる) パイプラインでは、変更のリリースに時間がかかりすぎることがあります。

例えば、同じステージのデプロイアクション後のテストアクションは、デプロイされたのと同じ変更をテストすることが保証されています。この例では、変更がテスト環境にデプロイされた後でテストされた後で、テスト環境からの最新の変更が本番環境にデプロイされます。推奨される例では、テスト環境と本番環境は別々のステージです。


CodeBuild
Succeeded - Just now
Details
2e04367f source: Trigger Initial build
Disable transition
DeployTestEnv
Deploy
CodeDeploy
Succeeded - 12 days ago
Details
Test
CodeBuild
Succeeded - 12 days ago
Details
2e04367f source: Trigger Initial build
Disable transition
DeployProdEnv
Deploy
CodeDeploy
Succeeded - Just now
Details
CodeBuild
Succeeded - Just now
Details
Source: Amazon S3 version id:
ZqY_zLkxqdI61Y3KmnBtwn15zreA29Tg
Disable transition
DeployTestEnv_Deploy
View current revisions
Deploy
CodeDeploy
Succeeded - Just now
Details
Source: Amazon S3 version id:
ZqY_zLkxqdI61Y3KmnBtwn15zreA29Tg
Disable transition
DeployTestEnv_Test
View current revisions
Test
CodeBuild
Details
Disable transition
DeployProdEnv_Build

左: 関連するテスト、デプロイ、および承認アクションをグループ化 (推奨)。右: 別のステージでの関連アクション (非推奨)。

インバウンド実行の仕組み

インバウンド実行は、使用できないステージ、トランジション、またはアクションが使用可能になるのを待ってから先に進む実行です。次のステージ、トランジション、またはアクションは、次の理由で使用できない可能性があります。

- 別の実行はすでに次のステージに入り、ロックされています。
- 次のステージに入るためのトランジションは無効になります。

現在の実行に後続のステージで完了する時間があるかどうかを制御する場合、または特定の時点ですべてのアクションを停止する場合は、インバウンド実行を保持するトランジションを無効にすることができます。インバウンド実行があるかどうかを判断するには、コンソールでパイプラインを表示するか、get-pipeline-state コマンドからの出力を表示します。

インバウンド実行は、以下の考慮事項に注意して動作します。

- アクション、トランジション、またはロックされたステージが使用可能になると、進行中のインバウンド実行がステージに入り、パイプラインを継続します。
- インバウンド実行が待機している間は、手動で停止できます。インバウンド実行には、`InProgress`、`Stopped`、または `Failed` の状態があります。
- インバウンド実行が停止または失敗した場合、再試行する失敗したアクションがないため、再試行できません。インバウンド実行が停止し、トランジションが有効な場合、停止したインバウンド実行はステージに継続されません。

インバウンド実行を表示または停止できます。

入力および出力アーティファクト

CodePipeline は、デベロッパーツールと統合され、コードの変更をチェックし、継続的デリバリープロセスのすべてのステージを経て構築およびデプロイします。アーティファクトは、パイプライン内のアクションによって処理されるファイルであり、アプリケーションコードが含まれるファイルやフォルダ、インデックスページファイル、スクリプトなどが該当します。例えば、Amazon S3 ソースアクションアーティファクトは、アプリケーションソースコードファイルがパイプラインソースア

クションに提供されるファイル名 (またはファイルパス) です。ファイルは、アーティファクト名の例 SampleApp_Windows.zip のように ZIP ファイルとして提供されます。ソースアクションの出力アーティファクトであるアプリケーションソースコードファイルは、アクションからの出力アーティファクトであり、ビルドアクションなどの次のアクションの入力アーティファクトでもあります。別の例として、ビルドアクションは、入力アーティファクトのアプリケーションソースコード (アプリケーションソースコードファイル) をコンパイルするビルドコマンドを実行することがあります。CodeBuild アクションの AWS CodeBuild ビルドおよびテストアクションリファレンス など、アーティファクトパラメータの詳細については、特定のアクションのアクション設定リファレンスページを参照してください。

アクションは、パイプラインの作成時に選択した Amazon S3 アーティファクトバケットに保存されている入力アーティファクトと出力アーティファクトを使用します。CodePipeline は、ステージのアクションタイプに応じて、入力または出力アーティファクトのファイルを zip して転送します。

> **Note**
>
> アーティファクトバケットは、ソースアクションとして S3 が選択されているパイプラインのソースファイルの場所として使用されるバケットとは異なります。

例えば、次のようになります。

1. CodePipeline は、ソースリポジトリへのコミットがあるときにパイプラインをトリガーして実行し、ソース ステージからの出力アーティファクト (構築されるすべてのファイル) を提供します。
2. 前のステップの出力アーティファクト (ビルドする任意のファイル) は、ビルドステージに入力アーティファクトとして取り込まれます。ビルドステージからの出力アーティファクト (ビルドされたアプリケーション) は、更新されたアプリケーションまたは更新された Docker イメージ (コンテナへのビルド済み) である場合があります。
3. 前のステップの出力アーティファクト (ビルドされたアプリケーション) は、デプロイステージ (AWS クラウドのステージング環境や本稼働環境など) に入力アーティファクトとして取り込まれます。アプリケーションをデプロイのフリートにデプロイすることも、ECS クラスターで実行するタスクにコンテナベースのアプリケーションをデプロイすることもできます。

アクションを作成または編集するときは、アクションの入力および出力アーティファクトを指定します。例えば、ソースステージとデプロイステージを含む 2 ステージのパイプラインに対し、[アクションの編集] で、デプロイアクションの入力アーティファクトに対するソースアクションのアーティファクト名を選択します。

- コンソールを使用して最初のパイプラインを作成する AWS アカウント と、CodePipeline は同じ に Amazon S3 バケットを作成し AWS リージョン 、すべてのパイプラインの項目を保存します。コンソールを使用して、そのリージョンに別のパイプラインを作成するたびに、CodePipeline はバケット内にそのパイプライン用のフォルダを作成します。このフォルダを使用して、自動リリースプロセスの実行時にパイプラインのアイテムを格納します。このバケットの名前は codepipeline-*region-12345EXAMPLE-1a2b-3c4d-67890EXAMPLE* です。ここで、*region* はパイプラインを作成した AWS リージョンであり、*12345EXAMPLE-1a2b-3c4d-67890EXAMPLE* はバケット名が一意であることを確認する汎用一意識別子 (UUID) に基づくランダムな値です。

> **Note**
>
> パイプラインを作成しているリージョンに codepipeline-region- で始まるバケットがすでにある場合、CodePipeline はそれをデフォルトのバケットとして使用します。また、辞書式順序に従います。例えば、codepipeline-region-abcexample は、codepipeline-region-defexample の前に選択されます。

CodePipeline はアーティファクト名を切り捨てます。これにより、一部のバケット名が類似しているように見える可能性があります。アーティファクト名が切り詰められたように見えても、CodePipeline は、名前が切り詰められたアーティファクトに影響されない方法でアーティファクトバケットにマッピングします。パイプラインは正常に動作します。これは、フォルダやアーティファクトでは問題となりません。パイプライン名には 100 文字の制限があります。アーティファクトフォルダ名は、短縮されたように見えても、パイプラインに対して依然として一意です。

パイプラインを作成または編集するときは、パイプライン AWS アカウント と にアーティファクトバケットが必要です。また AWS リージョン、アクションを実行する予定のリージョンごとに 1 つのアーティファクトバケットが必要です。コンソールを使用してパイプラインまたはクロスリージョンアクションを作成する場合、デフォルトのアーティファクトバケットは、アクションがあるリージョンの CodePipeline によって設定されます。

を使用してパイプライン AWS CLI を作成する場合、そのバケットがパイプラインと同じ AWS アカウント および にある限り、そのパイプラインのアーティファクト AWS リージョン を任意の Amazon S3 バケットに保存できます。アカウントに許可されている Amazon S3 バケットの制限を超えることが懸念される場合は、これを行うことができます。を使用してパイプライン AWS CLI を作成または編集し、クロスリージョンアクション (パイプラインとは異なるリージョンの

AWS プロバイダーとのアクション) を追加する場合は、アクションを実行する予定の追加のリージョンごとにアーティファクトバケットを指定する必要があります。

- すべてのアクションには種類があります。種類に応じて、アクションは次のいずれかまたは両方を持つ場合があります。
  - アクションが実行されている間に消費または動作するアーティファクトである入力アーティファクト。
  - アクションの出力である出力アーティファクト。

  パイプラインの各出力アーティファクトには一意の名前が必要です。アクションのすべての入力アーティファクトは、そのアクションがステージのアクションの直前であるか、あるいはいくつか前のステージで実行されているかどうかにかかわらず、パイプラインの以前のアクションの出力アーティファクトと一致していなければなりません。

  アーティファクトは、複数のアクションによって処理することができます。

ステージ条件はどのように機能しますか?

ルールを指定する条件ごとに、ルールが実行されます。条件が失敗した場合、ステージは指定された結果を実行します。ルールの一部として、CodePipeline でケース別に使用するリソースも必要に応じて指定します。例えば、`CloudWatchAlarm` ルールは CloudWatch アラームリソースを使用して条件のチェックを実行します。

条件は複数のルールと一致する場合があり、各ルールは 3 つのプロバイダーのいずれかを指定できます。

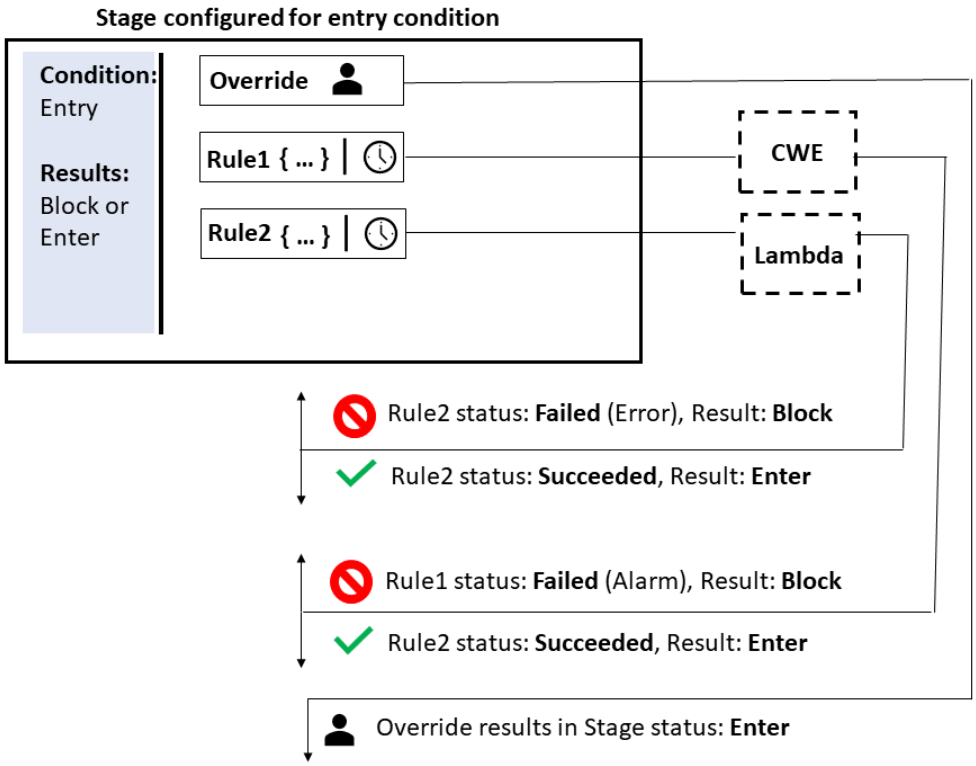
条件を作成するための大まかなフローは次のとおりです。

1. CodePipeline で利用可能な条件タイプから条件のタイプを選択します。例えば、ステージの成功後、続行前に一連のルールを使用してチェックを実行できるようにステージを設定するには、成功時の条件タイプを使用します。
2. ルールを選択します。例えば、`CloudWatchAlarm` ルールはアラームをチェックし、EB を使用して事前設定されたアラームのしきい値を確認します。チェックが成功し、アラームがしきい値を下回っていると、ステージを続行できます。
3. ルールが失敗した場合に使用するロールバックなど、結果を設定します。

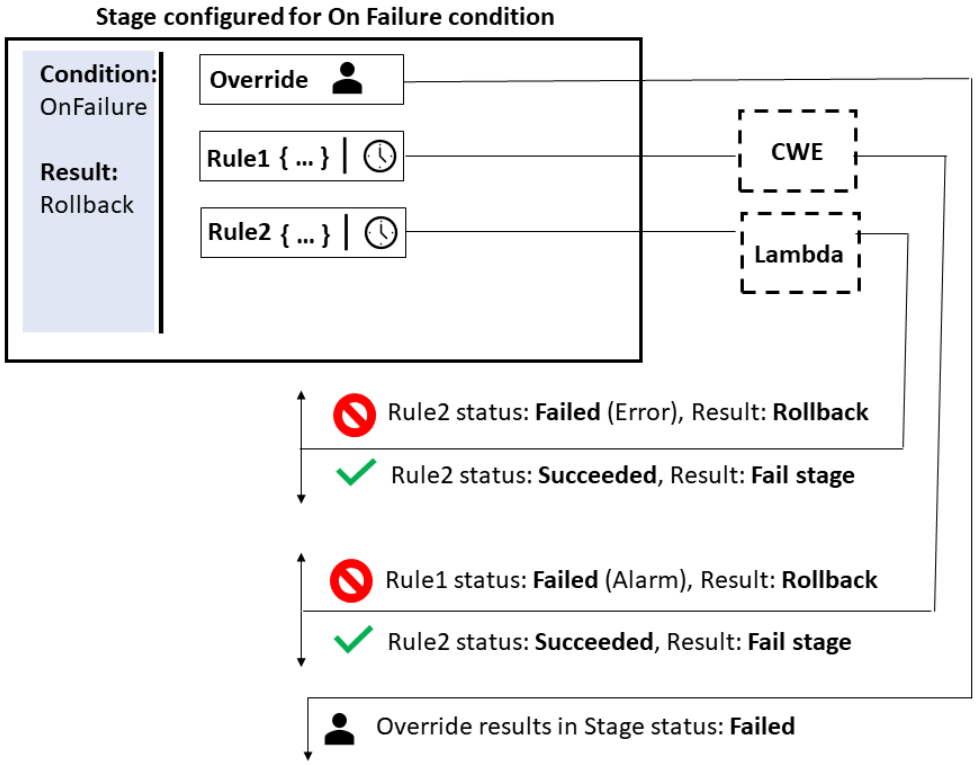
条件は、特定の式タイプで使用します。条件ごとに利用可能な特定の結果として、次のようなオプションがあります。

- 入力 - チェックするための条件。条件を満たすと、ステージへの入力が許可されます。ルールに適用される結果のオプションは、失敗またはスキップです。
- 失敗時 - 失敗したときにステージをチェックするための条件。ルールに適用される結果のオプションは、ロールバックです。
- 成功時 - ステージが成功したときにステージをチェックするための条件。ルールに適用される結果のオプションは、ロールバックまたは失敗です。

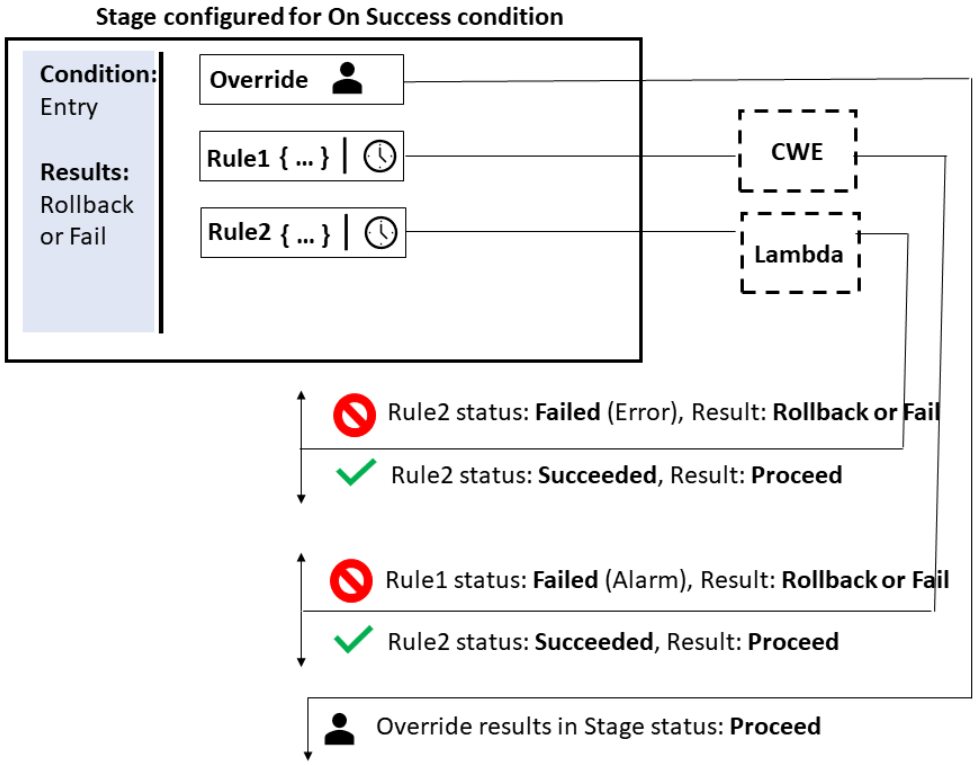
次の図は、CodePipeline の入力条件タイプのフロー例を示しています。条件は、条件が満たされない場合 (ルールが失敗した場合) に何が起こるかという質問に答えます。次のフローでは、入力条件に LambdaInvoke ルールと `CloudWatchAlarm` ルールを設定しています。ルールが失敗すると、設定済みの結果 (失敗など) が適用されます。



次の図は、CodePipeline の失敗時の条件タイプのフロー例を示しています。条件は、条件が満たされた場合 (すべてのルールがチェックに成功した場合) にどうなるかという質問に答えます。次のフローでは、失敗時の条件を LambdaInvoke ルールと `CloudWatchAlarm` ルールで設定しています。ルールが成功すると、設定済みの結果 (失敗など) が適用されます。



次の図は、CodePipeline の成功時の条件タイプのフロー例を示しています。条件は、条件が満たされた場合 (すべてのルールがチェックに成功した場合) にどうなるかという質問に答えます。次のフローでは、成功時の条件に `LambdaInvoke` ルールと `CloudWatchAlarm` ルールを設定しています。ルールが成功すると、設定済みの結果 (失敗など) が適用されます。



ステージ条件のルール

ステージ条件を設定するときは、事前定義されたルールから選択し、ルールの結果を指定します。条件内のルールのいずれかが失敗すると、条件の状態は [失敗しました] になり、すべてのルールが成功すると、[成功しました] になります。失敗時の条件と成功時の条件の基準を満たす方法は、ルールのタイプによって異なります。

ステージ条件に追加できるマネージドルールは以下のとおりです。

- 条件は、コマンドルールを使用して、条件のルール基準を満たすコマンドを指定できます。このルールの詳細については、「コマンド」を参照してください。
- AWS DeploymentWindow ルールを条件で使用すると、デプロイを許可する承認済みのデプロイ時間を指定できます。ルールの基準は、デプロイウィンドウに指定した cron 式で測定されます。デプロイウィンドウの日付と時刻がルールの cron 式の基準を満たすと、ルールは成功します。このルールの詳細については、「DeploymentWindow」を参照してください。
- AWS Lambda ルールを条件で使用すると、設定した Lambda 関数から返されるエラー状態をチェックできます。チェックが Lambda 関数の結果を受け取ると、ルールは満たされます。Lambda 関数からのエラーは、失敗時の条件の基準を満たします。このルールの詳細については、「LambdaInvoke」を参照してください。

- AWS CloudWatchAlarm ルールを条件で使用すると、CloudWatch イベントから設定したアラームをチェックできます。チェックが OK、ALARM、または INSUFFICIENT_DATA のアラーム状態を返すと、ルールが満たされます。成功時の条件の場合、OK と INSUFFICIENT_DATA が基準を満たします。失敗時の条件の場合、ALARM が基準を満たします。このルールの詳細については、「CloudwatchAlarm」を参照してください。
- VariableCheck ルールを条件で使用すると、出力変数を指定した式と照合する条件を作成できます。変数値がルール基準を満たすと (変数値が指定した出力変数以上であるなど)、ルールはチェックに合格します。このルールの詳細については、「VariableCheck」を参照してください。

パイプラインのタイプ

CodePipeline では、アプリケーションのニーズに合わせてパイプラインの機能とコストを調整できるように、特徴および価格が異なる以下のパイプラインのタイプを提供しています。

- V1 タイプのパイプラインは、標準のパイプライン、ステージ、アクションレベルのパラメータを含む JSON 構造になっています。
- V2 タイプのパイプラインは V1 タイプと同じ構造であり、リリースの安全性とトリガーの設定のための追加のパラメータがあります。

CodePipeline の料金については、料金を参照してください。

各パイプラインのタイプのパラメータの詳細については、CodePipeline パイプライン構造リファレンス のページを参照してください。どのタイプのパイプラインを選択するかについては、「適切なパイプラインのタイプの選択」を参照してください。

適切なパイプラインのタイプの選択

パイプラインのタイプは、各パイプラインバージョンでサポートされている一連の特徴と機能に基づいて決定します。

以下に、各タイプのパイプラインのユースケースと特徴をまとめました。

	V1 タイプ	V2 タイプ
特性		
ユースケース	• 標準デプロイ	• 実行時にパイプラインレベルの変数を渡すように設定したデプロイ • パイプラインが Git タグで開始されるように設定したデプロイ
アクションレベルの変数	サポート対象	サポート
PARALLEL 実行モード	サポート外	サポート
パイプラインレベルの変数	サポート外	サポート
QUEUED 実行モード	サポート外	サポート
パイプラインステージのロールバック	サポート外	サポート
ソースリビジョンの上書き	サポート外	サポート
ステージ条件	サポート外	サポート
Git タグ、プルリクエスト、ブランチ、またはファイルパスのトリガーとフィルタリング	サポート外	サポート
コマンドアクション	サポート外	サポート
スキップ結果を使用した入力条件の作成	サポート外	サポート
ステージ障害時の自動再試行を設定する	サポート外	サポート

CodePipeline の料金については、料金を参照してください。

Python スクリプトを作成して実行することで、V1 タイプのパイプラインを V2 タイプのパイプラインに移行する潜在的なコストを分析できます。

> **Note**
>
> 以下のサンプルスクリプトは、デモンストレーションと評価のみを目的としています。これは見積もりツールではなく、V2 タイプのパイプラインを実際に使用した場合のコストを保証するものではありません。また、適用される可能性のある税金も含まれていません。CodePipeline の料金については、料金を参照してください。

V1 タイプのパイプラインを V2 タイプのパイプラインに移行するコストを評価するのに役立つスクリプトを作成して実行するには

1. Python をダウンロードしてインストールします。
2. ターミナルウィンドウを開きます。次のコマンドを実行して、PipelineCostAnalyzer.py という名前の新しい Python スクリプトを作成します。

```
vi PipelineCostAnalyzer.py
```

3. 次のコードをコピーして、PipelineCostAnalyzer.py スクリプトに貼り付けます。

```
import boto3
import sys
import math
from datetime import datetime, timedelta, timezone

if len(sys.argv) < 3:
    raise Exception("Please provide region name and pipeline name as arguments.
 Example usage: python PipelineCostAnalyzer.py us-east-1 MyPipeline")
session = boto3.Session(profile_name='default', region_name=sys.argv[1])
pipeline = sys.argv[2]
codepipeline = session.client('codepipeline')

def analyze_cost_in_v2(pipeline_name):
    if codepipeline.get_pipeline(name=pipeline)['pipeline']['pipelineType'] ==
 'V2':
        raise Exception("Provided pipeline is already of type V2.")
    total_action_executions = 0
    total_blling_action_executions = 0
```

```
    total_action_execution_minutes = 0
    cost = 0.0
    hasNextToken = True
    nextToken = ""

    while hasNextToken:
        if nextToken=="":
            response =
 codepipeline.list_action_executions(pipelineName=pipeline_name)
        else:
            response =
 codepipeline.list_action_executions(pipelineName=pipeline_name,
 nextToken=nextToken)
        if 'nextToken' in response:
            nextToken = response['nextToken']
        else:
            hasNextToken= False
        for action_execution in response['actionExecutionDetails']:
            start_time = action_execution['startTime']
            end_time = action_execution['lastUpdateTime']
            if (start_time < (datetime.now(timezone.utc) - timedelta(days=30))):
                hasNextToken= False
                continue
            total_action_executions += 1
            if (action_execution['status'] in ['Succeeded', 'Failed', 'Stopped']):
                action_owner = action_execution['input']['actionTypeId']['owner']
                action_category = action_execution['input']['actionTypeId']
['category']
                if (action_owner == 'Custom' or (action_owner == 'AWS' and
 action_category == 'Approval')):
                    continue

                total_blling_action_executions += 1
                action_execution_minutes = (end_time -
 start_time).total_seconds()/60
                action_execution_cost = math.ceil(action_execution_minutes) * 0.002
                total_action_execution_minutes += action_execution_minutes
                cost = round(cost + action_execution_cost, 2)

    print ("{:<40}".format('Activity in last 30 days:'))
    print ("| {:<40} | {:<10}".format('_____________________________________',
 '____________________'))
    print ("| {:<40} | {:<10}".format('Total action executions:',
 total_action_executions))
```

```
    print ("| {:<40} | {:<10}".format('Total billing action executions:',
 total_blling_action_executions))
    print ("| {:<40} | {:<10}".format('Total billing action execution minutes:',
 round(total_action_execution_minutes, 2)))
    print ("| {:<40} | {:<10}".format('Cost of moving to V2 in $:', cost - 1))

analyze_cost_in_v2(pipeline)
```

4. ターミナルまたはコマンドプロンプトから、アナライザースクリプトを作成したディレクトリに変更します。

   そのディレクトリから次のコマンドを実行します。region は、分析する V1 パイプラインを作成した AWS リージョン です。必要に応じて、特定のパイプラインの名前を指定して評価することもできます。

   ```
   python3 PipelineCostAnalyzer.py region --pipelineName
   ```

   例えば、PipelineCostAnalyzer.py という名前の Python スクリプトを実行するには、次のコマンドを使用します。この例で、リージョンは us-west-2 です。

   ```
   python3 PipelineCostAnalyzer.py us-west-2
   ```

   > **Note**
   >
   > このスクリプトは、特定のパイプライン名を指定しない限り、指定した AWS リージョン 内のすべての V1 パイプラインを分析します。

5. 次のスクリプト出力例では、アクション実行のリスト、請求対象のアクション実行のリスト、アクション実行の合計ランタイム、アクションを V2 パイプラインで実行した場合の推定コストを確認できます。

   ```
   Activity in last 30 days:
    | ________________________________________      | ________________
    | Total action executions:                    | 9
    | Total billing action executions:            | 9
    | Total billing action execution minutes:     | 5.59
    | Cost of moving to V2 in $:                  | -0.76
   ```

この例で、最後の行の負の値は、V2 タイプのパイプラインに移行することで節約できる推定金額を表します。

> **Note**
>
> スクリプト出力および関連するコストやその他の情報を示す例は、あくまでも推定です。デモンストレーションと評価のみを目的としており、実際の節約を保証するものではありません。CodePipeline の料金については、料金を参照してください。

CodePipeline の使用開始

CodePipeline を初めて使用する場合は、この章の手順に従って設定した後、このガイドのチュートリアルに従ってください。

CodePipeline コンソールには、情報アイコン、またはそのページの Info リンクから開くことができる折りたたみ可能なパネル中に有益な情報を含んでいます。

()。

このパネルは、いつでも閉じることができます。

また、CodePipeline 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、「コンソールでのリソースの表示」を参照してください。

AWS CodePipeline を初めて使用する前に、 を作成し AWS アカウント 、最初の管理ユーザーを作成する必要があります。

トピック

- にサインアップする AWS アカウント
- CodePipeline への管理アクセスに マネージドポリシーを適用する
- のインストール AWS CLI
- CodePipeline のコンソールを開く
- 次の手順

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 にサインイン AWS し、選択した 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。

CodePipeline への管理アクセスに マネージドポリシーを適用する

CodePipeline とやり取りするためのアクセス許可を付与する必要があります。これを行う最も簡単な方法は、AWSCodePipeline_FullAccess マネージドポリシーを管理者ユーザーに適用することです。

> **Note**
>
> AWSCodePipeline_FullAccess ポリシーには、コンソールのユーザーが IAM ロールを CodePipeline など他の AWS のサービスに渡すためのアクセス許可が含まれます。これにより、サービスがロールの継承を行い、ユーザーの代わりにアクションを実行できるようになります。ポリシーをユーザー、ロール、またはグループにアタッチすると、iam:PassRole アクセス許可が適用されます。ポリシーが、信頼されたユーザーにのみ適用されていることを確認します。これらのアクセス許可が付与されたユーザーがコンソールを使用してパイプラインを作成または編集する場合は、次の方法を使用できます。
>
> - CodePipeline サービスロールを作成するか、既存のロールを選択してロールを CodePipeline に渡します。
> - 変更を検出するための CloudWatch Events ルールを作成し、CloudWatch Events サービスロールを CloudWatch Events に渡すことを選択する場合があります。
>
> 詳細については、「 にロールを渡すアクセス許可をユーザーに付与する AWS のサービス」を参照してください。

> **Note**
>
> AWSCodePipeline_FullAccess のポリシーでは、IAM ユーザーがアクセスしたすべての CodePipeline アクションおよびリソースだけでなく、パイプラインのステージ (CodeDeploy、Elastic Beanstalk、またはAmazon S3 など) pipeline 中のステージ作成時に可能なすべてのアクションに対するアクセス許可を付与します。ベストプラクティスとして、職務遂行に必要な許可のみを個人に付与することをお勧めします。IAM ユーザーを、限られたCodePipeline アクションおよびリソースのセットに制限する方法の詳細については、CodePipeline サービスロールからアクセス許可を削除する を参照してください。

アクセス許可を付与するにはユーザー、グループ、またはロールにアクセス許可を追加します。

- のユーザーとグループ AWS IAM アイデンティティセンター:

  アクセス許可セットを作成します。「AWS IAM アイデンティティセンター ユーザーガイド」の「アクセス許可セットを作成する」の手順に従ってください。

- IAM 内で、ID プロバイダーによって管理されているユーザー:

  ID フェデレーションのロールを作成します。詳細については IAM ユーザーガイド の サードパーティー ID プロバイダー (フェデレーション) 用のロールを作成する を参照してください。

- IAM ユーザー:
  - ユーザーが担当できるロールを作成します。手順については IAM ユーザーガイド の IAM ユーザーのロールの作成 を参照してください。
  - (お奨めできない方法) ポリシーをユーザーに直接アタッチするか、ユーザーをユーザーグループに追加します。「IAM ユーザーガイド」の「ユーザー (コンソール) へのアクセス許可の追加」の指示に従います。

のインストール AWS CLI

ローカル開発マシンで CLI AWS から CodePipeline コマンドを呼び出すには、 CLI AWS をインストールする必要があります。このステップは、このガイドの手順を CodePipeline コンソールでのみ使用して開始する場合は、省略可能です。

をインストールして設定するには AWS CLI

1. ローカルマシンで、 をダウンロードしてインストールします AWS CLI。これにより、コマンドラインから CodePipeline とやり取りすることができます。詳細については、AWS 「 コマンドラインインターフェイスのセットアップ」を参照してください。

   > **Note**
   >
   > CodePipeline は AWS CLI バージョン 1.7.38 以降でのみ動作します。インストール AWS CLI されている可能性がある のバージョンを確認するには、 コマンドを実行しますaws --version。の古いバージョン AWS CLI を最新バージョンにアップグレードするには、「 のアンインストール AWS CLI」の手順に従ってから、「 のインストール AWS Command Line Interface」の手順に従ってください。

2. 次のように、 configure コマンド AWS CLI を使用して を設定します。

```
aws configure
```

プロンプトが表示されたら、CodePipeline で使用する IAM ユーザーの AWS アクセスキーと AWS シークレットアクセスキーを指定します。デフォルトのリージョン名の入力を求められたら、 など、パイプラインを作成するリージョンを指定しますus-east-2。デフォルトの出力形式の入力を求められたら、json を指定します。例えば:

```
AWS Access Key ID [None]: Type your target AWS access key ID here, and then press
 Enter
AWS Secret Access Key [None]: Type your target AWS secret access key here, and then
 press Enter
Default region name [None]: Type us-east-2 here, and then press Enter
Default output format [None]: Type json here, and then press Enter
```

> **Note**
>
> IAM、アクセスキー、シークレットキーに関するさらなる詳細については、Managing Access Keys for IAM Users および How Do I Get Credentials? を参照してください。CodePipeline のために利用できるリージョンとエンドポイントに関するさらなる情報については、AWS CodePipeline endpoints and quotas を参照してください。

CodePipeline のコンソールを開く

- にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

次の手順

前提条件を完了しました。CodePipeline の使用を開始できます。 CodePipeline の使用を開始するには、CodePipeline チュートリアル を参照します。

CodePipeline との製品とサービスの統合

デフォルトでは、 AWS CodePipeline は多数の AWS のサービス およびパートナー製品およびサービスと統合されています。以下のセクションの情報は、使用している製品やサービスと統合するための CodePipeline の設定に役立ちます。

このサービスを利用する際に役立つ関連リソースは次のとおりです。

トピック

- CodePipeline アクションタイプとの統合
- CodePipeline との一般的統合
- コミュニティからの例

CodePipeline アクションタイプとの統合

このトピックの統合情報は、CodePipeline アクションの種類によって編成されます。

トピック

- ソースアクションの統合
- ビルドアクションの統合
- テストアクションの統合
- デプロイアクションの統合
- Amazon Simple Notification Service との承認アクションの統合
- 呼び出しアクションの統合

ソースアクションの統合

以下の情報は CodePipeline アクションの種類別に整理されていますので、CodePipeline を設定して以下のソースアクションプロバイダーとの統合に役立ちます。

トピック

- Amazon ECR ソースアクション
- Amazon S3 ソースアクション

- Bitbucket Cloud、GitHub (GitHub アプリ経由)、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドへの接続
- CodeCommit ソースアクション
- GitHub (OAuth アプリ経由) ソースアクション

Amazon ECR ソースアクション

Amazon ECR は Docker AWS イメージリポジトリサービスです。Docker イメージをリポジトリにアップロードするには、Docker のプッシュコマンドおよびプルコマンドを使用します。Amazon ECR リポジトリの URI とイメージは、ソースイメージ情報参照のために Amazon ECS タスク定義で使用されます。

詳細はこちら:

- 設定パラメータと JSON/YAML スニペット例を表示する場合、Amazon ECR ソースアクションリファレンス を参照してください
- パイプライン、ステージ、アクションを作成する
- チュートリアル: Amazon ECR ソース、ECS - CodeDeploy 間のデプロイでパイプラインを作成する

Amazon S3 ソースアクション

Amazon S3 はインターネット用のストレージサービスです。Simple Storage Service (Amazon S3) を使用すると、いつでもウェブ上の任意の場所から任意の量のデータを保存および取得できます。バージョン管理された Amazon S3 バケットをコードのソースアクションとして使用するように CodePipeline を設定できます。

> **Note**
>
> Amazon S3 は、デプロイアクションとしてパイプラインに含めることもできます。

詳細はこちら:

- 設定パラメータと JSON/YAML スニペット例を表示する場合、Amazon S3 ソースアクションリファレンス を参照してください
- ステップ 1: アプリケーションの S3 バケットを作成する

- パイプラインを作成する (CLI)
- CodePipeline は、Amazon EventBridge (以前の Amazon CloudWatch Events) を使用して、Amazon S3 ソースバケットの変更を検出します。「CodePipeline との一般的統合」を参照してください。

Bitbucket Cloud、GitHub (GitHub アプリ経由)、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドへの接続

接続 (`CodeStarSourceConnection` アクション) は、サードパーティーの Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、または GitLab セルフマネージドリポジトリへのアクセスに使用されます。

> **Note**
>
> この機能は、アジアパシフィック (香港)、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 (ジャカルタ)、アジアパシフィック (メルボルン)、アジアパシフィック (大阪)、アフリカ (ケープタウン)、中東 (バーレーン)、中東 (アラブ首長国連邦)、欧州 (スペイン)、欧州 (チューリッヒ)、イスラエル (テルアビブ)、または AWS GovCloud (米国西部) の各リージョンでは使用できません。利用可能なその他のアクションについては、「CodePipeline との製品とサービスの統合」を参照してください。欧州 (ミラノ) リージョンでのこのアクションに関する考慮事項については、「CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアクションの場合)」の注意を参照してください。

Bitbucket Cloud	Bitbucket Cloud リポジトリをソースとして使用するように CodePipeline を設定することができます。これ以前に Bitbucket アカウントと少なくとも 1 つの Bitbucket Cloud リポジトリを作成しておく必要があります。Bitbucket Cloud リポジトリのソースアクションを追加するには、新しいパイプラインを作成するか、既存のパイプラインを編集します。 **Note** Bitbucket Cloud リポジトリへの接続を作成できます。Bitbucket サーバーなど、インストールされている Bitbucket プロバイダーのタイプはサポートされていません。

	パイプラインがサードパーティーのコードリポジトリにアクセスできるように、接続と呼ばれるリソースを設定できます。接続を作成する場合、コネクタアプリをサードパーティのコードリポジトリと共にインストールし、接続に関連付けます。 Bitbucket Cloud の場合は、コンソールの [Bitbucket] オプションまたは CLI の `CodestarSourceConnection` アクションを使用します。「Bitbucket Cloud への接続」を参照してください。 この 完全クローン作成 アクションのオプションを使用して、リポジトリの Git メタデータを参照して、ダウンストリームのアクションで Git コマンドを直接実行できるようにします。このオプションは、CodeBuild ダウンストリームアクションでのみ使用できます。 詳細はこちら: • 設定パラメータと JSON/YAML スニペット例を表示する場合、CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアクションの場合) を参照してください。 • Bitbucket Cloud ソースを使用してパイプラインを作成する入門チュートリアルを表示するには、接続入門ガイドを参照してください。
GitHub または GitHub Enterprise Cloud	GitHub のリポジトリをソースとして使用するように CodePipeline を設定することができます。これ以前に GitHub アカウントと少なくとも 1 つの GitHub リポジトリを作成しておく必要があります。GitHub リポジトリのソースアクションを追加するには、新しいパイプラインを作成するか、既存のパイプラインを編集します。 パイプラインがサードパーティーのコードリポジトリにアクセスできるように、接続と呼ばれるリソースを設定できます。接続を作成する場合、コネクタアプリをサードパーティのコードリポジトリと共にインストールし、接続に関連付けます。 コンソールまたは CLI 内の `CodestarSourceConnection` アクションにある GitHub (GitHub アプリ経由) プロバイダオプションを使用します。「GitHub コネクション」を参照してください。

	この 完全クローン作成 アクションのオプションを使用して、リポジトリの Git メタデータを参照して、ダウンストリームのアクションで Git コマンドを直接実行できるようにします。このオプションは、CodeBuild ダウンストリームアクションでのみ使用できます。 詳細はこちら: • 設定パラメータと JSON/YAML スニペット例を表示する場合、CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアクションの場合) を参照してください • GitHub リポジトリに接続して 完全クローン作成 オプションを使用する方法に関するチュートリアルには、チュートリアル: CodeCommit パイプラインソースで完全なクローンを使用する を参照してください。 • 現在の GitHub (GitHub アプリ経由) アクションは GitHub のバージョン 2 ソースアクションです。GitHub (OAuth アプリ経由) アクションは、OAuth トークン認証で管理されるバージョン 1 の GitHub アクションです。GitHub (OAuth アプリ経由) アクションの使用はお勧めしませんが、GitHub (OAuth アプリ経由) アクションを持つ既存のパイプラインは何の影響なく引き続き動作します。現在、GitHub アプリで GitHub ソースアクションを管理するパイプライン内で CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアクションの場合) ソースアクションを使用できます。GitHub (OAuth アプリ経由) アクションを使用するパイプラインがある場合は、GitHub (OAuth アプリ経由) ソースアクションを GitHub (GitHub アプリ経由) ソースアクションに更新する で GitHub (GitHub アプリ経由) アクションを使用するように更新する手順を参照してください。
GitHub Enterprise Server	コードのソースとして GitHub Enterprise Server リポジトリを使用するように CodePipeline を設定できます。これ以前に GitHub アカウントと少なくとも 1 つの GitHub リポジトリを作成しておく必要があります。GitHub Enterprise Server リポジトリのソースアクションを追加するには、新しいパイプラインを作成するか、既存のパイプラインを編集します。 パイプラインがサードパーティーのコードリポジトリにアクセスできるように、接続と呼ばれるリソースを設定できます。接続を作成する場合、コネク

	タアプリをサードパーティのコードリポジトリと共にインストールし、接続に関連付けます。 コンソールまたは CLI 内の `CodestarSourceConnection` アクションの にある GitHub (EnterpriseServer) プロバイダオプションを使用します。「GitHub Enterprise Server 接続」を参照してください。 この 完全クローン作成 アクションのオプションを使用して、リポジトリの Git メタデータを参照して、ダウンストリームのアクションで Git コマンドを直接実行できるようにします。このオプションは、CodeBuild ダウンストリームアクションでのみ使用できます。 詳細はこちら: • 設定パラメータと JSON/YAML スニペット例を表示する場合、CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアクションの場合) を参照してください • GitHub リポジトリに接続して 完全クローン作成 オプションを使用する方法に関するチュートリアルには、チュートリアル: CodeCommit パイプラインソースで完全なクローンを使用する を参照してください。
GitLab.com	GitLab.com リポジトリをソースとして使用するように CodePipeline を設定することができます。これ以前に GitLab.com アカウントと少なくとも 1 つの GitLab.com リポジトリを作成しておく必要があります。GitLab.com リポジトリのソースアクションを追加するには、新しいパイプラインを作成するか、既存のパイプラインを編集します。 コンソールの GitLab プロバイダーオプション、または CLI の `CodestarSourceConnection` アクションと`GitLab` プロバイダーを使用します。「GitLab.com への接続」を参照してください。 詳細はこちら: • 設定パラメータと JSON/YAML スニペット例を表示する場合、CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、Gi

	tLab.com、および GitLab セルフマネージドアクションの場合) を参照してください
GitLab セルフマネージド	GitLab セルフマネージドインストールをコードのソースとして使用するように CodePipeline を設定することができます。以前に GitLab アカウントを作成し、セルフマネージド GitLab (エンタープライズエディションまたはコミュニティエディション) のサブスクリプションを持っている必要があります。新しいパイプラインを作成するか、既存のパイプラインを編集することによって、GitLab セルフマネージドリポジトリのソースアクションを追加できます。 パイプラインがサードパーティーのコードリポジトリにアクセスできるように、接続と呼ばれるリソースを設定できます。接続を作成する場合、コネクタアプリをサードパーティのコードリポジトリと共にインストールし、接続に関連付けます。 コンソールまたは CLI 内の `CodestarSourceConnection` アクションにある GitHub セルフマネージドプロバイダーオプションを使用します。「GitLab セルフマネージドの接続」を参照してください。 この 完全クローン作成 アクションのオプションを使用して、リポジトリの Git メタデータを参照して、ダウンストリームのアクションで Git コマンドを直接実行できるようにします。このオプションは、CodeBuild ダウンストリームアクションでのみ使用できます。 詳細はこちら: • 設定パラメータと JSON/YAML スニペット例を表示する場合、CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアクションの場合) を参照してください • このプロバイダータイプとの接続を作成する手順については、「GitLab セルフマネージドの接続」を参照してください。

CodeCommit ソースアクション

CodeCommit は、クラウド内のアセット (ドキュメント、ソースコード、バイナリファイルなど) を非公開で保存および管理するために使用できるバージョン管理サービスです。コードのソースとして CodeCommit リポジトリ内のブランチを使用するように、CodePipeline を設定できます。リポジトリを作成し、ローカルマシン上の作業ディレクトリに関連付けます。次に、ステージのソースアクションの一部としてブランチを使用するパイプラインを作成できます。CodeCommit リポジトリに接続するには、新しいパイプラインを作成するか、既存のパイプラインを編集します。

この 完全クローン作成 アクションのオプションを使用して、リポジトリの Git メタデータを参照して、ダウンストリームのアクションで Git コマンドを直接実行できるようにします。このオプションは、CodeBuild ダウンストリームアクションでのみ使用できます。

詳細はこちら:

- 設定パラメータと JSON/YAML スニペット例を表示する場合、CodeCommit ソースアクションリファレンス を参照してください。
- チュートリアル: シンプルなパイプラインを作成する (CodeCommit リポジトリ)
- CodePipeline は Amazon CloudWatch Events を使用して、パイプラインのソースとして使用される CodeCommit リポジトリの変更を検出します。各ソースアクションには対応するイベントルールがあります。このイベントルールは、リポジトリで変更が発生したときにパイプラインを開始します。「CodePipeline との一般的統合」を参照してください。

GitHub (OAuth アプリ経由) ソースアクション

GitHub (OAuth アプリ経由) アクションは、OAuth アプリで管理されるバージョン 1 の GitHub アクションです。サービス利用可能地域では、GitHub アプリで GitHub ソースアクションを管理するパイプライン内で CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアクションの場合) ソースアクションを使用できます。GitHub (OAuth アプリ経由) アクションを使用するパイプラインがある場合は、GitHub (OAuth アプリ経由) ソースアクションを GitHub (GitHub アプリ経由) ソースアクションに更新する で GitHub (GitHub アプリ経由) アクションを使用するように更新する手順を参照してください。

> **Note**
>
> GitHub (OAuth アプリ経由) アクションの使用はお勧めしませんが、GitHub (OAuth アプリ経由) アクションを持つ既存のパイプラインは何の影響なく引き続き動作します。

詳細はこちら:

- アプリベースの GitHub (OAuth アプリ経由) アクセスとは対照的な OAuth ベースの GitHub アクセスの詳細については、https://docs.github.com/en/developers/apps/differences-between-github-apps-and-oauth-apps を参照してください。
- GitHub (OAuth アプリ経由) アクションの詳細を含む付録の表示については、「付録 A: GitHub (OAuth アプリ経由) ソースアクション」を参照してください。

ビルドアクションの統合

以下の情報は CodePipeline アクションの種類別に整理されていますので、CodePipeline を設定して以下のビルドアクションプロバイダーとの統合に役立ちます。

トピック

- CodeBuild ビルドアクション
- CloudBees ビルドアクション
- Amazon ECR のビルドアクションとパブリッシュアクション
- Jenkins ビルドアクション
- TeamCity ビルドアクション

CodeBuild ビルドアクション

CodeBuild は完全マネージド型の構築サービスです。ソースコードのコンパイル、ユニットテストの実行、すぐにデプロイできるアーティファクトの生成を行います。

CodeBuild を、ビルドアクションとしてパイプラインのビルドステージに追加できます。詳細については、AWS CodeBuild ビルドおよびテストアクションリファレンス の CodePipeline アクション設定リファレンスを参照してください。

> **Note**
>
> CodeBuild は、ビルド出力の有無にかかわらず、テストアクションとしてパイプラインに含めることもできます。

詳細はこちら:

- 設定パラメータと JSON/YAML スニペット例を表示する場合、AWS CodeBuild ビルドおよびテストアクションリファレンス を参照してください。
- CodeBuild とは
- CodeBuild - 完全マネージド型ビルドサービス

CloudBees ビルドアクション

CodePipeline を設定し、CloudBees を使用して、パイプラインの 1 つ以上のアクションでコードをビルドまたはテストできます。

詳細はこちら:

- re:INVENT 2017: Cloud First with AWS

Amazon ECR のビルドアクションとパブリッシュアクション

Amazon ECR は Docker AWS イメージリポジトリサービスです。Docker イメージをリポジトリにアップロードするには、Docker のプッシュコマンドおよびプルコマンドを使用します。

パイプラインに ECRBuildAndPublish アクションを追加して、イメージの構築とプッシュを自動化できます。詳細については、ECRBuildAndPublish ビルドアクションリファレンス の CodePipeline アクション設定リファレンスを参照してください。

Jenkins ビルドアクション

CodePipeline を設定し、Jenkins CI を使用して、パイプラインの 1 つ以上のアクションでコードを作成またはテストできます。これ以前に Jenkins プロジェクトを作成し、そのプロジェクト用に CodePipeline プラグインをインストールして設定しておく必要があります。Jenkins プロジェクトに接続するには、新しいパイプラインを作成するか、既存のパイプラインを編集します。

Jenkins のアクセスは、プロジェクトベースで設定されます。CodePipeline で使用する Jenkins インスタンスには、CodePipeline Plugin for Jenkins をインストールする必要があります。また、Jenkins プロジェクトへの CodePipeline のアクセスを設定する必要があります。HTTPS/SSL 接続のみを受け入れるように設定して、Jenkins プロジェクトを安全に保護します。Jenkins プロジェクトが Amazon EC2 インスタンスにインストールされている場合は、各インスタンス AWS CLI に をインストールして AWS 認証情報を提供することを検討してください。次に、接続に使用する認証情報を使用して、それらのインスタンスで AWS プロファイルを設定します。これは、Jenkins ウェブインターフェイスを介した追加と保存の代替手段です。

詳細はこちら:

- Jenkins のアクセス
- チュートリアル: 4 ステージのパイプラインを作成する

TeamCity ビルドアクション

CodePipeline を設定し、TeamCity を使用して、パイプラインの 1 つ以上のアクションでコードを作成またはテストできます。

詳細はこちら:

- CodePipeline 用 TeamCity プラグイン

テストアクションの統合

以下の情報は CodePipeline アクションの種類別に整理されていますので、CodePipeline を設定して以下のビルドアクションプロバイダーとの統合に役立ちます。

トピック

- CodeBuild またはテストアクション
- AWS Device Farm テストアクション
- Ghost Inspector テストアクション
- OpenText LoadRunner Cloud テストアクション
- リフレクトテストの自動化

CodeBuild またはテストアクション

CodeBuild とは、クラウド上のフルマネージドビルドサービスです。CodeBuild はソースコードをコンパイルし、単体テストを実行して、すぐにデプロイできるアーティファクトを生成します。

テストアクションとしてパイプラインに CodeBuild を追加できます。詳細については、AWS CodeBuild ビルドおよびテストアクションリファレンス の CodePipeline アクション設定リファレンスを参照してください。

> **Note**
>
> CodeBuild は、必須ビルド出力アーティファクトを持つビルドアクションとしてパイプラインに含めることもできます。

詳細はこちら:

- 設定パラメータと JSON/YAML スニペット例を表示する場合、AWS CodeBuild ビルドおよびテストアクションリファレンス を参照してください。
- CodeBuild とは

AWS Device Farm テストアクション

AWS Device Farm は、実際に電話やタブレットで、Android や iOS、およびウェブアプリを物理的にテストしてやり取りできるアプリテストサービスです。パイプラインの 1 つ以上のアクションでコードをテスト AWS Device Farm するために を使用するように CodePipeline を設定できます。 AWS Device Farm では、独自のテストをアップロードしたり、組み込みのスクリプトフリーの互換性テストを使用したりできます。テストは並列実行されるため、テストは複数のデバイスで数分のうちに開始されます。高レベルの結果、低レベルのログ、pixel-to-pixelスクリーンショット、パフォーマンスデータを含むテストレポートは、テストが完了すると更新されます。は、PhoneGap、Titanium、Xamarin、Unity、およびその他のフレームワークで作成されたものを含む、ネイティブおよびハイブリッドの Android、iOS、および Fire OS アプリケーションのテスト AWS Device Farm をサポートしています。Android アプリのリモートアクセスをサポートしているため、テストデバイスと直接やり取りすることができます。

詳細はこちら:

- 設定パラメータと JSON/YAML スニペット例を表示する場合、AWS Device Farm テストアクションリファレンス を参照してください。
- とは AWS Device Farm
- CodePipeline テストステージ AWS Device Farm での の使用

Ghost Inspector テストアクション

CodePipeline を設定し、Ghost Inspector を使用して、パイプラインの 1 つ以上のアクションでコードをテストできます。

詳細はこちら:

- CodePipeline とのサービス統合用の Ghost Inspector のドキュメント

OpenText LoadRunner Cloud テストアクション

パイプラインの 1 つ以上のアクションで OpenText LoadRunner Cloud を使用するように CodePipeline を設定できます。

詳細はこちら:

- CodePipeline と統合するための LoadRunner Cloud ドキュメント

リフレクトテストの自動化

リフレクトは、AI を活用したテスト自動化ソリューションです。これにより、テストを簡素化し、手動プロセスの課題を克服できます。ノーコードテストの自動化により、リフレクトはテストの作成、実行、メンテナンスを合理化し、技術的知識を必要とせずに堅牢で反復可能なテストを作成できます。複雑さを排除し、ワークフローの中断を最小限に抑えることで、毎回テストを高速化し、高品質のアプリケーションを自信を持って提供できます。

詳細はこちら:

- AWS CodePipeline Reflect との統合をテストする

デプロイアクションの統合

以下の情報は CodePipeline アクションの種類別に整理されていますので、CodePipeline を設定して以下のデプロイアクションプロバイダーとの統合に役立ちます。

トピック

- Amazon EC2 デプロイアクション
- Amazon Elastic Kubernetes Service の EKS デプロイアクション
- Amazon S3 デプロイアクション
- AWS AppConfig デプロイアクション
- CloudFormation アクションをデプロイする
- CloudFormation StackSets デプロイアクション

- Amazon ECS デプロイアクション
- Elastic Beanstalk デプロイアクション
- OpsWorks アクションをデプロイする
- Service Catalog のデプロイアクション
- Amazon Alexa デプロイアクション
- CodeDeploy デプロイアクション
- XebiaLabs デプロイアクション

Amazon EC2 デプロイアクション

Amazon EC2 では、クラウドでコンピューティングの作成と管理を行えます。アプリケーションをインスタンスにデプロイするデプロイプロバイダーとして Amazon EC2 を使用するパイプラインにアクションを追加できます。

詳細はこちら:

- 「Amazon EC2 アクションリファレンス」のアクションリファレンスページを参照してください。
- チュートリアルについては、「チュートリアル: CodePipeline を使用して Amazon EC2 インスタンスにデプロイする」を参照してください。

Amazon Elastic Kubernetes Service の EKS デプロイアクション

Amazon EKS では、Kubernetes クラスターを作成および管理できます。イメージをクラスターにデプロイするデプロイプロバイダーとして Amazon EKS を使用するパイプラインにアクションを追加できます。Helm テンプレートまたは Kubernetes マニフェストファイルを使用できます。

詳細はこちら:

- 「Amazon Elastic Kubernetes Service の EKS デプロイアクションリファレンス」のアクションリファレンスページを参照してください。
- チュートリアルについては、「チュートリアル: CodePipeline を使用して Amazon EKS にデプロイする」を参照してください。

Amazon S3 デプロイアクション

Amazon S3 はインターネット用のストレージサービスです。Simple Storage Service (Amazon S3) を使用すると、いつでもウェブ上の任意の場所から任意の量のデータを保存および取得できます。デプロイプロバイダーとして Amazon S3 を使用するパイプラインにアクションを追加できます。

> **Note**
>
> ソースアクションとして Amazon S3 をパイプラインに含めることもできます。

詳細はこちら:

- パイプライン、ステージ、アクションを作成する
- チュートリアル: Amazon S3 をデプロイプロバイダとして使用するパイプラインを作成する

AWS AppConfig デプロイアクション

AWS AppConfig は、アプリケーション設定を作成、管理、迅速にデプロイ AWS Systems Manager する の機能です。AppConfig は、EC2 インスタンス、コンテナ AWS Lambda、モバイルアプリケーション、または IoT デバイスでホストされているアプリケーションで使用できます。

詳細はこちら:

- AWS AppConfig デプロイアクションリファレンス の CodePipeline アクション設定リファレンス
- チュートリアル: AWS AppConfig をデプロイプロバイダーとして使用するパイプラインを作成する

CloudFormation アクションをデプロイする

CloudFormation を使用すると、開発者やシステム管理者は、 テンプレートを使用して関連 AWS リソースのコレクションを簡単に作成および管理し、それらのリソースをプロビジョニングおよび更新できます。サービスのサンプルテンプレートを使用することも、独自のテンプレートを作成することもできます。テンプレートは、アプリケーションの実行に必要な AWS リソースと依存関係またはランタイムパラメータを記述します。

AWS サーバーレスアプリケーションモデル (AWS SAM) を拡張 CloudFormation して、サーバーレスアプリケーションの定義とデプロイを簡素化します。 AWS SAM は、Amazon API Gateway APIs、 AWS Lambda 関数、Amazon DynamoDB テーブルをサポートしています。CodePipeline を

および SAM AWS とともに CloudFormation 使用して、サーバーレスアプリケーションを継続的に配信できます。

デプロイプロバイダー AWS CloudFormation として を使用するパイプラインにアクションを追加できます。をデプロイプロバイダー AWS CloudFormation として使用すると、パイプライン実行の一部として CloudFormation スタックと変更セットに対してアクションを実行できます。 は、パイプラインの実行時にスタックと変更セットを作成、更新、置換、削除 CloudFormation できます。その結果、 CloudFormation テンプレート AWS とパラメータ定義で指定した仕様に従って、パイプラインの実行中にカスタムリソースを作成、プロビジョニング、更新、終了できます。

詳細はこちら:

- CloudFormation デプロイアクションリファレンス の CodePipeline アクション設定リファレンス
- CodePipeline を使用した継続的デリバリー — CodePipeline を使用して継続的デリバリーワークフローを構築する方法について説明します CloudFormation。
- Lambda ベースのアプリケーションのデプロイの自動化 — AWS サーバーレスアプリケーションモデルと CloudFormation を使用して、Lambda ベースのアプリケーションの継続的な配信ワークフローを構築する方法について説明します。

CloudFormation StackSets デプロイアクション

CloudFormation では、複数のアカウントと AWS リージョンにリソースをデプロイできます。

で CodePipeline を使用して CloudFormation スタックセット定義を更新し、インスタンスに更新をデプロイできます。

次のアクションをパイプラインに追加して、 AWS CloudFormation StackSets をデプロイプロバイダーとして使用できます。

- CloudFormations スタックセット
- CloudFormations スタックインスタンス

詳細はこちら:

- CloudFormation StackSets デプロイアクションリファレンス の CodePipeline アクション設定リファレンス
- チュートリアル: AWS CloudFormation StackSets デプロイアクションを使用してパイプラインを作成する

Amazon ECS デプロイアクション

Amazon ECS は、スケーラビリティに優れた高性能なコンテナ管理サービスであり、 AWS クラウドでコンテナベースのアプリケーションを実行することができます。パイプラインを作成すると、デプロイプロバイダとして Amazon ECS を選択できます。ソースコントロールリポジトリのコードを変更すると、パイプラインが新しい Docker イメージを作成し、コンテナレジストリにプッシュし、更新されたイメージを Amazon ECS にデプロイします。また、CodePipeline の ECS (Blue/Green) プロバイダアクションを使用して、CodeDeploy でトラフィックを Amazon ECS にルーティングおよびデプロイすることもできます。

詳細はこちら :

- Amazon ECS とは
- チュートリアル: CodePipeline を使用した継続的なデプロイ
- パイプライン、ステージ、アクションを作成する
- チュートリアル: Amazon ECR ソース、ECS - CodeDeploy 間のデプロイでパイプラインを作成する

Elastic Beanstalk デプロイアクション

Elastic Beanstalk は、Java、.NET、PHP、Node.js、Python、Ruby、Go、Docker で開発されたウェブアプリケーションとサービスを、Apache、Nginx、Passenger、IIS などの一般的なサーバーにデプロイしてスケーリングするサービスです。Elastic Beanstalk を使用してコードをデプロイするように CodePipeline を設定できます。パイプライン作成前、または パイプライン の作成ウィザードを使用する際、ステージのデプロイアクションで使用する Elastic Beanstalk アプリケーションと環境を作成できます。

> **Note**
>
> この機能は、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)、中東 (アラブ首長国連邦)、欧州 (スペイン)、または欧州 (チューリッヒ) リージョンでは利用できません。利用可能なその他のアクションについては、「CodePipeline との製品とサービスの統合」を参照してください。

詳細はこちら:

- Elastic Beanstalk を使用して開始する

- パイプライン、ステージ、アクションを作成する

OpsWorks アクションをデプロイする

OpsWorks は、Chef を使用してあらゆる形状とサイズのアプリケーションを設定および運用するのに役立つ設定管理サービスです。を使用すると AWS OpsWorks Stacks、パッケージのインストール、ソフトウェア設定、ストレージなどのリソースなど、アプリケーションのアーキテクチャと各コンポーネントの仕様を定義できます。CodePipeline を設定 AWS OpsWorks Stacks して、 でカスタム Chef クックブックとアプリケーションと組み合わせてコードをデプロイできます OpsWorks。

- カスタム Chef クックブック – Chef クックブック OpsWorks を使用して、パッケージのインストールと設定、アプリケーションのデプロイなどのタスクを処理します。
- アプリケーション – OpsWorks アプリケーションは、アプリケーションサーバーで実行するコードで構成されます。アプリケーションコードは、Amazon S3 バケットなどのリポジトリに格納されています。

パイプラインを作成する前に、 OpsWorks スタックとレイヤーを作成します。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を使用するときに、ステージのデプロイアクションで使用する OpsWorks アプリケーションを作成できます。

の CodePipeline サポート OpsWorks は現在、米国東部 (バージニア北部) リージョン (us-east-1) でのみ利用できます。

詳細はこちら:

- AWS OpsWorks Stacksでの CodePipeline
- クックブックとレシピ
- AWS OpsWorks アプリケーション

Service Catalog のデプロイアクション

Service Catalog を使用すると、組織は での使用が承認された製品のカタログを作成および管理できます AWS。

製品テンプレートの更新とバージョンを Service Catalog にデプロイするように CodePipeline を設定できます。デプロイアクションで使用する Service Catalog 製品を作成したら、[パイプラインを作成する] ウィザードを使用してパイプラインを作成できます。

詳細はこちら:

- チュートリアル: Service Catalog にデプロイするパイプラインを作成する
- パイプライン、ステージ、アクションを作成する

Amazon Alexa デプロイアクション

Amazon Alexa Skills Kit を使用すると、クラウドベースのスキルをビルドし、Alexa 対応デバイスのユーザーに配布できます。

> **Note**
>
> この特徴は、アジアパシフィック (香港) またはヨーロッパ (ミラノ) リージョンでは使用できません。当該地域で使用可能な他のデプロイアクションを使用する場合、デプロイアクションの統合 を参照してください。

デプロイプロバイダとして Alexa Skills Kit を使用するパイプラインにアクションを追加できます。パイプラインによってソースの変更が検出され、更新が Alexa サービスの Alexa スキルにデプロイされます。

詳細はこちら:

- チュートリアル: Amazon Alexa Skill をデプロイするパイプラインを作成する

CodeDeploy デプロイアクション

CodeDeploy は、Amazon EC2/オンプレミスインスタンス、Amazon Elastic Container Service コンピューティングプラットフォーム、サーバーレス AWS Lambda コンピューティングプラットフォームへのアプリケーションのデプロイを調整します。CodePipeline を設定して、 CodeDeploy でコードをデプロイすることができます。パイプラインを作成する前または [パイプラインの作成] ウィザードを使用するときに、デプロイアクションで使用できる 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:

- ステップ 3: CodeDeploy でアプリケーションを作成する

- チュートリアル: シンプルなパイプラインを作成する (CodeCommit リポジトリ)

XebiaLabs デプロイアクション

CodePipeline を設定し、 XebiaLabs を使用して、パイプラインの 1 つ以上のアクションでコードをデプロイできます。

詳細はこちら:

- CodePipeline で XL デプロイを使用する

Amazon Simple Notification Service との承認アクションの統合

Amazon SNS は、高速かつ柔軟な完全マネージド型のプッシュ通知サービスです。このサービスを使用すると、個々のメッセージを送信したり、多数の受信者にメッセージをファンアウトしたりできます。Amazon SNS により、簡単かつコスト効率の高い方法で、モバイルデバイスユーザーおよびメール受信者にプッシュ通知を送信したり、他の分散サービスにメッセージを送信したりできます。

CodePipeline で手動承認リクエストを作成する場合は、必要に応じて Amazon SNS にトピックを発行して、サブスクライブしているすべての IAM ユーザーに承認アクションを確認する準備ができたことが通知されます。

詳細はこちら:

- Amazon SNS とは
- Amazon SNS アクセス権限を CodePipeline サービスロールに付与する

呼び出しアクションの統合

以下の情報は CodePipeline アクションの種類別に整理されていますので、CodePipeline を設定して、以下の呼び出しアクションプロバイダーとの統合に役立ちます。

トピック

- Amazon Inspector の呼び出しアクション
- Lambda 呼び出しアクション
- Step Functions アクション呼び出し

Amazon Inspector の呼び出しアクション

Amazon Inspector は、ソフトウェアの脆弱性や意図しないネットワークの露出についてワークロードを自動的に検出し、継続的にスキャンする脆弱性管理サービスです。Amazon Inspector は tar や war などの複数のアーカイブ形式をサポートします。また、Amazon Inspector は Rust や Go バイナリなどのバイナリをサポートします。

CodePipeline の `InspectorScan` アクションを設定して、ソースコードまたは Amazon ECR イメージリポジトリの脆弱性のスキャンを自動化できます。

詳細はこちら:

- Amazon Inspector `InspectorScan` 呼び出しアクションリファレンス の CodePipeline アクション設定リファレンス

Lambda 呼び出しアクション

Lambda を使用することで、サーバーのプロビジョニングや管理をすることなく、コードを実行できます。Lambda 関数を使用してパイプラインに柔軟性と機能性を追加するように CodePipeline を設定できます。パイプライン作成前、または パイプライン作成 ウィザードを使用する際、ステージにアクションとして追加する Lambda 関数を作成できます。

詳細はこちら:

- AWS Lambda アクションリファレンスを呼び出す の CodePipeline アクション設定リファレンス
- CodePipeline のパイプラインで AWS Lambda 関数を呼び出す

Step Functions アクション呼び出し

Step Functions では、ステートマシンの作成と設定ができます。CodePipeline では、ステップファンクションでアクションを呼び出し、ステートマシンの実行をトリガーするように設定できます。

詳細はこちら:

- AWS Step Functions アクションリファレンスを呼び出す の CodePipeline アクション設定リファレンス
- チュートリアル: パイプラインで AWS Step Functions 呼び出しアクションを使用する

CodePipeline との一般的統合

次の AWS のサービス 統合は、CodePipeline アクションタイプに基づいていません。

Amazon CloudWatch	Amazon CloudWatch は AWS リソースをモニタリングします。 詳細はこちら: • Amazon CloudWatch とは
Amazon EventBridge	Amazon EventBridge は、定義したルール AWS のサービス に基づいての変更を検出し、変更が発生した AWS のサービス ときに指定された 1 つ以上の でアクションを呼び出すウェブサービスです。 • 何かが変更されたときに自動的にパイプライン実行を開始する - Amazon EventBridge で設定されたルールのターゲットとして CodePipeline を設定できます。これにより、別のサービスが変更されたときに自動的にパイプラインが開始されるように設定されます。 詳細はこちら: • Amazon EventBridge とは • CodePipeline でパイプラインを編集する. • CodeCommit ソースアクションと EventBridge • パイプラインの状態が変更されたときに通知を受け取る - EventBridge ルールを設定して、パイプライン、ステージ、またはアクションの実行状態の変更を検出して対応することができます。 詳細はこちら: • CodePipeline イベントのモニタリング • チュートリアル: CloudWatch Events ルールをセットアップし、パイプラインの状態の変更の E メール通知を送信します。
AWS Cloud9	AWS Cloud9 は、ウェブブラウザからアクセスするオンライン IDE です。この IDE では、リッチなコード編集エクスペリエンスを実現しており、複数のプログラミング言語、ランタイムデバッガ、組み込みターミナルがサポートされています。バックグラウンドでは、Amazon

	EC2 インスタンスは AWS Cloud9 開発環境をホストします。詳細については、「AWS Cloud9 ユーザーガイド」を参照してください。 詳細はこちら: • AWS Cloud9のセットアップ
AWS CloudTrail	CloudTrail は、アカウントによって、または AWS アカウントに代わって行われた AWS API コールおよび関連イベントをキャプチャし、指定した Amazon S3 バケットにログファイルを配信します。CodePipeline コンソールからの API コール、 からの CodePipeline コマンド AWS CLI、および CodePipeline API からの API コールをキャプチャするように CloudTrail を設定できます。 詳細はこちら: • を使用した CodePipeline API コールのログ記録 AWS CloudTrail
AWS CodeStar 通知	パイプラインの実行開始時など、重要な変更をユーザーに知らせるための通知を設定できます。詳細については、「通知ルールの作成」を参照してください。

AWS Key Management Service	AWS KMS は、データの暗号化に使用される暗号化キーの作成と管理を容易にするマネージド型サービスです。デフォルトでは、CodePipeline は AWS KMS を使用して Amazon S3 バケットに保存されているパイプラインのアーティファクトを暗号化します。 詳細はこちら: • 1 つのアカウントからソースバケット、アーティファクトバケット、およびサービスロールを使用し、別の AWS アカウントから CodeDeploy リソースを使用するパイプラインを作成するには AWS 、カスタマーマネージド KMS キーを作成し、パイプラインにキーを追加して、クロスアカウントアクセスを有効にするようにアカウントポリシーとロールを設定する必要があります。詳細については、「別の AWS アカウントのリソースを使用する CodePipeline でパイプラインを作成する」を参照してください。 • CloudFormation スタックを別のアカウントにデプロイする AWS アカウントからパイプラインを作成するには AWS 、カスタマーマネージド KMS キーを作成し、そのキーをパイプラインに追加して、スタックを別の AWS アカウントにデプロイするアカウントポリシーとロールを設定する必要があります。詳細については、CodePipeline を使用して AWS CloudFormation スタックを別のアカウントにデプロイする方法」を参照してください。 • パイプラインの S3 アーティファクトバケットのサーバー側の暗号化を設定するには、デフォルトの AWS マネージド KMS キーを使用するか、カスタマーマネージド KMS キーを作成して、暗号化キーを使用するようにバケットポリシーを設定できます。詳細については、「CodePipeline 用に Amazon S3 に保存したアーティファクトのサーバー側の暗号化を設定する」を参照してください。 AWS KMS keyでは、キー ID、キー ARN、またはエイリアス ARN を使用できます。 **Note** エイリアスは、 KMS キー を作成したアカウントでのみ認識されます。クロスアカウントアクションの場合、キー ID または

キー ARN のみを使用してキーを識別できます。クロスアカウントアクションには他のアカウント (AccountB) のロールを使用するため、キー ID を指定すると他のアカウント (AccountB) のキーが使用されます。

コミュニティからの例

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おり、包括的なリストや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や正確性について責任を負いません。

トピック

- 統合の例: ブログ投稿

統合の例: ブログ投稿

- サードパーティーの Git リポジトリからの AWS CodePipeline ビルドステータスの追跡

  サードパーティーのリポジトリにパイプラインとビルドアクションのステータスを表示するリソースを設定して、デベロッパーがコンテキストを切り替えずにステータスを簡単に追跡できるようにする方法を説明します。

  2017 年 3 月発行

- AWS CodeCommit、、および を使用して CI/CD AWS CodeBuildAWS CodeDeployを完了する AWS CodePipeline

  CodeCommit、CodePipeline、CodeBuild、および CodeDeploy の各サービスを使用して、バージョン管理された Java アプリケーションをコンパイル、ビルド、および一連の Amazon EC2 Linux インスタンスにインストールするパイプラインを設定する方法について説明します。

  2015 年 9 月公開

- AWS Step Functions の CI/CD パイプラインのテストと作成

  Step Functions ステートマシンとパイプラインを調整するリソースの設定方法について説明します。

  2020 年 3 月発行

- CodePipeline を使用した DevSecOps

  予防的および発見的なセキュリティ制御を自動化するために CodePipeline で CI/CD パイプラインを使用する方法について説明します。この投稿では、パイプラインを使用してシンプルなセキュリティグループを作成し、ソース、テスト、本番環境の段階でセキュリティチェックを実行して、AWS アカウントのセキュリティ体制を改善する方法について説明します。

  2017 年 3 月発行

- CodePipeline、CodeBuild、Amazon ECR、および を使用した Amazon ECS への継続的なデプロイ CloudFormation

  Amazon Elastic Container Service (Amazon ECS) への継続的デプロイパイプラインの作成方法について説明します。アプリケーションは、CodePipeline、CodeBuild、Amazon ECR、および CloudFormationを使用して Docker コンテナとして配信されます。

  - サンプル CloudFormation テンプレートとそれを使用して、GitHub の ECS リファレンスアーキテクチャ: 継続的デプロイリポジトリから独自の継続的デプロイパイプラインを作成する手順をダウンロードします。

  2017 年 1 月投稿

- サーバーレスアプリケーションの継続的なデプロイ

  のコレクションを使用して AWS のサービス 、サーバーレスアプリケーションの継続的なデプロイパイプラインを作成する方法について説明します。サーバーレスアプリケーションモデル (SAM) を使用してアプリケーションとそのリソースを定義し、CodePipeline はアプリケーションのデプロイを調整します。

  - Gin フレームワークと API ゲートウェイプロキシシムでの実行で書かれたサンプルアプリケーションの表示

  発行日：2016年12月

- CodePipeline と Dynatrace を使用した DevOps デプロイメントのスケーリング

Dynatrace のモニタリングソリューションを使用して、CodePipeline のパイプラインを拡張し、コードがコミットされる前にテストの実行を自動的に分析し、最適なリードタイムを維持する方法について説明します。

2016 年 11 月公開

- AWS CloudFormation と CodeCommit を使用して CodePipeline AWS Elastic Beanstalk で のパイプラインを作成する CodeCommit

  AWS Elastic Beanstalkのアプリケーション用に継続的デリバリーを CodePipeline パイプラインに実装する方法について説明します。すべての AWS リソースは、 テンプレートを使用して CloudFormation 自動的にプロビジョニングされます。このウォークスルーには、CodeCommit と AWS Identity and Access Management (IAM) も組み込まれています。

2016 年 5 月投稿

- で CodeCommit と CodePipeline を自動化する CloudFormation

  CloudFormation を使用して、CodeCommit、CodePipeline CodeDeploy 、および を使用する継続的デリバリーパイプラインの AWS リソースのプロビジョニングを自動化します AWS Identity and Access Management。

2016 年 4 月公開

- でクロスアカウントパイプラインを作成する AWS CodePipeline

  AWS Identity and Access Managementを使用して AWS CodePipeline のパイプラインへのクロスアカウントアクセスのプロビジョニングを自動化する方法について説明します。 CloudFormation テンプレートに例を含めます。

2016 年 3 月発行

- ASP.NET コアパート 2 の学習: 継続的デリバリー

  CodeDeploy と を使用して ASP.NET Core アプリケーションの完全な継続的デリバリーシステムを作成する方法について説明します AWS CodePipeline。

2016 年 3 月発行

- AWS CodePipeline コンソールを使用してパイプラインを作成する

AWS CodePipeline コンソールを使用して、 に基づくウォークスルーで 2 ステージパイプラインを作成する方法について説明します AWS CodePipeline チュートリアル: 4 ステージのパイプラインを作成する。

2016 年 3 月発行

- を使用した AWS CodePipeline パイプラインのモッキング AWS Lambda

パイプラインが動作する前に、CodePipeline ソフトウェア配信プロセスの設計時に、このプロセスのアクションやステージを視覚化できる Lambda 関数を呼び出す方法について説明します。パイプライン構造を設計するときに、Lambda 関数を使用して、パイプラインが正常に完了するかどうかをテストできます。

2016 年 2 月投稿

- を使用した CodePipeline での AWS Lambda 関数の実行 AWS CloudFormation

ユーザーガイドタスク で使用されるすべての AWS リソースをプロビジョニングする AWS CloudFormation スタックを作成する方法について説明しますCodePipeline のパイプラインで AWS Lambda 関数を呼び出す。

2016 年 2 月投稿

- でのカスタム CodePipeline アクションのプロビジョニング AWS CloudFormation

を使用して CodePipeline でカスタムアクション AWS CloudFormation 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

- を使用した CodePipeline のプロビジョニング AWS CloudFormation

AWS CloudFormationを使用して CodePipeline に基本的な継続的デリバリーパイプラインをプロビジョニングする方法について説明します。

発行日：2015年12月

- カスタムアクション OpsWorks を使用して CodePipeline から にデプロイする AWS Lambda

CodePipeline AWS OpsWorks を使用して にデプロイするパイプラインと AWS Lambda 関数を設定する方法について説明します。

2015 年 7 月発行

CodePipeline チュートリアル

のステップを完了したらCodePipeline の使用開始、このユーザーガイドの AWS CodePipeline チュートリアルのいずれかを試すことができます。

トピック

- チュートリアル: CodePipeline を使用して Amazon EC2 インスタンスにデプロイする
- チュートリアル: CodePipeline (V2 タイプ) を使用して Docker イメージを構築し Amazon ECR にプッシュする
- チュートリアル: CodePipeline を使用して Amazon EKS にデプロイする
- チュートリアル: コンピューティングを使用してコマンドを実行するパイプラインを作成する (V2 タイプ)
- チュートリアル: Git タグを使用してパイプラインを開始する
- チュートリアル: プルリクエストのブランチ名をフィルタリングしてパイプラインを開始する (V2 タイプ)
- チュートリアル: パイプラインレベルの変数を使用する
- チュートリアル: シンプルなパイプラインを作成する (S3 バケット)
- チュートリアル: シンプルなパイプラインを作成する (CodeCommit リポジトリ)
- チュートリアル: 4 ステージのパイプラインを作成する
- チュートリアル: CloudWatch Events ルールをセットアップし、パイプラインの状態の変更の E メール通知を送信します。
- チュートリアル: を使用して Android アプリを構築およびテストするパイプラインを作成する AWS Device Farm
- チュートリアル: で iOS アプリをテストするパイプラインを作成する AWS Device Farm
- チュートリアル: Service Catalog にデプロイするパイプラインを作成する
- チュートリアル: を使用してパイプラインを作成する AWS CloudFormation
- チュートリアル: AWS CloudFormation デプロイアクションの変数を使用するパイプラインを作成する
- チュートリアル: CodePipeline を使用した Amazon ECS 標準デプロイ
- チュートリアル: Amazon ECR ソース、ECS - CodeDeploy 間のデプロイでパイプラインを作成する

- チュートリアル: Amazon Alexa Skill をデプロイするパイプラインを作成する
- チュートリアル: Amazon S3 をデプロイプロバイダとして使用するパイプラインを作成する
- チュートリアル: サーバーレスアプリケーションを に発行するパイプラインを作成する AWS Serverless Application Repository
- チュートリアル: CodePipeline を使用した Lambda 関数のデプロイ
- チュートリアル: Lambda 呼び出しアクションで変数を使用する
- チュートリアル: パイプラインで AWS Step Functions 呼び出しアクションを使用する
- チュートリアル: AWS AppConfig をデプロイプロバイダーとして使用するパイプラインを作成する
- チュートリアル: CodeCommit パイプラインソースで完全なクローンを使用する
- チュートリアル: CodeCommit パイプラインソースでフルクローンを使用する
- チュートリアル: AWS CloudFormation StackSets デプロイアクションを使用してパイプラインを作成する
- チュートリアル: パイプラインの変数チェックルールを入力条件として作成する

チュートリアル: CodePipeline を使用して Amazon EC2 インスタンスにデプロイする

このチュートリアルは、Amazon EC2 で設定したインスタンスにコードをデプロイするデプロイアクションを CodePipeline で作成する方法について説明します。

> **Note**
>
> コンソールでのパイプライン作成の一環として、CodePipeline は S3 アーティファクトバケットをアーティファクトとして使用します (これは S3 ソースアクションで使用するバケットとは異なります)。S3 アーティファクトバケットがパイプラインのアカウントとは異なるアカウントにある場合は、S3 アーティファクトバケットが によって所有 AWS アカウントされており、安全で信頼できることを確認してください。

> **Note**
>
> EC2 デプロイアクションは V2 タイプのパイプラインでのみ使用できます。

前提条件

このチュートリアルで CD パイプラインを作成する前に、いつくかのリソースを用意する必要があります。使用を開始するために必要なものは以下のとおりです。

> **Note**
>
> これらのリソースはすべて、同じ AWS リージョン内に作成する必要があります。

- サンプル `script.sh` ファイルを追加するソースコントロールリポジトリ (このチュートリアルでは GitHub を使用します) 。
- このアクションのアクセス許可で更新された既存の CodePipeline サービスロールを使用する必要があります。サービスロールを更新するには、「EC2 デプロイアクションのサービスロールポリシーのアクセス許可」を参照してください。

これらの前提条件を満たした後、チュートリアルに進んで CD パイプラインを作成できます。

ステップ 1: Amazon EC2 Linux インスタンスを作成する

このステップでは、サンプルアプリケーションをデプロイする Amazon EC2 インスタンスを作成します。このプロセスの一環として、リソースを作成するリージョンでインスタンスロールをまだ作成していない場合は、IAM でインスタンスロールを作成します。

インスタンス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. コンソールダッシュボードで [ロール] を選択します。
3. [ロールの作成] を選択してください。
4. [信頼されたエンティティのタイプを選択] で、[AWS のサービス] を選択します。[ユースケースの選択] で、[EC2] を選択します。[Select your use case (ユースケースを選択)] で、[EC2] を選択します。[次へ] を選択します。
5. **`AmazonSSMManagedEC2InstanceDefaultPolicy`** という名前のマネージドポリシーを検索して選択します。
6. **`AmazonSSMManagedInstanceCore`** という名前のマネージドポリシーを検索して選択します。[Next: Tags (次へ: タグ)] を選択します。

7. [次へ: レビュー] を選択します。ロールの名前を入力します (例: **EC2InstanceRole**)。

   > **Note**
   >
   > 次のステップのロール名をメモしておきます。このロールは、インスタンスの作成時に選択します。

   > **Note**
   >
   > このロールにアクセス許可を追加して、パイプラインの作成後にパイプラインの S3 アーティファクトバケットへのアクセスを許可します。

   [ロールの作成] を選択してください。

インスタンスを起動するには

1. Amazon EC2 コンソールの https://console.aws.amazon.com/ec2/ を開いてください。
2. サイドナビゲーションから [インスタンス] を選択し、ページの上部から [インスタンスの起動] を選択します。
3. [名前] に「**MyInstances**」と入力します。これにより、インスタンスにはキーが **Name** で、値が **MyInstances** というタグがが割り当てられます。
4. アプリケーションイメージと OS イメージ (Amazon マシンイメージ) で、 AWS ロゴが付いた Amazon Linux AMI オプションを見つけ、選択されていることを確認します。(この AMI は Amazon Linux 2 AMI (HVM) と表記され、「無料利用枠対象」と表示されています。)
5. [インスタンスタイプ] で、インスタンスのハードウェア構成として無料利用枠対象となる `t2.micro` タイプを選択します。
6. [キーペア (ログイン)] で、キーペアを選択するか作成します。
7. [ネットワーク設定] で、ステータスが [有効] であることを確認します。
8. [Advanced Details] (高度な詳細) を展開します。[IAM インスタンスプロファイル] で、前の手順で作成した IAM ロール (**EC2InstanceRole** など) を選択します。

> **Note**
>
> インスタンスロールを空白のままにしないでください。デフォルトのロールが作成され、作成したロールが選択されなくなります。

9. [概要] の [インスタンス数] に 2 と入力します。
10. [インスタンスを起動] を選択します。
11. [インスタンス] ページで、起動のステータスを表示できます。インスタンスを起動すると、その初期状態は `pending` です。インスタンスを起動した後は、状態が `running` に変わり、パブリック DNS 名を受け取ります ([パブリック DNS] 列が表示されていない場合は、[表示/非表示] アイコンを選択してから、[パブリック DNS] を選択します)。

ステップ 2: EC2 インスタンスロールにアーティファクトバケットのアクセス許可を追加する

パイプラインのアーティファクトバケットへのアクセスを許可するには、インスタンス用に作成した EC2 インスタンスロールを更新する必要があります。

> **Note**
>
> インスタンスを作成する際は、既存の EC2 インスタンスロールを作成または使用します。`Access Denied` エラーを回避するには、インスタンスロールに S3 バケットアクセス許可を追加して、CodePipeline アーティファクトバケットにインスタンスアクセス許可を付与する必要があります。デフォルトのロールを作成するか、パイプラインのリージョンのアーティファクトバケットにスコープダウンされた `s3:GetObject` アクセス許可で既存のロールを更新します。

1. CodePipeline コンソールでパイプラインに移動します。[設定] を選択します。既存のパイプラインのアーティファクト保存場所の名前を確認します。アーティファクトバケットの Amazon リソースネーム (ARN) をメモしてコピーします。
2. IAM コンソールに移動し、[ロール] を選択します。このチュートリアルのステップ 1 で作成したインスタンスロールを選択します。
3. [Permissions] (アクセス許可) タブで [Add inline policy] (インラインポリシーの追加) を選択します。

4. 次の JSON をポリシードキュメントに追加し、Resource フィールドの値をバケット ARN に置き換えます。

```
{
    "Effect": "Allow",
    "Principal": "*",
    "Action": "s3:GetObject",
    "Resource": "arn:aws:s3:::BucketName"
}
```

5. [更新] を選択します。

ステップ 3: リポジトリにスクリプトファイルを追加する

このサンプルテキストを貼り付けて、デプロイのスクリプト後のステップ用の script.sh ファイルを作成します。

```
echo "Hello World!"
```

ソースリポジトリに **script.sh** ファイルを追加するには

1. テキストエディタを開き、上記のファイルをコピーして新しいファイルに貼り付けます。
2. ソースリポジトリに script.sh ファイルをコミットし、プッシュします。

   a. ファイルを追加します。

   ```
   git add .
   ```

   b. 変更をコミットします。

   ```
   git commit -m "Adding script.sh."
   ```

   c. コミットをプッシュします。

   ```
   git push
   ```

   リポジトリ内のパスを書き留めます。

```
/MyDemoRepo/test/script.sh
```

ステップ 4: パイプラインを作成する

CodePipeline ウィザードを使用してパイプラインステージを作成し、ソースリポジトリを接続します。

パイプラインを作成するには

1. CodePipeline コンソール (https://console.aws.amazon.com/codepipeline/) を開きます。
2. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。
3. [ステップ 1: 作成オプションを選択する] ページの [作成オプション] で、[カスタムパイプラインを構築する] オプションを選択します。[次へ] を選択します。
4. [ステップ 2: パイプラインの設定を選択する] で、[パイプライン名] に「**MyPipeline**」と入力します。
5. CodePipeline は、特徴と料金が異なる V1 タイプと V2 タイプのパイプラインを提供しています。V2 タイプは、コンソールで選択できる唯一のタイプです。詳細については、「パイプラインタイプ」を参照してください。CodePipeline の料金については、料金を参照してください。
6. [サービスロール] で、[既存サービスロールの使用] を選択し、このアクションに必要なアクセス許可で更新された CodePipeline サービスロールを選択します。このアクション用に CodePipeline サービスロールを設定するには、「EC2 デプロイアクションのサービスロールポリシーのアクセス許可」を参照してください。
7. [詳細設定] をデフォルト設定のままにし、[次へ] を選択します。
8. [ステップ 3: ソースステージの追加] ページで、ソースステージを追加します。

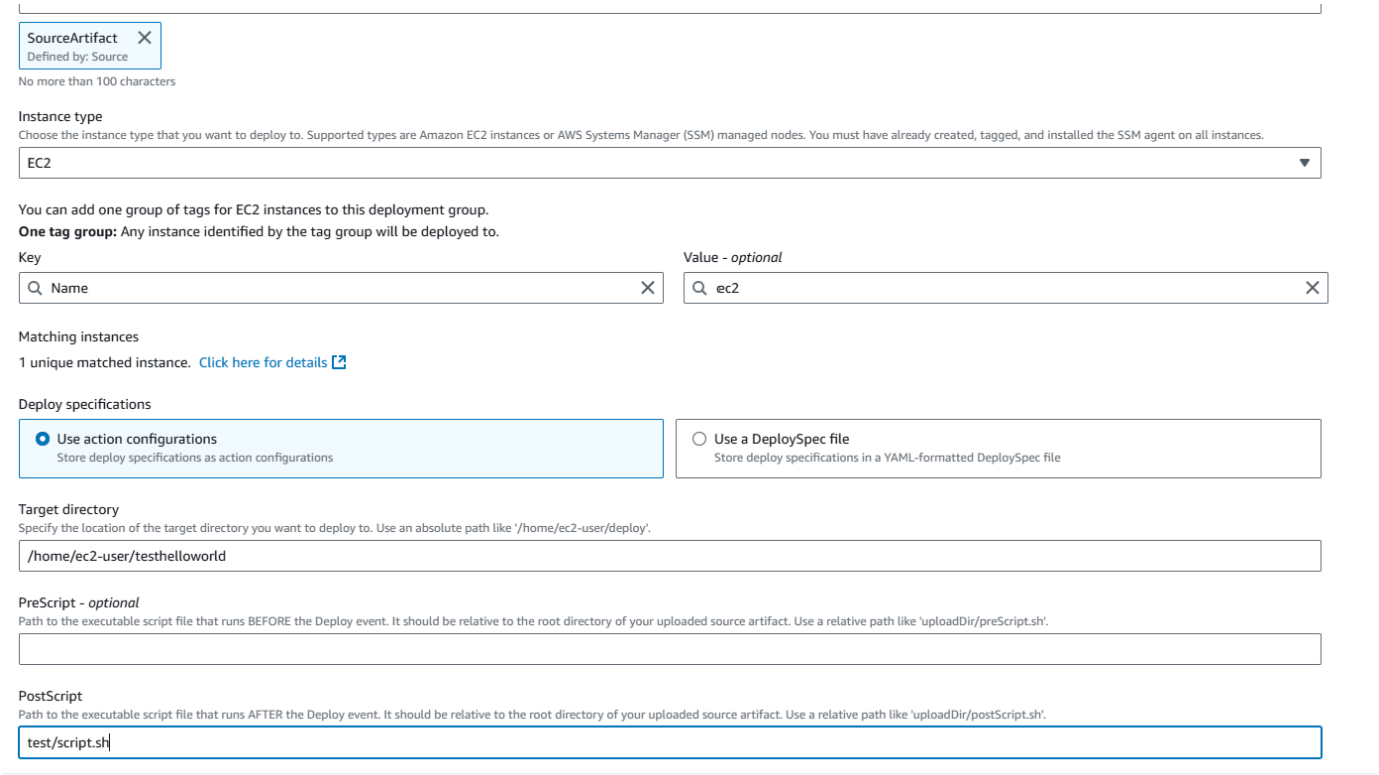
   a. [ソースプロバイダー] で、[GitHub (GitHub アプリ経由)] を選択します。

   b. 接続 で、既存の接続を選択するか、新規の接続を作成します。GitHub ソースアクション用の接続を作成または管理する方法については、GitHub コネクション を参照してください。

   c. リポジトリ名 で、GitHub リポジトリの名前を選択します。

   [次へ] を選択します。
9. [ステップ 4: ビルドステージの追加] ページで、[スキップ] を選択します。

10. [ステップ 5: デプロイステージの追加] ページで、[EC2] を選択します。



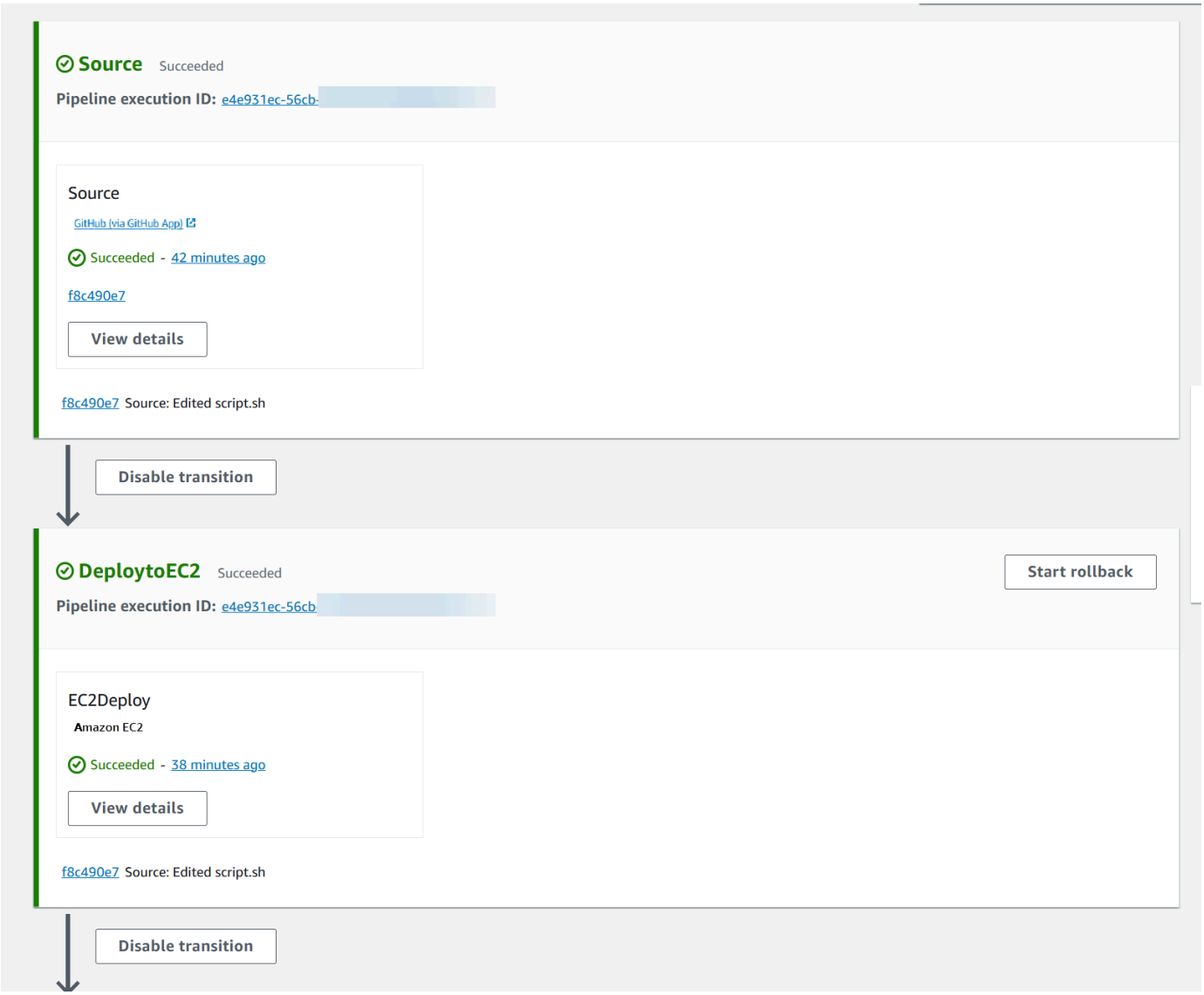
    a. [ターゲットディレクトリ] で、`/home/ec2-user/testhelloworld` などのデプロイ先のインスタンスのディレクトリを入力します。

> **Note**
>
> アクションがインスタンスで使用するデプロイディレクトリを指定します。アクションは、デプロイの一部としてインスタンスに指定されたディレクトリの作成を自動化します。

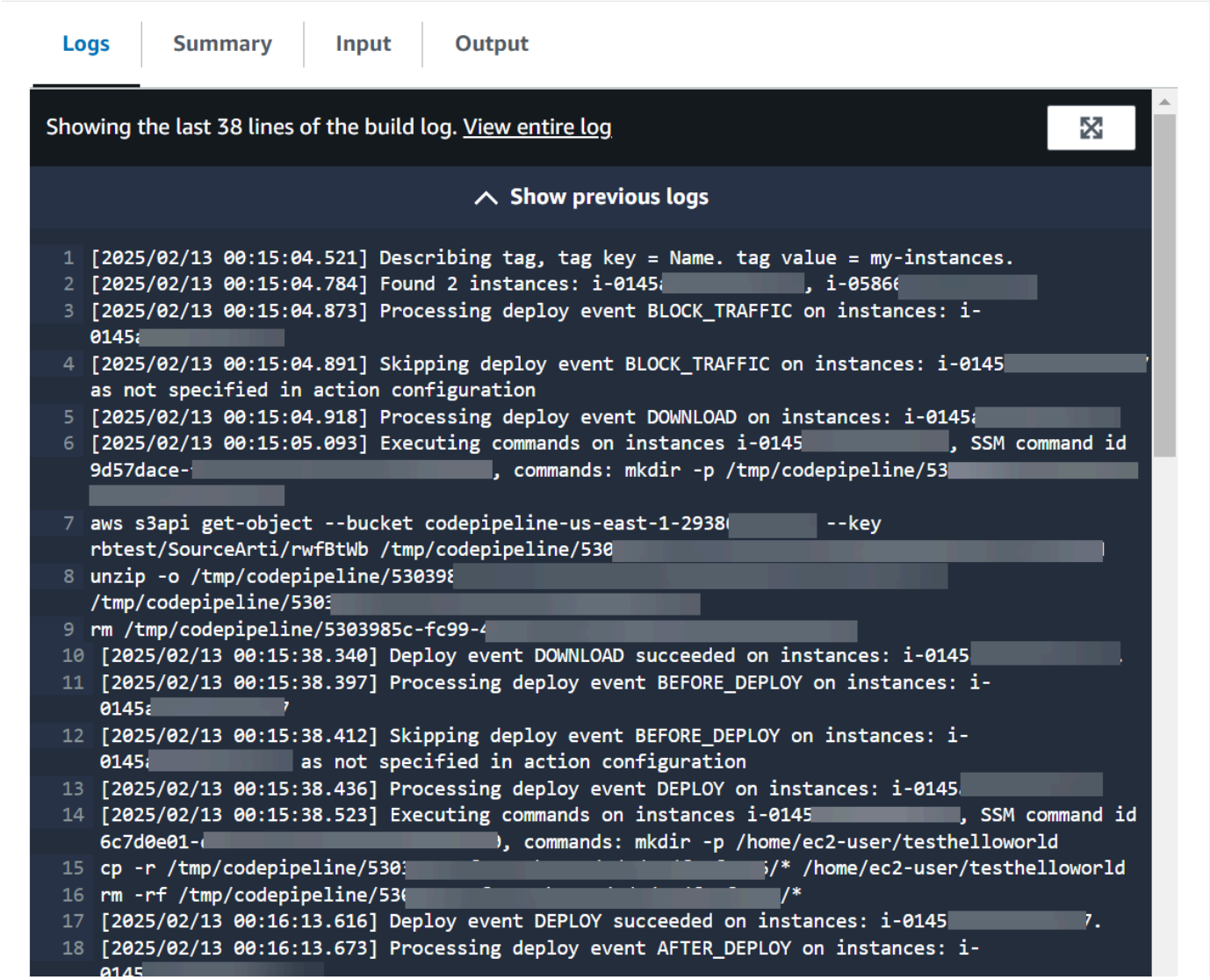
    b. [PostScript] で、`test/script.sh` などのスクリプトのパスとファイル名を入力します。

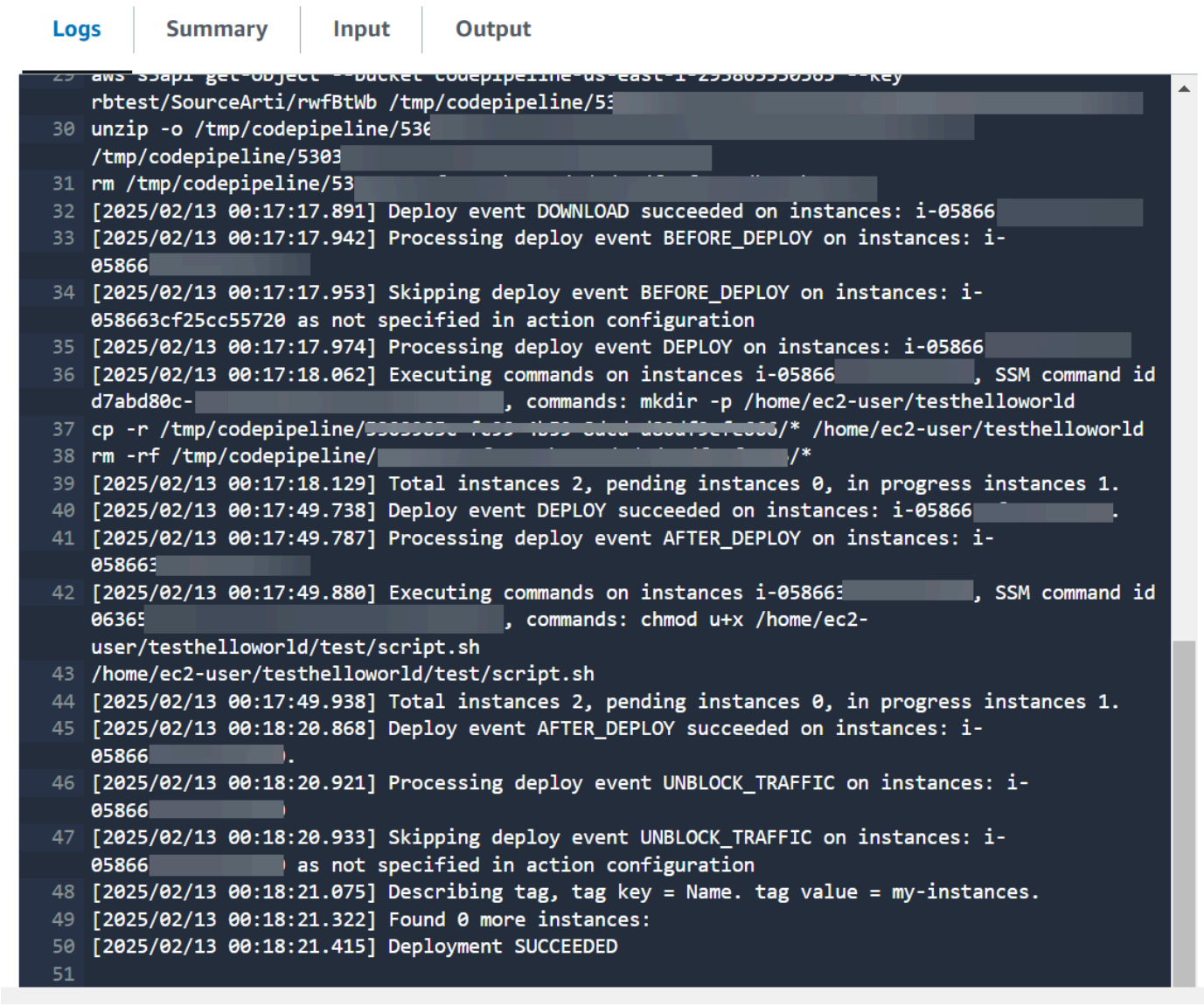
    c. [次へ] を選択します。

11. [Step 6: Review] ページで、パイプラインの設定を確認し、[Create pipeline] を選択してパイプラインを作成します。



12. パイプラインが正常に実行されたら、[詳細を表示] を選択してアクションのログを表示し、マネージドコンピューティングアクション出力を表示します。


Logs
Summary
Input
Output
Showing the last 38 lines of the build log. View entire log
Show previous logs
1 [2025/02/13 00:15:04.521] Describing tag, tag key = Name. tag value = my-instances.
2 [2025/02/13 00:15:04.784] Found 2 instances: i-0145 , i-0586
3 [2025/02/13 00:15:04.873] Processing deploy event BLOCK_TRAFFIC on instances: i-
0145
4 [2025/02/13 00:15:04.891] Skipping deploy event BLOCK_TRAFFIC on instances: i-0145
as not specified in action configuration
5 [2025/02/13 00:15:04.918] Processing deploy event DOWNLOAD on instances: i-0145
6 [2025/02/13 00:15:05.093] Executing commands on instances i-0145 , SSM command id
9d57dace- , commands: mkdir -p /tmp/codepipeline/53
7 aws s3api get-object --bucket codepipeline-us-east-1-2938 --key
rbtest/SourceArti/rwfBtWb /tmp/codepipeline/530
8 unzip -o /tmp/codepipeline/530398
/tmp/codepipeline/530
9 rm /tmp/codepipeline/5303985c-fc99-4
10 [2025/02/13 00:15:38.340] Deploy event DOWNLOAD succeeded on instances: i-0145
11 [2025/02/13 00:15:38.397] Processing deploy event BEFORE_DEPLOY on instances: i-
0145
12 [2025/02/13 00:15:38.412] Skipping deploy event BEFORE_DEPLOY on instances: i-
0145 as not specified in action configuration
13 [2025/02/13 00:15:38.436] Processing deploy event DEPLOY on instances: i-0145
14 [2025/02/13 00:15:38.523] Executing commands on instances i-0145 , SSM command id
6c7d0e01- , commands: mkdir -p /home/ec2-user/testhelloworld
15 cp -r /tmp/codepipeline/530 /* /home/ec2-user/testhelloworld
16 rm -rf /tmp/codepipeline/53 /*
17 [2025/02/13 00:16:13.616] Deploy event DEPLOY succeeded on instances: i-0145 .
18 [2025/02/13 00:16:13.673] Processing deploy event AFTER_DEPLOY on instances: i-
0145



ステップ 5: パイプラインをテストする

パイプラインには、end-to-endのネイティブ AWS 継続的デプロイを実行するためのすべてが必要です。次は、コードの変更をソースリポジトリにプッシュすることで機能をテストします。

パイプラインをテストするには

1. 設定済みソースリポジトリにコード変更を行い、変更をコミットしてプッシュします。
2. CodePipeline コンソール (https://console.aws.amazon.com/codepipeline/) を開きます。
3. リストからパイプラインを選択します。

4. ステージを通してパイプラインの進行状況を監視します。パイプラインが完了し、アクションによってスクリプトがインスタンスにデプロイされます。
5. トラブルシューティングの詳細については、「EC2 デプロイアクションがエラーメッセージ No such file で失敗する」を参照してください。

チュートリアル: CodePipeline (V2 タイプ) を使用して Docker イメージを構築し Amazon ECR にプッシュする

このチュートリアルは、ソースコードの変更後に Docker イメージを実行して Amazon ECR にプッシュする CodePipeline のビルドアクションの作成について説明します。このチュートリアルでは、プッシュされたイメージをデプロイする Amazon ECS デプロイアクションを追加する方法についても説明します。

> **Important**
>
> コンソールでのパイプライン作成の一環として、CodePipeline は S3 アーティファクトバケットをアーティファクトとして使用します (これは S3 ソースアクションで使用するバケットとは異なります)。S3 アーティファクトバケットがパイプラインのアカウントとは異なるアカウントにある場合は、S3 アーティファクトバケットが によって所有 AWS アカウントされており、安全で信頼できることを確認してください。

> **Note**
>
> このチュートリアルでは、GitHub ソースリポジトリと Amazon ECS クラスターにデプロイするための Amazon ECS 標準アクションを備えた CodePipeline パイプラインの ECRBuildAndPublish ビルドアクションについて説明します。Amazon ECS のソースとして ECR イメージリポジトリを含むパイプラインを使用する CodePipeline での CodeDeploy のブルー/グリーンデプロイアクションに関するチュートリアルについては、「チュートリアル: Amazon ECR ソース、ECS - CodeDeploy 間のデプロイでパイプラインを作成する」を参照してください。

> **Important**
>
> このアクションでは、CodePipeline マネージド CodeBuild コンピューティングを使用して、ビルド環境でコマンドを実行します。コマンドアクションを実行すると、 AWS CodeBuildで別途料金が発生します。

前提条件

このチュートリアルで CD パイプラインを作成する前に、いつくかのリソースを用意する必要があります。使用を開始するために必要なものは以下のとおりです。

> **Note**
>
> これらのリソースはすべて、同じ AWS リージョン内に作成する必要があります。

- ソースコントロールリポジトリ (このチュートリアルでは GitHub を使用します) 。このチュートリアルでは、以下を追加します。
  - ステップ 1 では、CodePipeline の ECRBuildAndPublish ビルドアクションの入力アーティファクトとしてソースリポジトリにサンプル Dockerfile を追加します。
  - ステップ 2 では、CodePipeline の Amazon ECS 標準デプロイアクションの要件として、ソースリポジトリにサンプル imagedefinitions.json ファイルを追加します。
- Dockerfile から構築したイメージを含む Amazon ECR イメージリポジトリ。詳細については、Amazon Elastic Container Registry ユーザーガイド の「リポジトリの作成」と「イメージをプッシュする」を参照してください。
- イメージリポジトリと同じリージョンで作成された Amazon ECS クラスターとサービス。詳細については、「Amazon Simple Queue Service デベロッパーガイド」の「クラスターの作成」と「サービスの作成」を参照してください。

これらの前提条件を満たした後、チュートリアルに進んで CD パイプラインを作成できます。

ステップ 1: ソースリポジトリに Dockerfile を追加する

このチュートリアルでは、ECRBuildAndPublish アクションを使用して Docker イメージを構築し、Amazon ECR にイメージをプッシュします。CodePipeline のマネージドコンピューティングアクションは、CodeBuild を使用して ECR ログインとイメージプッシュのコマンドを実行しま

す。buildspec.yml ファイルをソースコードリポジトリに追加して CodeBuild に処理方法を指示する必要はありません。この例では、次のようにリポジトリに Dockerfile のみを指定します。

このサンプルテキストを貼り付けて Dockerfile ファイルを作成します。このサンプル Dockerfile は、前提条件の ECR イメージの手順で使用されているサンプルと同じです。

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

# Install dependencies
RUN yum update -y && \
 yum install -y httpd

# Install apache and write hello world message
RUN echo 'Hello World!' > /var/www/html/index.html

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

EXPOSE 80

CMD /root/run_apache.sh
```

ソースリポジトリに **Dockerfile** ファイルを追加するには

1. テキストエディタを開き、上記の Dockerfile をコピーして新しいファイルに貼り付けます。
2. ソースリポジトリに Dockerfile ファイルをコミットし、プッシュします。

   a. ファイルを追加します。

   ```
   git add .
   ```

   b. 変更をコミットします。

   ```
   git commit -m "Adding Dockerfile."
   ```

   c. コミットをプッシュします。

   ```
   git push
   ```

ファイルは、必ずリポジトリのルートレベルに配置してください。

```
/ Dockerfile
```

ステップ 2: imagedefinitions.json ファイルをソースリポジトリに追加する

このチュートリアルでは、CodePipeline の Amazon ECS 標準デプロイアクションを使用して、コンテナを Amazon ECS クラスターにデプロイします。Amazon ECS 標準デプロイアクションには、イメージ名と URI を含む imagedefinitions.json ファイルが必要です。imagedefinitions.json ファイルの詳細については、「Amazon ECS 標準デプロイアクション用の imagedefinitions.json ファイル」を参照してください。

このサンプルテキストを貼り付けて `imagedefinitions.json` ファイルを作成します。`hello-world` などの Dockerfile の名前を使用し、イメージが保存されている Amazon ECR リポジトリの URI を使用します。

```
[
  {
    "name": "hello-world",
    "imageUri": "ACCOUNT-ID.dkr.ecr.us-east-1.amazonaws.com/actions/image-repo"
  }
]
```

ソースリポジトリに **`imagedefinitions.json`** ファイルを追加するには

1. テキストエディタを開き、上記の例をコピーして新しいファイルに貼り付けます。
2. ソースリポジトリに `imagedefinitions.json` ファイルをコミットし、プッシュします。

   a. ファイルを追加します。

   ```
   git add .
   ```

   b. 変更をコミットします。

   ```
   git commit -m "Adding imagedefinitions.json."
   ```

   c. コミットをプッシュします。

```
git push
```

ファイルは、必ずリポジトリのルートレベルに配置してください。

```
/ imagedefinitions.json
```

ステップ 3: パイプラインを作成する

CodePipeline ウィザードを使用してパイプラインステージを作成し、ソースリポジトリを接続します。

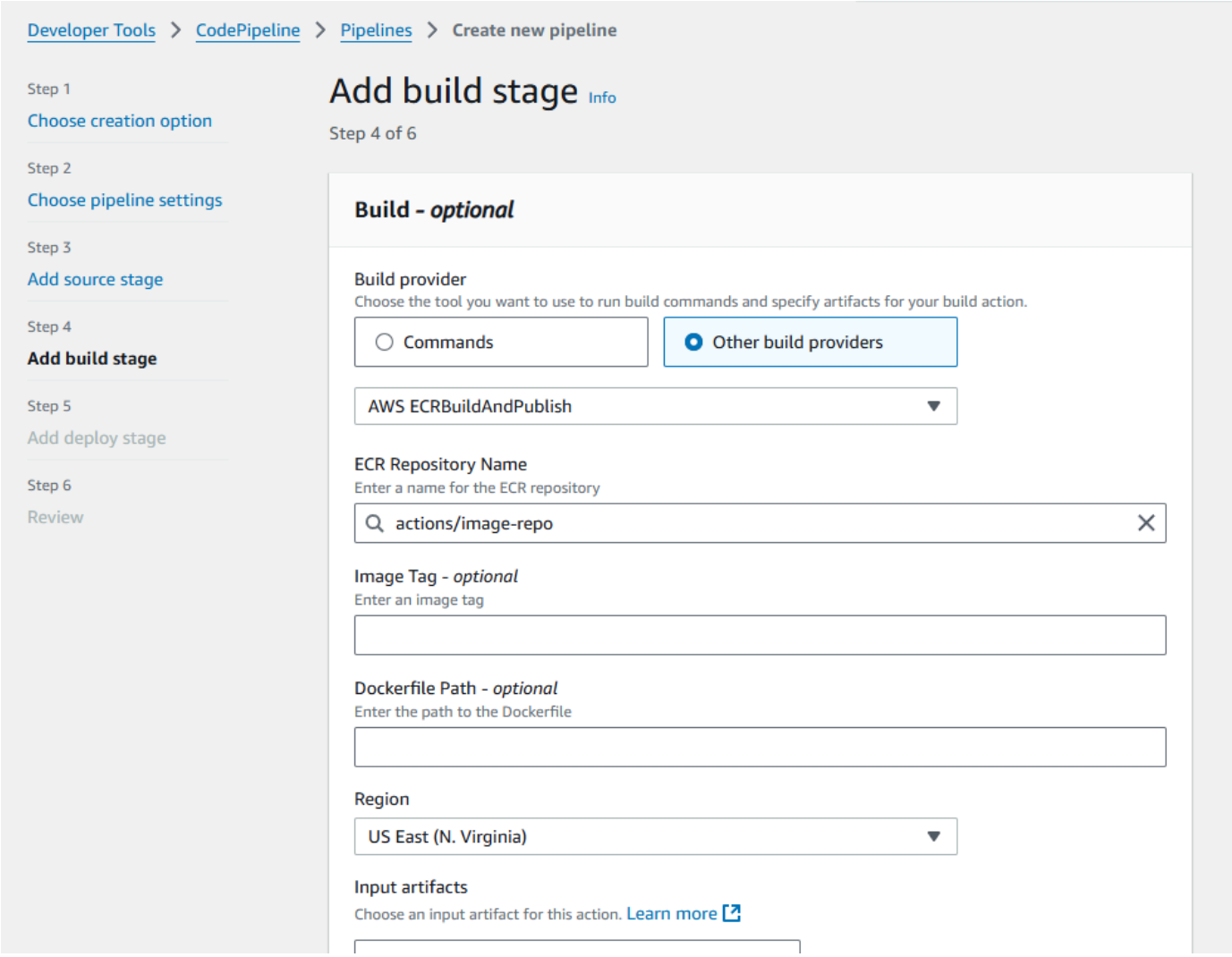
パイプラインを作成するには

1. CodePipeline コンソール (https://console.aws.amazon.com/codepipeline/) を開きます。
2. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。
3. [ステップ 1: 作成オプションを選択する] ページの [作成オプション] で、[カスタムパイプラインを構築する] オプションを選択します。[次へ] を選択します。
4. [ステップ 2: パイプラインの設定を選択する] で、[パイプライン名] に「**MyPipeline**」と入力します。
5. CodePipeline は、特徴と料金が異なる V1 タイプと V2 タイプのパイプラインを提供しています。V2 タイプは、コンソールで選択できる唯一のタイプです。詳細については、「パイプラインタイプ」を参照してください。CodePipeline の料金については、料金を参照してください。
6. サービスロール で、新しいサービスロール を選択して、CodePipeline が IAM でサービスロールを作成できるようにします。
7. [詳細設定] をデフォルト設定のままにし、[次へ] を選択します。
8. [ステップ 3: ソースステージの追加] ページで、ソースステージを追加します。
   a. [ソースプロバイダー] で、[GitHub (GitHub アプリ経由)] を選択します。
   b. 接続 で、既存の接続を選択するか、新規の接続を作成します。GitHub ソースアクション用の接続を作成または管理する方法については、GitHub コネクション を参照してください。
   c. リポジトリ名 で、GitHub リポジトリの名前を選択します。

d. [デフォルトブランチ] で、パイプラインを手動で開始する場合、または Git タグではないソースイベントで開始する場合に指定するブランチを選択します。変更元がトリガーでない場合やパイプラインの実行が手動で開始された場合は、デフォルトブランチの HEAD コミットが変更として使用されます。

[次へ] を選択します。

9. ステップ 4: ビルドステージの追加ページで、その他のビルドプロバイダーを選択し、ECRBuildAndPublish を選択します。



a. [リポジトリ名] でリポジトリ名を選択します。

b. [次へ] を選択します。

10. [ステップ 5: テストステージの追加] で、[テストステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

[次へ] を選択します。

11. [ステップ 6: デプロイステージの追加] ページで、[導入段階をスキップ] を選択します。以下の手順を使用して、ECS アクションを追加します。
12. ステップ 7: 確認ページで、パイプライン設定を確認し、パイプラインの作成を選択してパイプラインを作成します。
13. パイプラインを編集して、Amazon ECS デプロイアクションをパイプラインに追加します。

    a. 右上の [編集] を選択します。

    b. 図の最下部で [+ Add stage] (+ ステージの追加) を選択します。[ステージ名] に名前 (**`Deploy`** など) を入力します。

    c. [+ Add action group (+ アクションの追加)] を選択します。

    d. [アクション名] に名前を入力します。

    e. [アクションプロバイダ] で [Amazon ECS] を選択します。[リージョン] をデフォルトでパイプラインのリージョンにすることを許可します。

    f. [入力アーティファクト] で、`SourceArtifact` などのソースアクションからの入力アーティファクトを選択します。

    g. [Cluster name (クラスター名)] で、サービスが実行されている Amazon ECS クラスターを選択します。

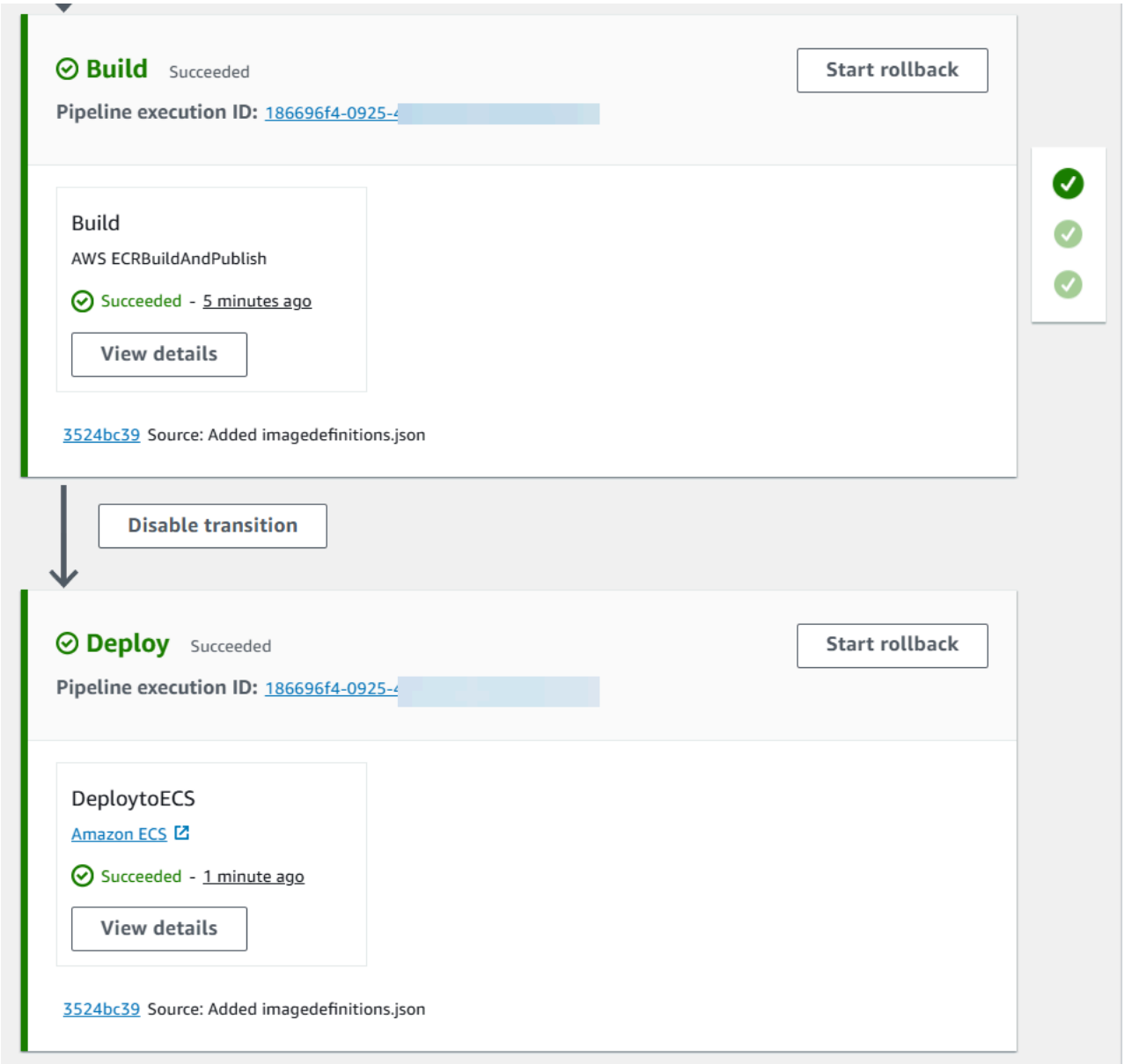
    h. [サービス名] で更新するサービスを選択します。

    i. [保存] を選択します。

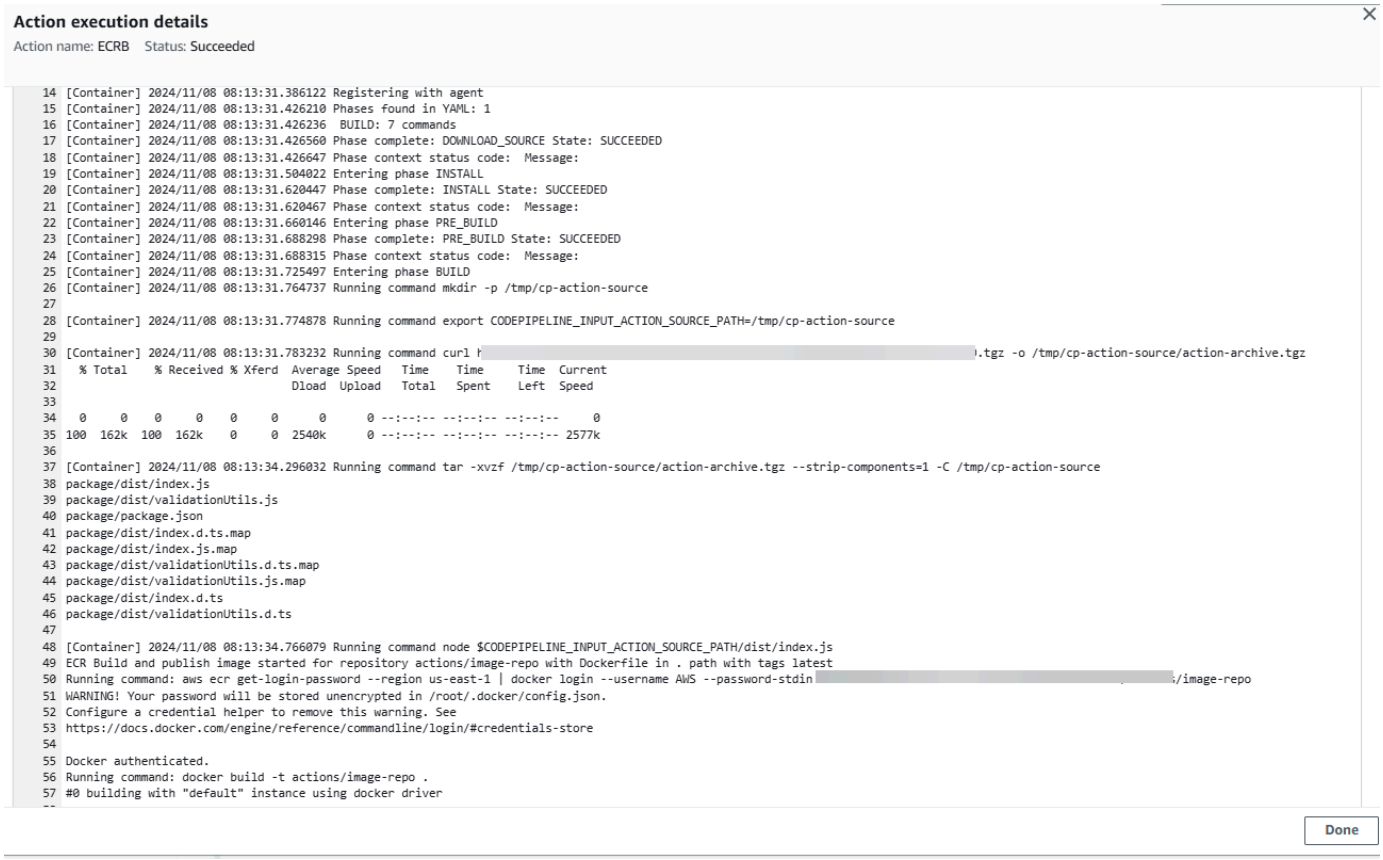
    j. 編集中のステージで、[完了] を選択します。 AWS CodePipeline のペインで [保存] を選択し、警告メッセージで [保存] を選択します。

    k. 変更を送信してパイプラインのビルドを開始するには、[変更をリリース]、[リリース] の順に選択します。

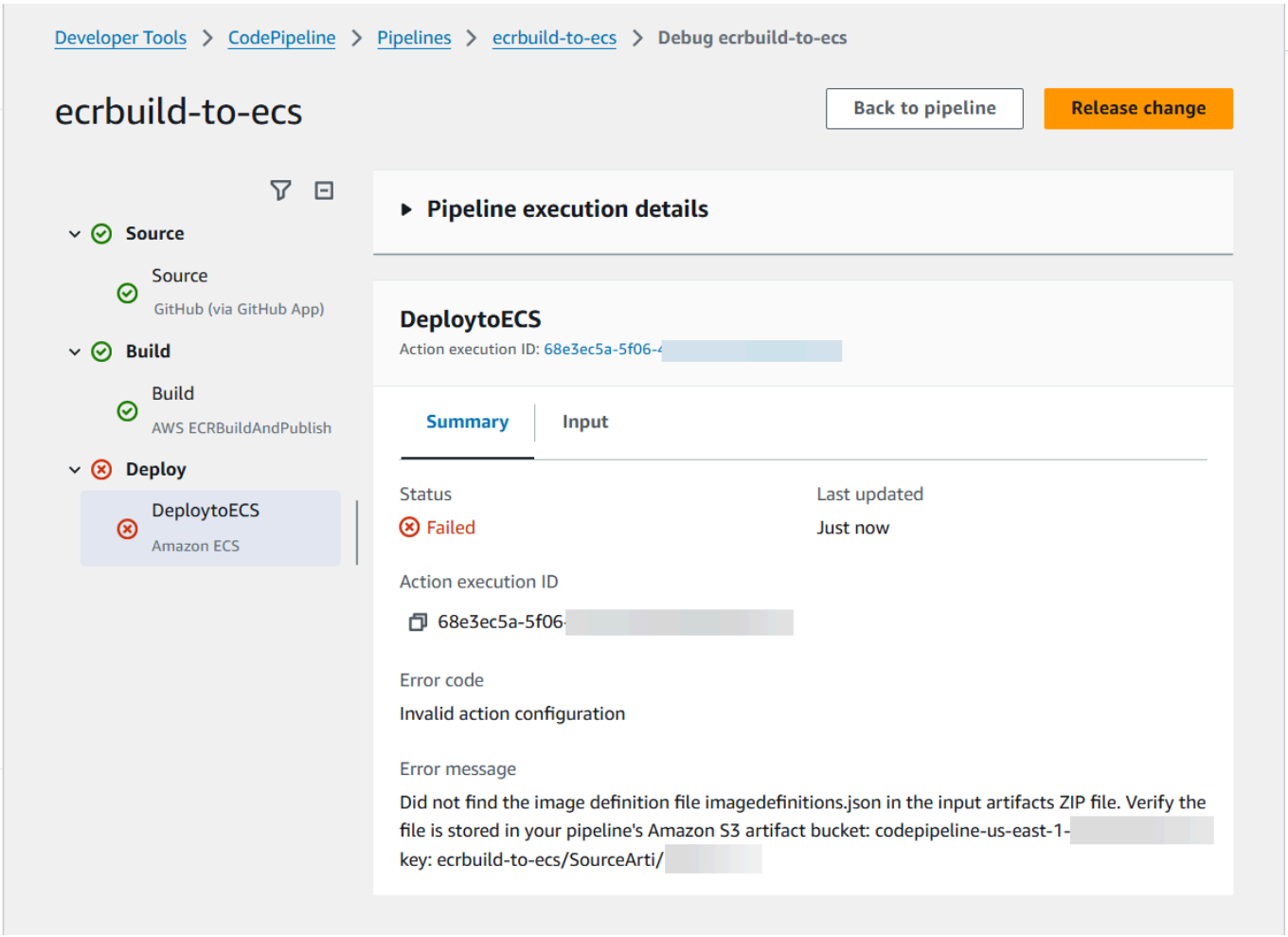
14. パイプラインが実行されたら、パイプラインの構造とステータスを確認します。



15. パイプラインが正常に実行されたら、[詳細を表示] を選択してアクションのログを表示し、マネージドコンピューティングアクション出力を表示します。



16. 失敗したアクションのトラブルシューティングを行います。例えば、imagedefinitions.json ファイルがソースリポジトリにない場合、ECS デプロイアクションは失敗する可能性があります。以下は、imagedefinitions.json ファイルがない場合に表示されるエラーメッセージの例です。



ステップ 4: パイプラインのテスト

パイプラインには、end-to-endのネイティブ AWS 継続的デプロイを実行するためのすべてが必要です。次は、コードの変更をソースリポジトリにプッシュすることで機能をテストします。

パイプラインをテストするには

1. 設定済みソースリポジトリにコード変更を行い、変更をコミットしてプッシュします。
2. CodePipeline コンソール (https://console.aws.amazon.com/codepipeline/) を開きます。
3. リストからパイプラインを選択します。
4. ステージを通してパイプラインの進行状況を監視します。パイプラインが終了し、アクションが Docker イメージをコードの変更で作成された ECR にプッシュすることを確認します。

チュートリアル: CodePipeline を使用して Amazon EKS にデプロイする

このチュートリアルは、Amazon EKS で設定したクラスターにコードをデプロイするデプロイアクションを CodePipeline で作成する方法について説明します。

EKS アクションは、パブリック EKS クラスターとプライベート EKS クラスターの両方をサポートします。プライベートクラスターは EKS で推奨されるタイプですが、どちらのタイプもサポートされています。

> **Note**
>
> コンソールでのパイプライン作成の一環として、CodePipeline は S3 アーティファクトバケットをアーティファクトとして使用します (これは S3 ソースアクションで使用するバケットとは異なります)。S3 アーティファクトバケットがパイプラインのアカウントとは異なるアカウントにある場合は、S3 アーティファクトバケットが によって所有 AWS アカウントされており、安全で信頼できることを確認してください。

> **Note**
>
> このアクションでは、CodePipeline マネージド CodeBuild コンピューティングを使用して、ビルド環境でコマンドを実行します。コマンドアクションを実行すると、 AWS CodeBuildで別途料金が発生します。

> **Note**
>
> EKS デプロイアクションは V2 タイプのパイプラインでのみ使用できます。

前提条件

このチュートリアルで CD パイプラインを作成する前に、いつくかのリソースを用意する必要があります。使用を開始するために必要なものは以下のとおりです。

> **Note**
>
> これらのリソースはすべて、同じ AWS リージョン内に作成する必要があります。

- サンプル `deployment.yaml` ファイルを追加するソースコントロールリポジトリ (このチュートリアルでは GitHub を使用します) 。
- 以下の ステップ 3: IAM の CodePipeline サービスロールポリシーを更新する を使用してこのアクションのアクセス許可で更新された、既存の CodePipeline サービスロールを使用する必要があります。必要なアクセス許可は、作成するクラスターのタイプに基づきます。詳細については、「サービスロールのポリシーのアクセス許可」を参照してください。
- ECR またはイメージリポジトリにプッシュした作業イメージとリポジトリタグ。

これらの前提条件を満たした後、チュートリアルに進んで CD パイプラインを作成できます。

ステップ 1: (オプション) Amazon EKS でクラスターを作成する

パブリックエンドポイントまたはプライベートエンドポイントを使用して EKS クラスターを作成できます。

次の手順では、EKS でパブリッククラスターまたはプライベートクラスターを作成します。クラスターをすでに作成している場合、このステップはオプションです。

Amazon EKS にクラスターを作成する

このステップでは、EKS にクラスターを作成します。

パブリッククラスターを作成する

1. EKS コンソールを開いて [クラスターの作成] を選択します。
2. [名前] で、クラスターに名前を付けます。[次へ] を選択します。
3. [作成] を選択します。

Amazon EKS でプライベートクラスターを作成する

プライベートエンドポイントを使用してクラスターを作成する場合は、プライベートサブネットのみをアタッチし、それらにインターネット接続があることを確認してください。

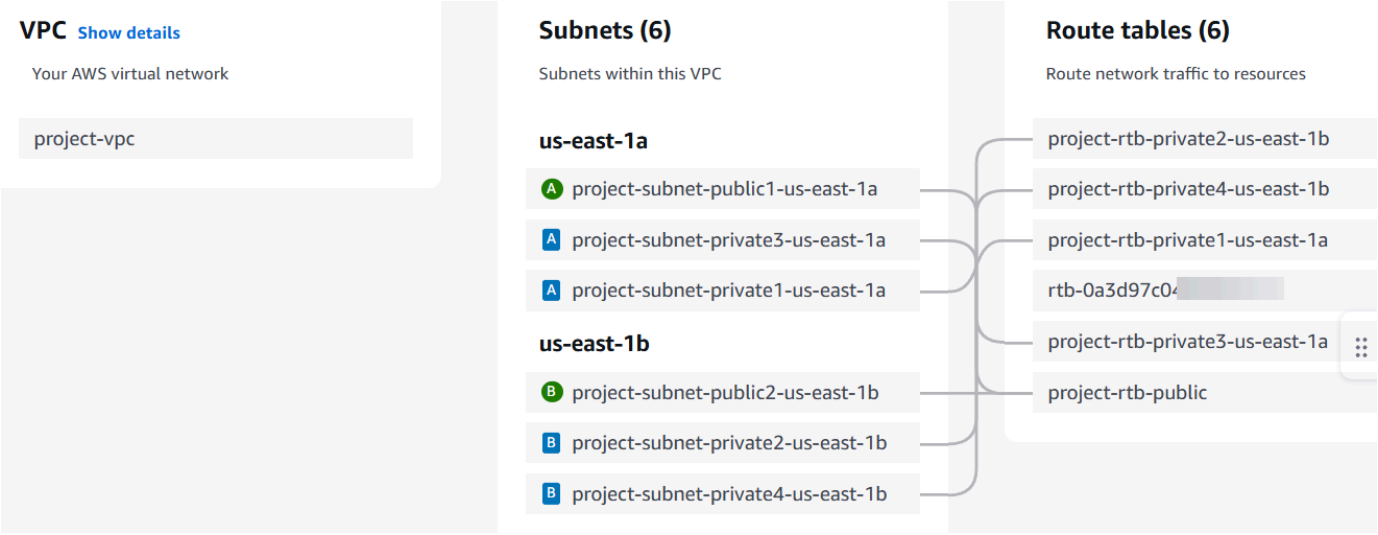
プライベートエンドポイントを使用してクラスターを作成するには、次の 5 つのサブステップに従います。

コンソールで VPC を作成します。

1. VPC コンソールを開いて [VPC の作成] を選択します。
2. [VPC 設定] で、[VPC など] を選択します。
3. 1 つのパブリックサブネットと 4 つのプライベートサブネットを作成することを選択します。[Create VPC (VPC の作成)] を選択します。
4. サブネットページで、[プライベート] を選択します。

VPC 内のプライベートサブネットを確認する

1. VPC に移動し、VPC ID を選択して VPC の詳細ページを開きます。
2. VPC の詳細ページで、[リソースマップ] タブを選択します。
3. 図を確認して、プライベートサブネットを書き留めます。サブネットにはパブリックまたはプライベートのステータスを示すラベルが表示され、各サブネットはルートテーブルにマッピングされます。



プライベートクラスターにはすべてのプライベートサブネットがあることに注意してください。

4. NAT ゲートウェイをホストするパブリックサブネットを作成します。一度に VPC にアタッチできるインターネットゲートウェイは 1 つだけです。

パブリックサブネット内に NAT ゲートウェイを作成する

1. パブリックサブネットで、NAT ゲートウェイを作成します。VPC コンソールに移動し、[インターネットゲートウェイ]を選択します。[インターネットゲートウェイの作成] を選択します。
2. [名前] にインターネットゲートウェイの名前を入力します。[インターネットゲートウェイの作成] を選択します。

プライベートサブネットのルートテーブルを更新して、トラフィックを NAT ゲートウェイに転送します。

プライベートサブネットのルートテーブルに NAT ゲートウェイを追加する

1. VPC コンソールに移動し、[サブネット] を選択します。
2. プライベートサブネットごとにそれを選択し、詳細ページでそのサブネットのルートテーブルを選択します。ルートテーブルの編集 を選択します。
3. プライベートサブネットのルートテーブルを更新して、インターネットトラフィックを NAT ゲートウェイに転送します。[Add Rule] (ルートの追加) を選択します。追加するオプションから [NAT ゲートウェイ] を選択します。作成したインターネットゲートウェイを選択します。
4. パブリックサブネットのカスタムルートテーブルを作成します。パブリックサブネットのネットワークアクセスコントロールリスト (ACL) で、プライベートサブネットからのインバウンドトラフィックが許可されていることを確認します。
5. [Save changes] (変更の保存) をクリックします。

このステップでは、EKS にクラスターを作成します。

プライベートクラスターを作成する

1. EKS コンソールを開いて [クラスターの作成] を選択します。
2. [名前] で、クラスターに名前を付けます。[次へ] を選択します。
3. VPC およびその他の設定情報を指定します。[作成] を選択します。

EKS クラスターは、パブリッククラスターでもプライベートクラスターでもかまいません。このステップは、プライベートエンドポイントのみを持つクラスター用です。クラスターがプライベートであることを確認してください。

ステップ 2: Amazon EKS でプライベートクラスターを設定する

このステップは、プライベートクラスターを作成している場合にのみ適用されます。このステップは、プライベートエンドポイントのみを持つクラスター用です。

クラスターを設定する

1. [ネットワーク] タブの EKS クラスターにのみプライベートサブネットをアタッチします。ステップ 1: (オプション) Amazon EKS でクラスターを作成する の「VPC 内のプライベートサブネットを確認する」セクションでキャプチャされたプライベートサブネットをアタッチします。
2. CodePipeline はパイプラインの S3 アーティファクトバケットからアーティファクトを保存および取得するため、プライベートサブネットがインターネットにアクセスできることを確認します。

ステップ 3: IAM の CodePipeline サービスロールポリシーを更新する

このステップでは、**cp-service-role** などの既存の CodePipeline サービスロールを CodePipeline がクラスターに接続するために必要なアクセス許可で更新します。既存のロールがない場合は、新しいロールを作成します。

CodePipeline サービスロールを次の手順で更新します。

CodePipeline サービスロールポリシーを更新するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. コンソールダッシュボードで [ロール] を選択します。
3. **cp-service-role** などの CodePipeline サービスロールを検索します。
4. 新しいインラインポリシーを追加します。
5. [ポリシーエディター] に、次のポリシーを入力します。

   - パブリッククラスターの場合は、次のアクセス許可を追加します。

     JSON

```
{
    "Version":"2012-10-17",

    "Statement": [
        {
```

```
            "Sid": "EksClusterPolicy",
            "Effect": "Allow",
            "Action": "eks:DescribeCluster",
            "Resource": "arn:aws:eks:us-east-1:111122223333:cluster/my-
cluster"
        },
        {
            "Sid": "EksVpcClusterPolicy",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeDhcpOptions",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeRouteTables",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeVpcs"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

- プライベートクラスターの場合は、次のアクセス許可を追加します。プライベートクラスターには、該当する場合、VPC に対する追加のアクセス許可が必要です。

  JSON

```
{
    "Version":"2012-10-17",

    "Statement": [
        {
            "Sid": "EksClusterPolicy",
            "Effect": "Allow",
            "Action": "eks:DescribeCluster",
            "Resource": "arn:aws:eks:us-east-1:111122223333:cluster/my-
cluster"
        },
        {
            "Sid": "EksVpcClusterPolicy",
            "Effect": "Allow",
            "Action": [
```

```
                "ec2:DescribeDhcpOptions",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeRouteTables",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeVpcs"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "ec2:CreateNetworkInterface",
            "Resource": "*",
            "Condition": {
                "StringEqualsIfExists": {
                    "ec2:Subnet": [
                        "arn:aws:ec2:us-east-1:ACCOUNT-ID:subnet/
subnet-03ebd65daeEXAMPLE",
                        "arn:aws:ec2:us-east-1:ACCOUNT-ID:subnet/
subnet-0e377f6036EXAMPLE",
                        "arn:aws:ec2:us-east-1:ACCOUNT-ID:subnet/
subnet-0db658ba1cEXAMPLE",
                        "arn:aws:ec2:us-east-1:ACCOUNT-ID:subnet/
subnet-0db658ba1cEXAMPLE"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "ec2:CreateNetworkInterfacePermission",
            "Resource": "*",
            "Condition": {
                "ArnEquals": {
                    "ec2:Subnet": [
                        "arn:aws:ec2:us-east-1:111122223333:subnet/
subnet-03ebd65daeEXAMPLE",
                        "arn:aws:ec2:us-east-1:111122223333:subnet/
subnet-0e377f6036EXAMPLE",
                        "arn:aws:ec2:us-east-1:111122223333:subnet/
subnet-0db658ba1cEXAMPLE",
```

```
                        "arn:aws:ec2:us-east-1:111122223333:subnet/
subnet-0db658ba1cEXAMPLE"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "ec2:DeleteNetworkInterface",
            "Resource": "*",
            "Condition": {
                "StringEqualsIfExists": {
                    "ec2:Subnet": [
                        "arn:aws:ec2:us-east-1:ACCOUNT-ID:subnet/
subnet-03ebd65daeEXAMPLE",
                        "arn:aws:ec2:us-east-1:ACCOUNT-ID:subnet/
subnet-0e377f6036EXAMPLE",
                        "arn:aws:ec2:us-east-1:ACCOUNT-ID:subnet/
subnet-0db658ba1cEXAMPLE",
                        "arn:aws:ec2:us-east-1:ACCOUNT-ID:subnet/
subnet-0db658ba1cEXAMPLE"
                    ]
                }
            }
        }
    ]
}
```

6. [ポリシーの更新] を選択してください。

ステップ 4: CodePipeline サービスロールのアクセスエントリを作成する

このステップでは、ステップ 3 で更新した CodePipeline サービスロールとマネージドアクセスポリシーを追加するアクセスエントリをクラスターに作成します。

1. EKS コンソールを開いてクラスターに移動します。
2. アクセス タブを選択します。
3. [IAM アクセスエントリ] で、[アクセスエントリの作成] を選択します。
4. [IAM プリンシパル ARN] で、**cp-service-role** などのアクション用に更新したロールを入力します。[次へ] を選択します。

5. [ステップ 2: アクセスポリシーの追加] ページの [ポリシー名] で、AmazonEKSClusterAdminPolicy などのアクセス用のマネージドポリシーを選択します。[Add policy] (ポリシーの追加) を選択します。[次へ] を選択します。

> **Note**
>
> これは、CodePipeline アクションが Kubernetes と通信するために使用するポリシーです。ベストプラクティスとして、マネージドポリシーではなく最小特権でポリシーのアクセス許可を絞り込むには、代わりにカスタムポリシーをアタッチします。

6. 確認ページで、[作成] を選択します。

ステップ 5: ソースリポジトリを作成し、Helm chart 設定ファイルを追加する

このステップでは、アクションに適した設定ファイル (Kubernetes マニフェストファイルまたは Helm チャート) を作成し、その設定ファイルをソースリポジトリに保存します。設定に適したファイルを使用します。詳細については、https://kubernetes.io/docs/reference/kubectl/quick-reference/ または https://helm.sh/docs/topics/charts/ を参照してください。

- Kubernetes の場合は、マニフェストファイルを使用します。
- Helm の場合は、Helm チャートを使用します。

1. GitHub リポジトリを作成するか、既存のものを使用します。
2. 以下の例に示すように、Helm チャートファイルのリポジトリに新しい構造を作成します。

```
mychart
|-- Chart.yaml
|-- charts
|-- templates
|   |-- NOTES.txt
|   |-- _helpers.tpl
|   |-- deployment.yaml
|   |-- ingress.yaml
|   `-- service.yaml
`-- values.yaml
```

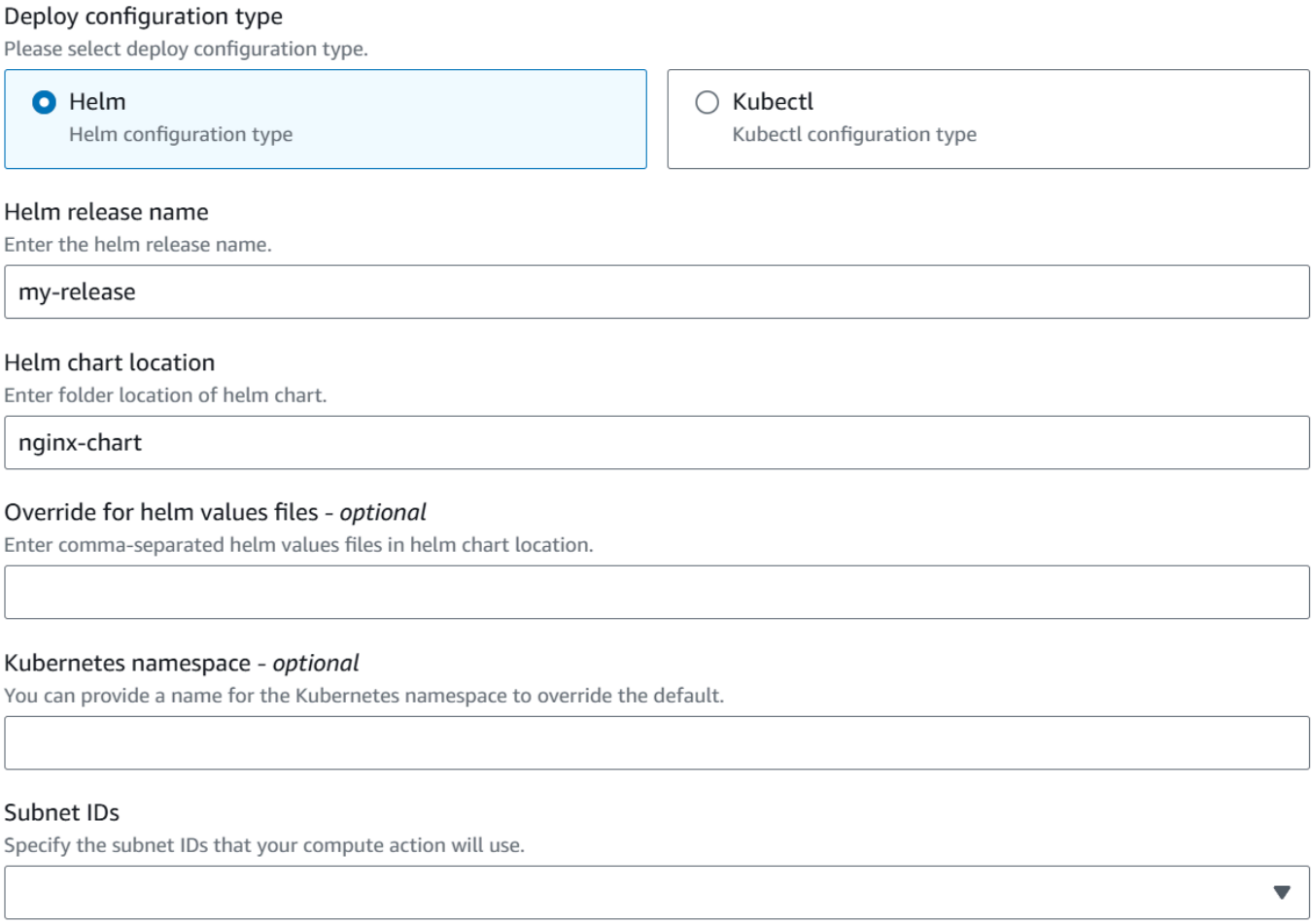
3. ファイルをリポジトリのルートレベルに追加します。

ステップ 6: パイプラインを作成する

CodePipeline ウィザードを使用してパイプラインステージを作成し、ソースリポジトリを接続します。

パイプラインを作成するには

1. CodePipeline コンソール (https://console.aws.amazon.com/codepipeline/) を開きます。
2. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。
3. [ステップ 1: 作成オプションを選択する] ページの [作成オプション] で、[カスタムパイプラインを構築する] オプションを選択します。[次へ] を選択します。
4. [ステップ 2: パイプラインの設定を選択する] で、[パイプライン名] に「**MyEKSPipeline**」と入力します。
5. CodePipeline は、特徴と料金が異なる V1 タイプと V2 タイプのパイプラインを提供しています。V2 タイプは、コンソールで選択できる唯一のタイプです。詳細については、「パイプラインタイプ」を参照してください。CodePipeline の料金については、料金を参照してください。
6. [サービスロール] で、ステップ 3 で更新したサービスロールを選択します。
7. [詳細設定] をデフォルト設定のままにし、[次へ] を選択します。
8. [ステップ 3: ソースステージの追加] ページの [ソースプロバイダー] で、[GitHub リポジトリへの接続を作成すること] を選択します。
9. [ステップ 4: ビルドステージの追加] ページで、[スキップ] を選択します。
10. [ステップ 5: デプロイステージの追加] ページで、[Amazon EKS] を選択します。

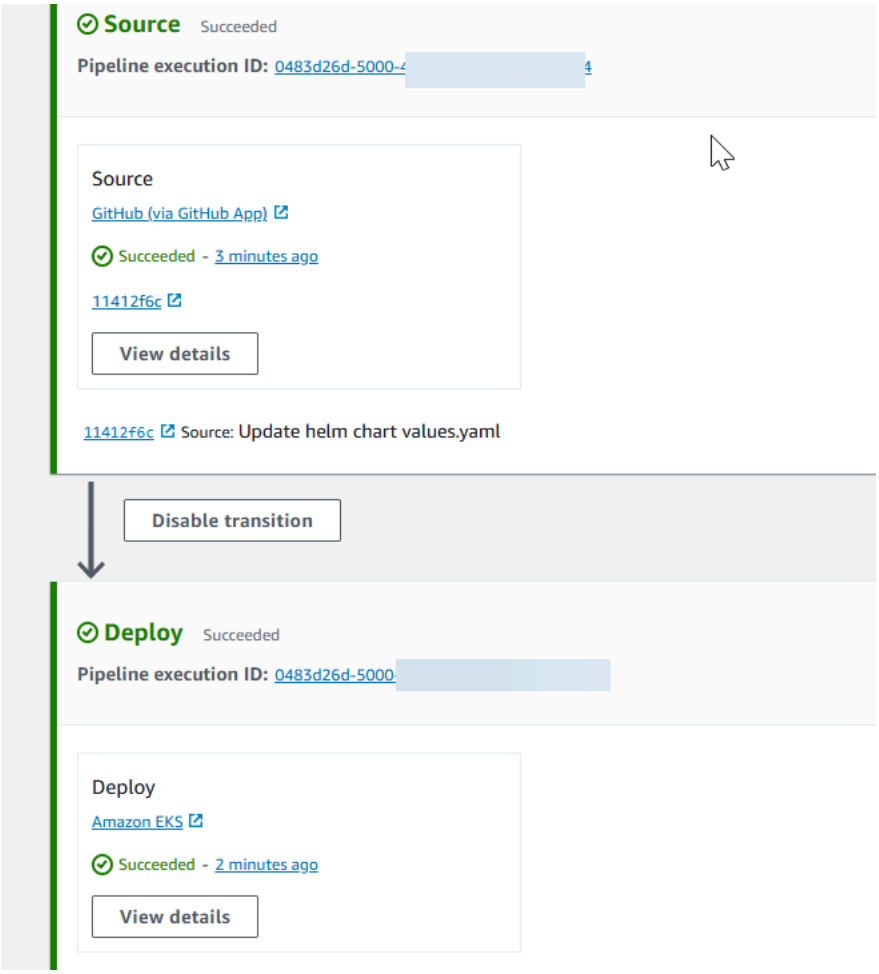


a. [デプロイ設定タイプ] で、[Helm] を選択します。

b. [Helm チャートの場所] で、`my-release` などのリリース名を入力します。Helm チャートの場所には、 などの Helm チャートファイルのパスを入力します`mychart`。

c. [次へ] を選択します。

11. [Step 6: Review] ページで、パイプラインの設定を確認し、[Create pipeline] を選択してパイプラインを作成します。



12. パイプラインが正常に実行されたら、[詳細を表示] を選択してアクションのログを表示し、アクション出力を表示します。

チュートリアル: コンピューティングを使用してコマンドを実行するパイプラインを作成する (V2 タイプ)

このチュートリアルでは、ビルドステージでコマンドアクションを使用して、指定したビルドコマンドを継続的に実行するパイプラインを設定します。コマンドアクションの詳細については、「コマンドアクションリファレンス」を参照してください。

> **Important**
>
> パイプライン作成の一環として、CodePipeline は、ユーザーが指定した S3 アーティファクトバケットをアーティファクトとして使用します (これは S3 ソースアクションで使用するバケットとは異なります)。S3 アーティファクトバケットがパイプラインのアカウントとは異

なるアカウントにある場合は、S3 アーティファクトバケットが によって所有 AWS アカウント されており、安全で信頼できることを確認してください。

前提条件

以下のものを用意しておく必要があります。

- GitHub リポジトリ。「チュートリアル: CodeCommit パイプラインソースで完全なクローンを使用する」で作成した GitHub リポジトリを使用できます。

ステップ 1: ソースファイルを作成して GitHub リポジトリにプッシュする

このセクションでは、サンプルのソースファイルを作成して、パイプラインがソースステージで使用するリポジトリにプッシュします。この例では、以下を作成してプッシュします。

- `README.txt` ファイル。

ソースファイルを作成するには

1. 次のテキストを含むファイルを作成します。

```
Sample readme file
```

2. `README.txt` という名前でファイルを保存します。

ファイルを GitHub リポジトリにプッシュするには

1. ファイルを リポジトリにプッシュまたはアップロードします。これらのファイルは、 AWS CodePipelineでのデプロイアクションのために パイプライン作成 ウィザードによって作成されたソースアーティファクトです。ファイルは、ローカルディレクトリに次のように表示されます。

```
README.txt
```

2. ローカルコンピュータで複製されたリポジトリから Git コマンドラインを使用するには:

   a. 以下のコマンドを実行して、すべてのファイルを一度にステージングします。

```
git add -A
```

b. 以下のコマンドを実行して、コミットメッセージによりファイルをコミットします。

```
git commit -m "Added source files"
```

c. 以下のコマンドを実行して、ローカルリポジトリから リポジトリにファイルをプッシュします。

```
git push
```

ステップ 2: パイプラインを作成する

このセクションでは、次のアクションを使用してパイプラインを作成します。

- ソースファイルが保存されているリポジトリの GitHub (GitHub アプリ経由) アクションを含むソースステージ。
- コマンドアクションを含むビルドステージ。

ウィザードを使用してパイプラインを作成するには

1. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。
3. [ステップ 1: 作成オプションを選択する] ページの [作成オプション] で、[カスタムパイプラインを構築する] オプションを選択します。[次へ] を選択します。
4. [ステップ 2: パイプラインの設定を選択する] で、[パイプライン名] に「**MyCommandsPipeline**」と入力します。
5. CodePipeline は、特徴と料金が異なる V1 タイプと V2 タイプのパイプラインを提供しています。V2 タイプは、コンソールで選択できる唯一のタイプです。詳細については、「パイプラインタイプ」を参照してください。CodePipeline の料金については、料金を参照してください。
6. [サービスロール] で、[新しいサービスロール] を選択し、CodePipeline に IAM でのサービスロールの作成を許可します。

> **Note**
>
> 既存のサービスロールを使用している場合、コマンドアクションを使用するには、サービスロールに以下のアクセス許可を追加する必要があります。サービスロールポリシーステートメントでリソースベースのアクセス許可を使用して、アクセス許可の範囲をパイプラインリソースレベルに絞り込みます。詳細については、「サービスロールのポリシーのアクセス許可」のポリシー例を参照してください。
>
> - logs:CreateLogGroup
> - logs:CreateLogStream
> - logs:PutLogEvents

7. [詳細設定] をデフォルト設定のままにし、[次へ] を選択します。
8. [ステップ 3: ソースステージの追加] ページで、ソースステージを追加します。

   a. [ソースプロバイダー] で、[GitHub (GitHub アプリ経由)] を選択します。

   b. 接続 で、既存の接続を選択するか、新規の接続を作成します。GitHub ソースアクション用の接続を作成または管理するには、「GitHub コネクション」を参照してください。

   c. [リポジトリ名] で、GitHub.com リポジトリの名前を選択します。

   d. [デフォルトブランチ] で、パイプラインを手動で開始する場合、または Git タグではないソースイベントで開始する場合に指定するブランチを選択します。変更元がトリガーでない場合やパイプラインの実行が手動で開始された場合は、デフォルトブランチの HEAD コミットが変更として使用されます。必要に応じて、フィルタリング (トリガー) を使用してウェブフックを指定することもできます。詳細については、「トリガーとフィルタリングを使用してパイプラインを自動的に開始する」を参照してください。

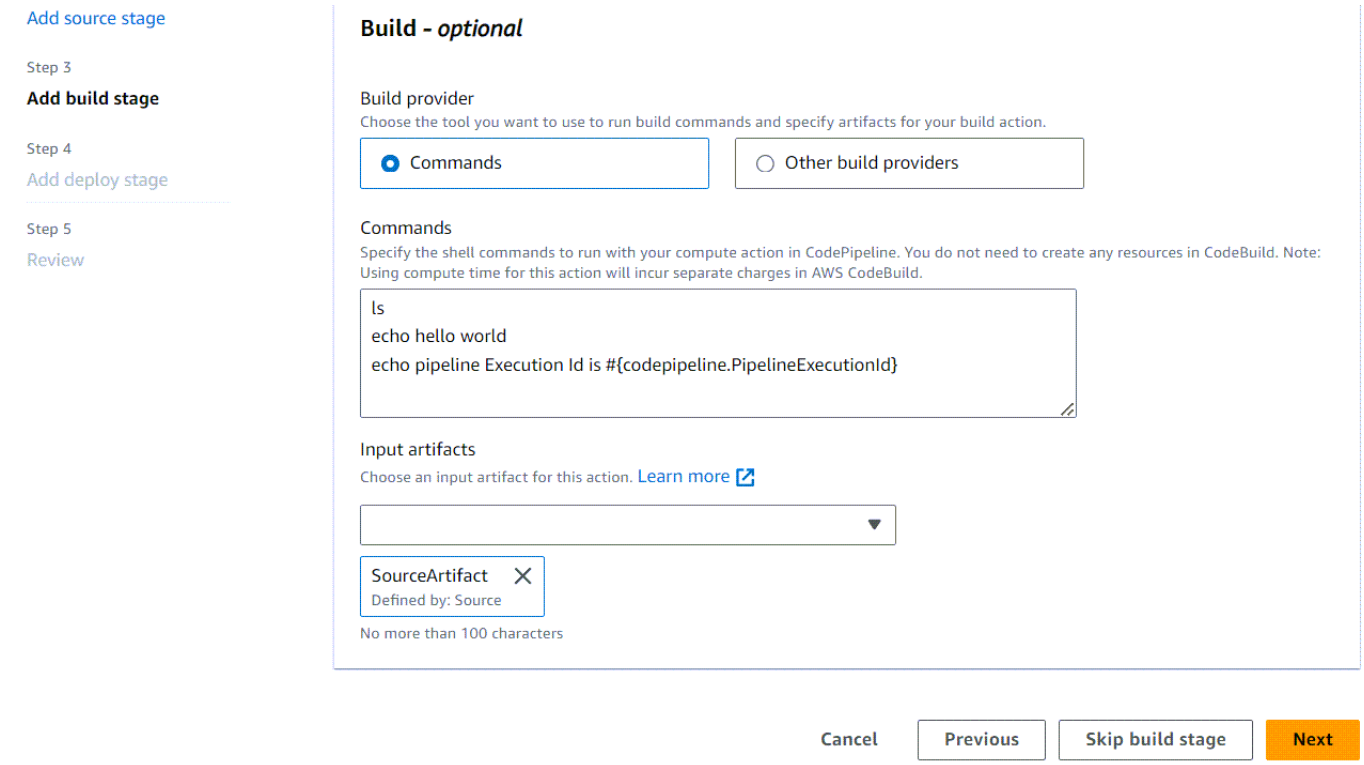
   [Next] (次へ) を選択します。
9. [ステップ 4: ビルドステージ を追加する] で、[コマンド] を選択します。

> **Note**
>
> コマンドアクションを実行すると、 AWS CodeBuildで別途料金が発生します。

以下のコマンドを入力します。

```
ls
echo hello world
cat README.txt
echo pipeline Execution Id is #{codepipeline.PipelineExecutionId}
```

[次へ] を選択します。



10. [ステップ 5: テストステージの追加] で、[テストステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

    [次へ] を選択します。

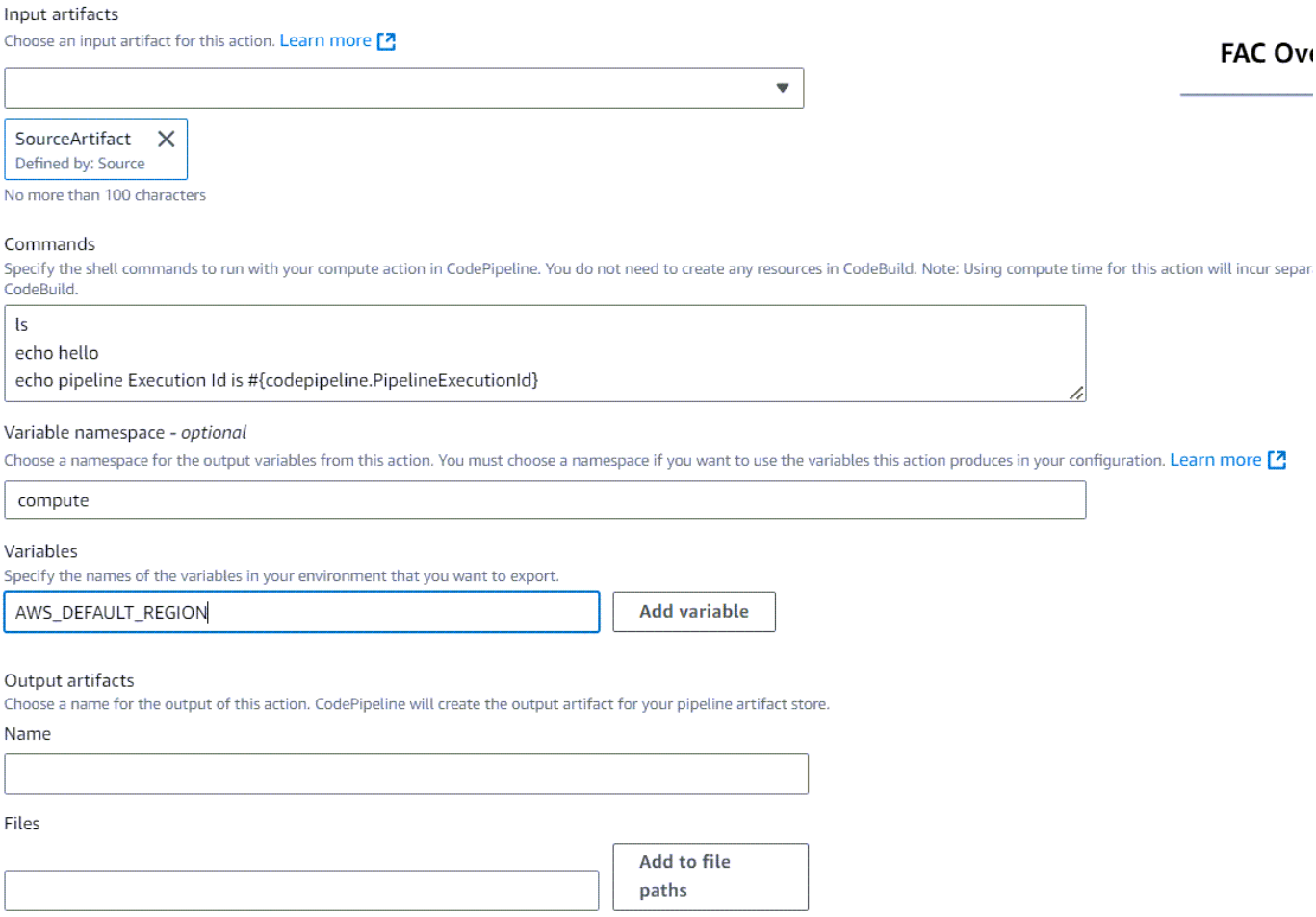
11. [ステップ 6: デプロイステージの追加] ステージで、[デプロイステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

    [次へ] を選択します。

12. ステップ 7: 情報を確認してから、パイプラインの作成を選択します。

13. アクションを作成するための最後のステップとして、アクションの出力変数となる環境変数をアクションに追加します。コマンドアクションで、[編集] を選択します。[編集] 画面で、[変数の名前空間] フィールドに「compute」と入力して、アクションの変数の名前空間を指定します。

    CodeBuild 出力変数 AWS_Default_Region を追加し、[変数を追加] を選択します。



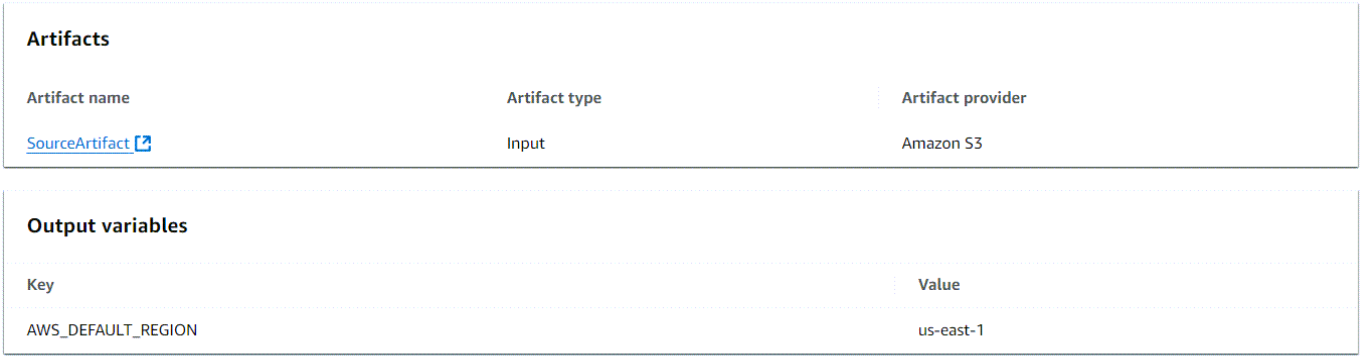
ステップ 3: パイプラインを実行してビルドコマンドを検証する

変更をリリースして、パイプラインを実行します。実行履歴、ビルドログ、出力変数を表示して、ビルドコマンドが実行したことを確認します。

アクションのログと出力変数を表示するには

1. パイプラインが正常に実行したら、アクションのログと出力を表示できます。
2. アクションの出力変数を表示するには、[履歴]し、[タイムライン] の順に選択します。

   アクションに追加した出力変数を表示します。コマンドアクションの出力に、アクション Region に解決された出力変数が表示されます。



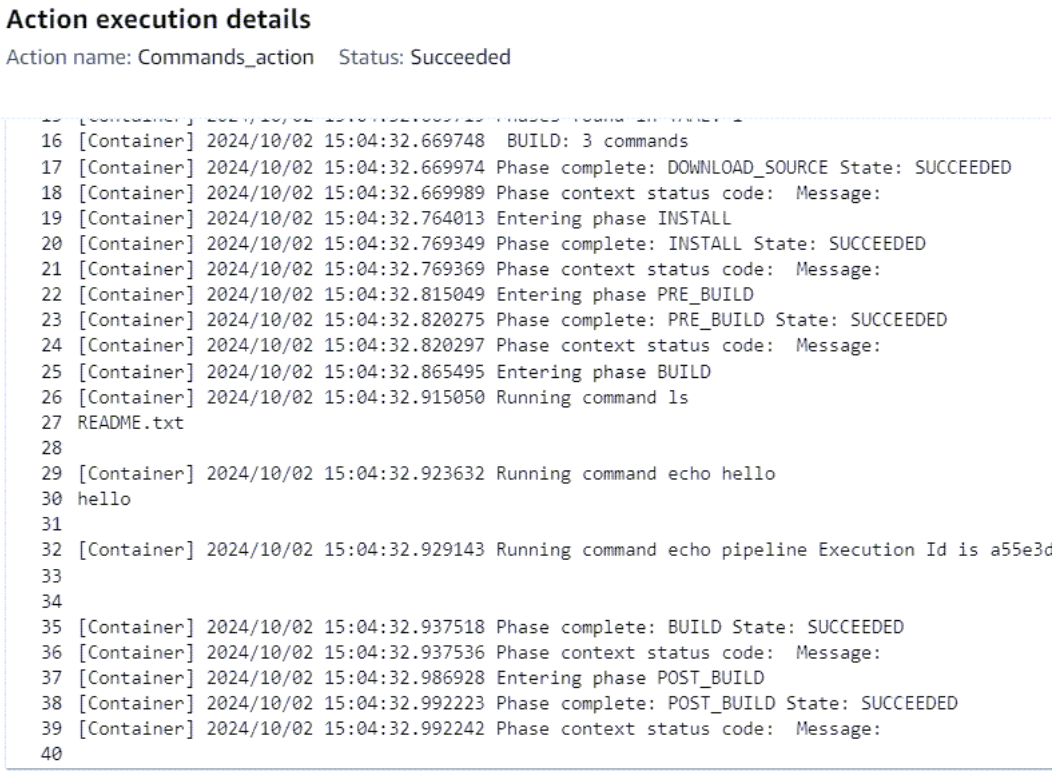
Artifacts

Artifact name	Artifact type	Artifact provider
SourceArtifact	Input	Amazon S3

Output variables

Key	Value
AWS_DEFAULT_REGION	us-east-1

3. アクションのログを表示するには、成功したコマンドアクションの [詳細を表示] を選択します。コマンドアクションのログを表示します。



Action execution details

Action name: Commands_action Status: Succeeded

```
16 [Container] 2024/10/02 15:04:32.669748  BUILD: 3 commands
17 [Container] 2024/10/02 15:04:32.669974 Phase complete: DOWNLOAD_SOURCE State: SUCCEEDED
18 [Container] 2024/10/02 15:04:32.669989 Phase context status code:  Message:
19 [Container] 2024/10/02 15:04:32.764013 Entering phase INSTALL
20 [Container] 2024/10/02 15:04:32.769349 Phase complete: INSTALL State: SUCCEEDED
21 [Container] 2024/10/02 15:04:32.769369 Phase context status code:  Message:
22 [Container] 2024/10/02 15:04:32.815049 Entering phase PRE_BUILD
23 [Container] 2024/10/02 15:04:32.820275 Phase complete: PRE_BUILD State: SUCCEEDED
24 [Container] 2024/10/02 15:04:32.820297 Phase context status code:  Message:
25 [Container] 2024/10/02 15:04:32.865495 Entering phase BUILD
26 [Container] 2024/10/02 15:04:32.915050 Running command ls
27 README.txt
28
29 [Container] 2024/10/02 15:04:32.923632 Running command echo hello
30 hello
31
32 [Container] 2024/10/02 15:04:32.929143 Running command echo pipeline Execution Id is a55e3d
33
34
35 [Container] 2024/10/02 15:04:32.937518 Phase complete: BUILD State: SUCCEEDED
36 [Container] 2024/10/02 15:04:32.937536 Phase context status code:  Message:
37 [Container] 2024/10/02 15:04:32.986928 Entering phase POST_BUILD
38 [Container] 2024/10/02 15:04:32.992223 Phase complete: POST_BUILD State: SUCCEEDED
39 [Container] 2024/10/02 15:04:32.992242 Phase context status code:  Message:
40
```

チュートリアル: Git タグを使用してパイプラインを開始する

このチュートリアルでは、GitHub リポジトリに接続するパイプラインを作成し、ソースアクションのトリガータイプとして Git タグを設定します。コミット時に Git タグが作成されると、パイプラインが開始されます。この例では、タグ名の構文に基づいてタグをフィルタ処理するパイプラインの作

成方法を示しています。glob パターンを使用したフィルタ処理の詳細については、「構文での glob パターンの使用」を参照してください。

> **Important**
>
> パイプライン作成の一環として、CodePipeline は、ユーザーが指定した S3 アーティファクトバケットをアーティファクトとして使用します (これは S3 ソースアクションで使用するバケットとは異なります)。S3 アーティファクトバケットがパイプラインのアカウントとは異なるアカウントにある場合は、S3 アーティファクトバケットが によって所有 AWS アカウント されており、安全で信頼できることを確認してください。

このチュートリアルでは、`CodeStarSourceConnection` アクションタイプを使用して GitHub に接続します。

> **Note**
>
> この機能は、アジアパシフィック (香港)、アフリカ (ケープタウン)、中東 (バーレーン)、または欧州 (チューリッヒ) リージョンでは利用できません。利用可能なその他のアクションについては、「CodePipeline との製品とサービスの統合」を参照してください。欧州 (ミラノ) リージョンでのこのアクションに関する考慮事項については、「CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアクションの場合)」の注意を参照してください。

トピック

- 前提条件
- ステップ 1: CloudShell を開いてリポジトリを複製する
- ステップ 2: Git タグでトリガーするパイプラインを作成する
- ステップ 3: リリースに対するコミットにタグを付ける
- ステップ 4: 変更をリリースしてログを表示する

前提条件

開始する前に、以下を実行する必要があります。

- GitHub アカウントで GitHub リポジトリを作成します。
- GitHub の認証情報を準備してください。を使用して接続 AWS マネジメントコンソール を設定すると、GitHub 認証情報を使用してサインインするように求められます。

ステップ 1: CloudShell を開いてリポジトリを複製する

コマンドラインインターフェイスを使用して、リポジトリの複製、コミット、タグの追加を行うことができます。このチュートリアルでは、コマンドラインインターフェイス用に CloudShell インスタンスを起動します。

1. AWS マネジメントコンソールにサインインします。
2. 上部のナビゲーションバーで、 AWS アイコンを選択します。 AWS マネジメントコンソール のメインページが表示されます。
3. 上部のナビゲーションバーで、 AWS CloudShell アイコンを選択します。CloudShell が開きます。CloudShell 環境が作成されるまで待ちます。

   > **Note**
   >
   > CloudShell アイコンが表示されない場合は、CloudShell でサポートされているリージョンにいることを確認してください。このチュートリアルは、米国西部 (オレゴン) リージョンにいることを前提としています。

4. GitHub で、目的のリポジトリに移動します。[コード] を選択してから、[HTTPS] を選択します。パスをコピーします。Git リポジトリのクローンを作成するアドレスがクリップボードにコピーされます。
5. 次のコマンドを実行してリポジトリを複製します。

   ```
   git clone https://github.com/<account>/MyGitHubRepo.git
   ```

6. プロンプトが表示されたら、GitHub アカウントの `Username` と `Password` を入力します。`Password` エントリには、アカウントのパスワードではなく、ユーザーが作成したトークンを使用する必要があります。

ステップ 2: Git タグでトリガーするパイプラインを作成する

このセクションでは、以下のアクションを使用してパイプラインを作成します。

- Bitbucket リポジトリとアクションへの接続を持つソースステージ。
- ビルドアクションを含む AWS CodeBuild ビルドステージ。

ウィザードを使用してパイプラインを作成するには

1. CodePipeline コンソール (http://console.aws.amazon.com/codesuite/codepipeline/home) にサインインします。
2. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。
3. [ステップ 1: 作成オプションを選択する] ページの [作成オプション] で、[カスタムパイプラインを構築する] オプションを選択します。[次へ] を選択します。
4. [ステップ 2: パイプラインの設定を選択する] で、[パイプライン名] に「**MyGitHubTagsPipeline**」と入力します。
5. [パイプラインのタイプ] で、デフォルトの選択を [V2] のままにします。パイプラインのタイプによって特徴および価格が異なります。詳細については、「パイプラインのタイプ」を参照してください。
6. [サービスロール] で、[New service role (新しいサービスロール)] を選択します。

> **Note**
>
> 既存の CodePipeline サービスロールを代わりに使用する場合は、サービスロールポリシーに対する `codestar-connections:UseConnection` IAM アクセス許可を追加したことを確認してください。CodePipeline サービスロールの手順については、CodePipeline サービスロールにアクセス許可を追加する」を参照してください。

7. [詳細設定] では、デフォルト値のままにします。アーティファクトストアで、[Default location] (デフォルトの場所)を選択し、パイプライン用に選択したリージョン内のパイプラインのデフォルトのアーティファクトストア (デフォルトとして指定された Amazon S3 アーティファクトバケットなど) を使用します。

> **Note**
>
> これはソースコードのソースバケットではありません。パイプラインのアーティファクトストアです。パイプラインごとに S3 バケットなどの個別のアーティファクトストアが必要です。

[次へ] を選択します。

8. [ステップ 3: ソースステージの追加] ページで、ソースステージを追加します。

   a. [ソースプロバイダー] で、[GitHub (GitHub アプリ経由)] を選択します。

   b. 接続 で、既存の接続を選択するか、新規の接続を作成します。GitHub ソースアクション用の接続を作成または管理する方法については、GitHub コネクション を参照してください。

   c. リポジトリ名 で、GitHub リポジトリの名前を選択します。

   d. [デフォルトブランチ] で、パイプラインを手動で開始する場合、または Git タグではないソースイベントで開始する場合に指定するブランチを選択します。変更元がトリガーでない場合やパイプラインの実行が手動で開始された場合は、デフォルトブランチの HEAD コミットが変更として使用されます。

   e. [ウェブフックイベント] の [フィルタータイプ] で、[タグ] を選択します。

      [タグまたはパターン] フィールドに、`release*` と入力します。

      > **Important**
      >
      > Git タグのトリガータイプで開始されるパイプラインは、WebhookV2 イベントに対して設定されます。パイプラインの開始に Webhook イベント (すべてのプッシュイベントに対して変更検出を行う) は使用されません。

   [次へ] を選択します。

9. [Add build stage (ビルドステージの追加)] で、ビルドステージを追加します。

   a. [ビルドプロバイダ] で、[AWS CodeBuild] を選択します。[リージョン] をデフォルトでパイプラインのリージョンにすることを許可します。

   b. [プロジェクトを作成] を選択します。

   c. [プロジェクト名] に、このビルドプロジェクトの名前を入力します。

   d. [環境イメージ] で、[Managed image (マネージド型イメージ)] を選択します。[Operating system] で、[Ubuntu] を選択します。

   e. [ランタイム] で、[Standard (標準)] を選択します。[イメージ] で、[aws/codebuild/standard:5.0] を選択します。

   f. [サービスロール] で、[New service role (新しいサービスロール)] を選択します。

> **Note**
>
> CodeBuild サービスロールの名前を書き留めます。このチュートリアルの最後のステップでは、ロール名が必要になります。

g. [Buildspec] の Build specifications (ビルド仕様) で、[Insert build commands] (ビルドコマンドの挿入) を選択します。エディタに切り替え を選択し、ビルドコマンド に以下を貼り付けます。

```
version: 0.2
#env:
  #variables:
     # key: "value"
     # key: "value"
  #parameter-store:
     # key: "value"
     # key: "value"
  #git-credential-helper: yes
phases:
  install:
    #If you use the Ubuntu standard image 2.0 or later, you must specify
 runtime-versions.
    #If you specify runtime-versions and use an image other than Ubuntu
 standard image 2.0, the build fails.
    runtime-versions:
      nodejs: 12
    #commands:
      # - command
      # - command
  #pre_build:
    #commands:
      # - command
      # - command
  build:
    commands:
      -
  #post_build:
    #commands:
      # - command
      # - command
artifacts:
```

```
  files:
     - '*'
    # - location
  name: $(date +%Y-%m-%d)
  #discard-paths: yes
  #base-directory: location
#cache:
  #paths:
    # - paths
```

h. [Continue to CodePipeline] (CodePipeline に進む) を選択します。CodePipeline コンソールに戻り、ビルドコマンドを使用して設定する CodeBuild プロジェクトが作成されます。ビルドプロジェクトでは、サービスロールを使用して AWS のサービス アクセス許可を管理します。このステップには数分かかる場合があります。

i. [次へ] を選択します。

10. [ステップ 5: テストステージの追加] で、[テストステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

    [次へ] を選択します。

11. [ステップ 6: デプロイステージの追加] ページで、[デプロイステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。[次へ] を選択します。

12. ステップ 7: 確認で、パイプラインの作成を選択します。

ステップ 3: リリースに対するコミットにタグを付ける

パイプラインを作成し、Git タグを指定した後、GitHub リポジトリ内のコミットにタグを付けることができます。以下の手順では、コミットに `release-1` タグを付けます。Git リポジトリ内の各コミットには、それぞれ一意の Git タグが必要です。コミットを選択してタグを付けると、さまざまなブランチからの変更をパイプラインのデプロイに組み込むことができます。release というタグ名は、GitHub のリリースの概念には当てはまりません。

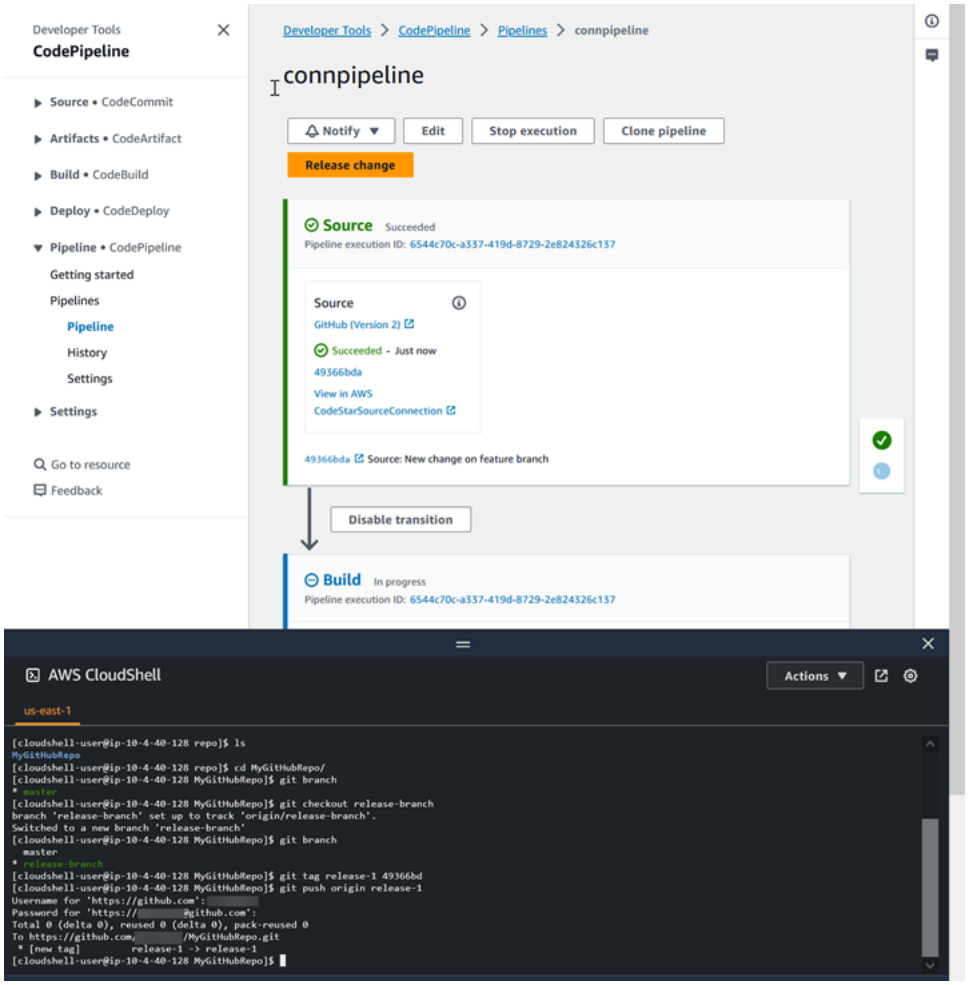
1. コピーしたコミット ID のうち、タグを付けるものを参照します。各ブランチのコミットを表示するには、CloudShell ターミナルで以下のコマンドを入力して、タグを付けるコミット ID を取得します。

```
git log
```

2. CloudShell ターミナルで、コミットにタグを付け、元のリポジトリにプッシュするコマンドを入力します。コミットにタグを付けた後、git Push コマンドを使用してタグを元のリポジトリにプッシュします。以下の例では、次のコマンドを入力して、ID 49366bd の 2 番目のコミットに `release-1` タグを使用しています。このタグはパイプラインの `release*` タグフィルタによって処理されて、パイプラインの実行が開始されます。

```
git tag release-1 49366bd
```

```
git push origin release-1
```



ステップ 4: 変更をリリースしてログを表示する

1. パイプラインが正常に実行されたら、デプロイステージで [ログの表示] を選択します。

[ログ] で、CodeBuild のビルド出力を表示します。このコマンドは、入力された変数の値を出力します。

2. [履歴] ページで、[トリガー] 列を表示します。トリガータイプ GitTag : release-1 を表示します。

チュートリアル: プルリクエストのブランチ名をフィルタリングしてパイプラインを開始する (V2 タイプ)

このチュートリアルでは、GitHub.com リポジトリに接続するパイプラインを作成します。このリポジトリでは、プルリクエストをフィルタリングするトリガー設定でパイプラインを開始するようにソースアクションを設定します。指定したブランチに対して指定したプルリクエストイベントが発生すると、パイプラインが開始します。この例では、ブランチ名のフィルタリングを許可するパイプラインの作成方法を示します。トリガーの操作の詳細については、「プッシュリクエストイベントタイプとプルリクエストイベントタイプのフィルターを追加する (CLI)」を参照してください。glob 形式の正規表現パターンを使用したフィルタリングの詳細については、「構文での glob パターンの使用」を参照してください。

> **Important**
>
> パイプライン作成の一環として、CodePipeline は、ユーザーが指定した S3 アーティファクトバケットをアーティファクトとして使用します (これは S3 ソースアクションで使用するバケットとは異なります)。S3 アーティファクトバケットがパイプラインのアカウントとは異なるアカウントにある場合は、S3 アーティファクトバケットが によって所有 AWS アカウント されており、安全で信頼できることを確認してください。

このチュートリアルでは、`CodeStarSourceConnection` アクションタイプを使用して GitHub.com に接続します。

トピック

- 前提条件
- ステップ 1: 指定したブランチのプルリクエストに応じて開始するパイプラインを作成する
- ステップ 2: GitHub.com でプルリクエストを作成してマージし、パイプライン実行を開始する

前提条件

開始する前に、以下を実行する必要があります。

- GitHub.com アカウントで GitHub.com リポジトリを作成します。
- GitHub の認証情報を準備してください。を使用して接続 AWS マネジメントコンソール を設定すると、GitHub 認証情報を使用してサインインするように求められます。

ステップ 1: 指定したブランチのプルリクエストに応じて開始するパイプラインを作成する

このセクションでは、以下のアクションを使用してパイプラインを作成します。

- GitHub.com のリポジトリおよびアクションへの接続を持つソースステージ。
- ビルドアクションを含む AWS CodeBuild ビルドステージ。

ウィザードを使用してパイプラインを作成するには

1. CodePipeline コンソール (http://console.aws.amazon.com/codesuite/codepipeline/home) にサインインします。
2. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。
3. [ステップ 1: 作成オプションを選択する] ページの [作成オプション] で、[カスタムパイプラインを構築する] オプションを選択します。[次へ] を選択します。
4. [ステップ 2: パイプラインの設定を選択する] で、[パイプライン名] に「**MyFilterBranchesPipeline**」と入力します。
5. [パイプラインのタイプ] で、デフォルトの選択を [V2] のままにします。パイプラインのタイプによって特徴および価格が異なります。詳細については、「パイプラインのタイプ」を参照してください。
6. [サービスロール] で、[New service role (新しいサービスロール)] を選択します。

> **Note**
>
> 既存の CodePipeline サービスロールを代わりに使用する場合は、サービスロールポリシーに対する `codeconnections:UseConnection` IAM アクセス許可を追加したこと

を確認してください。CodePipeline サービスロールの手順については、CodePipeline サービスロールにアクセス許可を追加する」を参照してください。

7. [詳細設定] では、デフォルト値のままにします。アーティファクトストアで、[Default location] (デフォルトの場所)を選択し、パイプライン用に選択したリージョン内のパイプラインのデフォルトのアーティファクトストア (デフォルトとして指定された Amazon S3 アーティファクトバケットなど) を使用します。

> **Note**
>
> これはソースコードのソースバケットではありません。パイプラインのアーティファクトストアです。パイプラインごとに S3 バケットなどの個別のアーティファクトストアが必要です。

[次へ] を選択します。

8. [ステップ 3: ソースステージの追加] ページで、ソースステージを追加します。

   a. [ソースプロバイダー] で、[GitHub (GitHub アプリ経由)] を選択します。

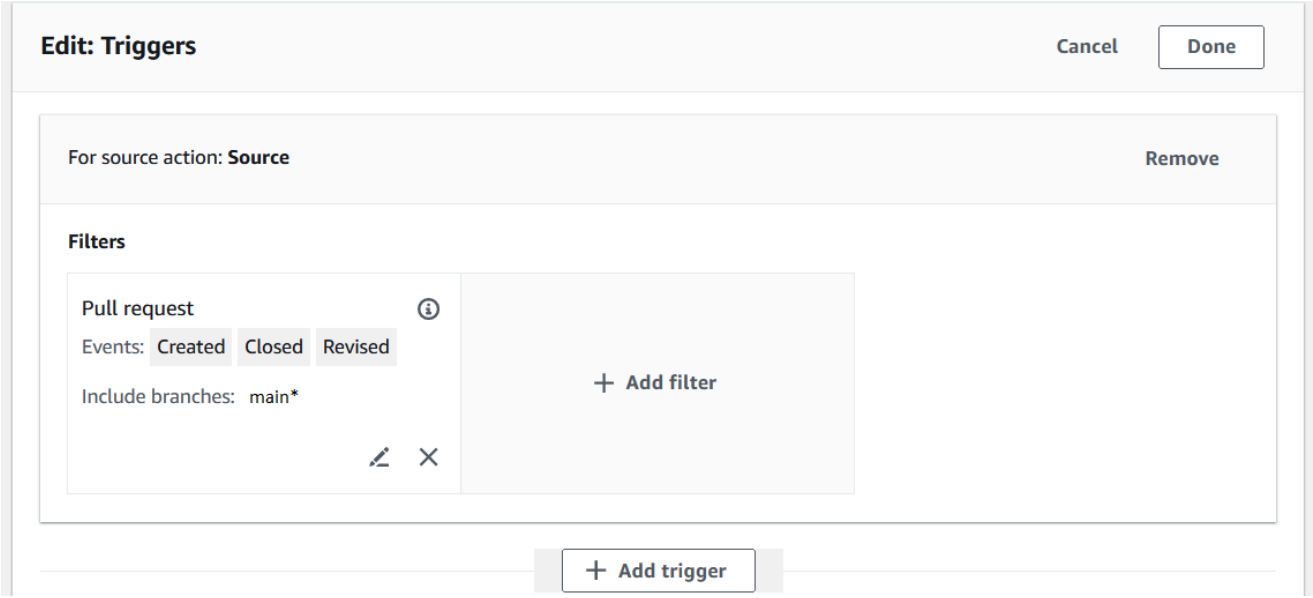
   b. 接続 で、既存の接続を選択するか、新規の接続を作成します。GitHub ソースアクション用の接続を作成または管理するには、「GitHub コネクション」を参照してください。

   c. [リポジトリ名] で、GitHub.com リポジトリの名前を選択します。

   d. [トリガータイプ] で、[フィルターを指定] を選択します。

      [イベントタイプ] で、[プルリクエスト] を選択します。プルリクエストですべてのイベントを選択し、プルリクエストの作成、更新、またはクローズに応じてイベントが発生するようにします。

      [ブランチ] で、[含める] フィールドに「`main*`」と入力します。



> **Important**
>
> このトリガータイプで開始するパイプラインは、WebhookV2 イベントに対して設定します。パイプラインの開始には Webhook イベント (すべてのプッシュイベントでの変更検出) を使用しません。

[次へ] を選択します。

9. [ステップ 4: ビルドステージの追加] の [ビルドプロバイダー] で、[AWS CodeBuild] を選択します。[リージョン] をデフォルトでパイプラインのリージョンにすることを許可します。「チュートリアル: Git タグを使用してパイプラインを開始する」の指示に従って、ビルドプロジェクトを選択または作成します。このアクションは、パイプラインの作成に必要な 2 番目のステージとして、このチュートリアルでのみ使用します。
10. [ステップ 5: テストステージの追加] で、[テストステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

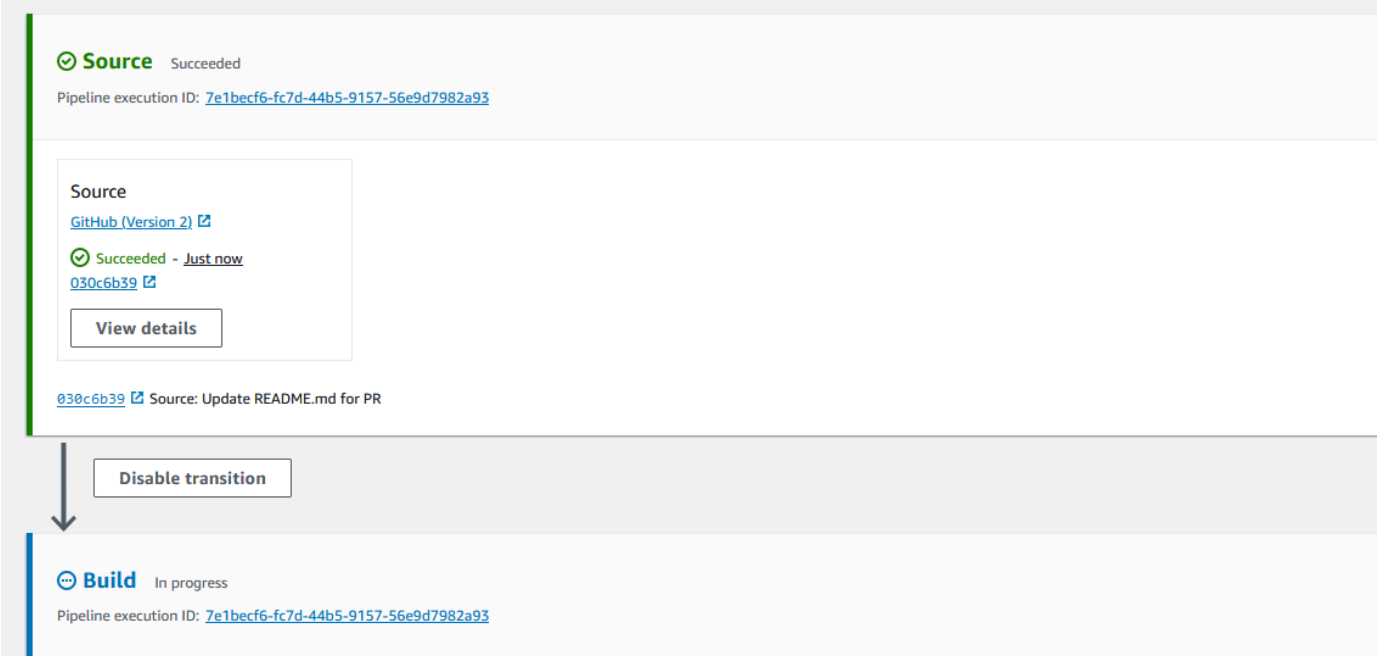
    [次へ] を選択します。
11. [ステップ 6: デプロイステージの追加] ページで、[デプロイステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。[次へ] を選択します。
12. ステップ 7: 確認で、パイプラインの作成を選択します。

ステップ 2: GitHub.com でプルリクエストを作成してマージし、パイプライン実行を開始する

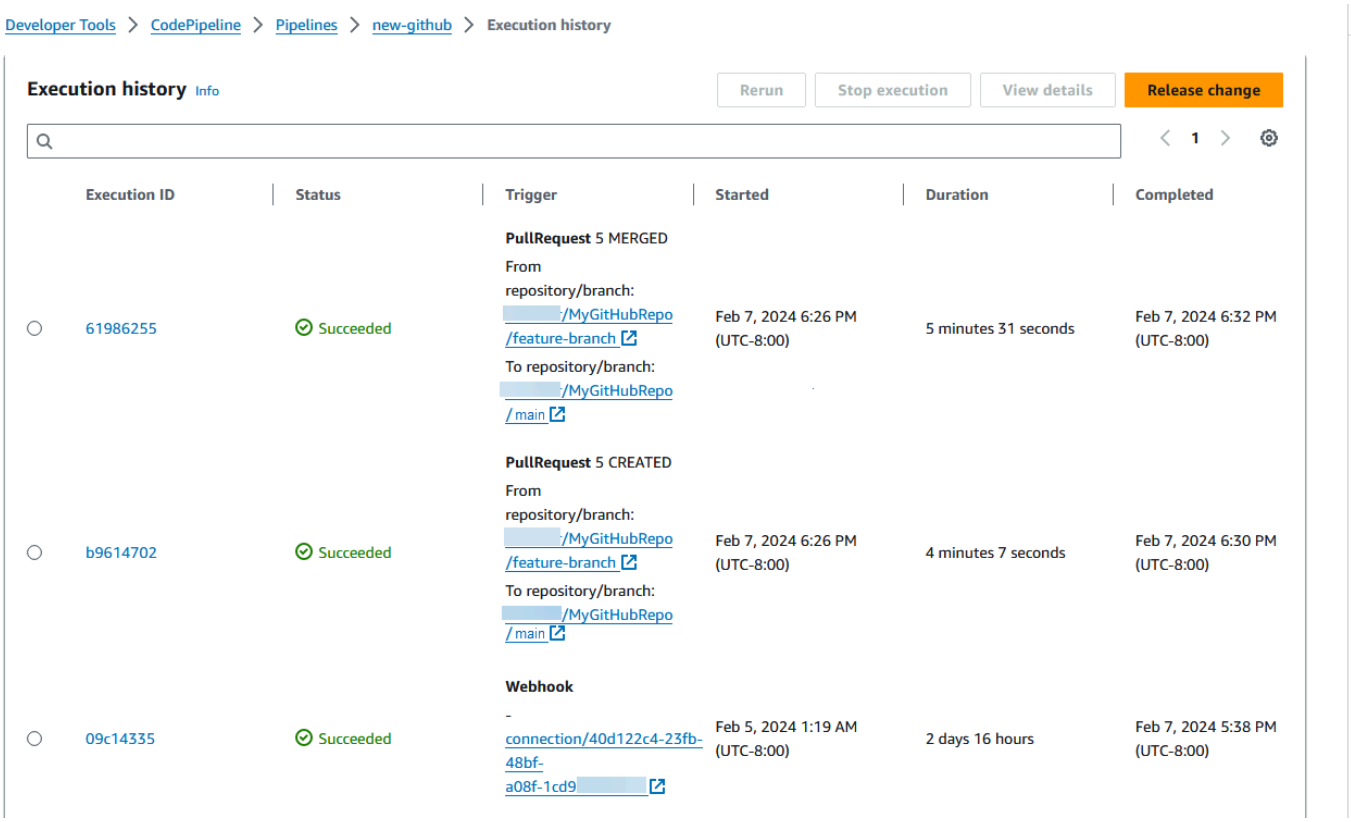
このセクションでは、プルリクエストを作成してマージします。これにより、プルリクエストを開く 1 つの実行と、プルリクエストを閉じる 1 つの実行でパイプラインが開始します。

プルリクエストを作成してパイプラインを開始するには

1. GitHub.com で、機能ブランチの README.md に変更を加え、`main` ブランチにプルリクエストを送信することで、プルリクエストを作成します。`Update README.md for PR` のようなメッセージを付けて変更をコミットします。
2. パイプラインは、プルリクエストのソースメッセージとして Update README.md for PR を示すソースリビジョンで開始します。



3. [履歴] を選択します。パイプライン実行履歴で、パイプライン実行を開始した CREATED および MERGED プルリクエストのステータスイベントを表示します。



チュートリアル: パイプラインレベルの変数を使用する

このチュートリアルでは、パイプラインを作成し、パイプラインレベルの変数を追加します。そのパイプラインで、変数の値を出力する CodeBuild ビルドアクションが実行されるようにします。

> **Important**
>
> パイプライン作成の一環として、CodePipeline は、ユーザーが指定した S3 アーティファクトバケットをアーティファクトとして使用します (これは S3 ソースアクションで使用するバケットとは異なります)。S3 アーティファクトバケットがパイプラインのアカウントとは異なるアカウントにある場合は、S3 アーティファクトバケットが によって所有 AWS アカウント されており、安全で信頼できることを確認してください。

トピック

- 前提条件
- ステップ 1: パイプラインを作成してプロジェクトをビルドする
- ステップ 2: 変更をリリースしてログを表示する

前提条件

開始する前に、以下を実行する必要があります。

- CodeCommit リポジトリを作成します。
- リポジトリに .txt ファイルを追加します。

ステップ 1: パイプラインを作成してプロジェクトをビルドする

このセクションでは、以下のアクションを使用してパイプラインを作成します。

- CodeCommit リポジトリへの接続を持つソースステージ。
- ビルドアクションを含む AWS CodeBuild ビルドステージ。

ウィザードを使用してパイプラインを作成するには

1. CodePipeline コンソール (http://console.aws.amazon.com/codesuite/codepipeline/home) にサインインします。
2. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。
3. [ステップ 1: 作成オプションを選択する] ページの [作成オプション] で、[カスタムパイプラインを構築する] オプションを選択します。[次へ] を選択します。
4. [ステップ 2: パイプラインの設定を選択する] で、[パイプライン名] に「**MyVariablesPipeline**」と入力します。
5. [パイプラインのタイプ] で、デフォルトの選択を [V2] のままにします。パイプラインのタイプによって特徴および価格が異なります。詳細については、「パイプラインのタイプ」を参照してください。
6. [サービスロール] で、[New service role (新しいサービスロール)] を選択します。

> **Note**
>
> 既存の CodePipeline サービスロールを代わりに使用する場合は、サービスロールポリシーに対する `codeconnections:UseConnection` IAM アクセス許可を追加したことを確認してください。CodePipeline サービスロールの手順については、CodePipeline サービスロールにアクセス許可を追加する」を参照してください。

7. [変数] で、[変数の追加] を選択します。[名前] に「`timeout`」と入力します。[デフォルト] に「1000」と入力します。説明として「**`Timeout`** 」と入力します。

   これにより、パイプラインの実行開始時に値を宣言できる変数が作成されます。変数名は `[A-Za-z0-9@\-_]+` と一致する必要があり、空文字列以外であれば任意の名前で構いません。

8. [詳細設定] では、デフォルト値のままにします。アーティファクトストアで、[Default location] (デフォルトの場所)を選択し、パイプライン用に選択したリージョン内のパイプラインのデフォルトのアーティファクトストア (デフォルトとして指定された Amazon S3 アーティファクトバケットなど) を使用します。

   > **Note**
   >
   > これはソースコードのソースバケットではありません。パイプラインのアーティファクトストアです。パイプラインごとに S3 バケットなどの個別のアーティファクトストアが必要です。

   [次へ] を選択します。

9. [ステップ 3: ソースステージを追加する] ページでソースステージを追加します。

   a. [ソースプロバイダ] で、AWS CodeCommit を選択します。

   b. リポジトリ名とブランチ名で、リポジトリとブランチを選択します。

   [次へ] を選択します。

10. [ステップ 4: ビルドステージの追加] で、ビルドステージを追加します。

    a. [ビルドプロバイダ] で、[AWS CodeBuild] を選択します。[リージョン] をデフォルトでパイプラインのリージョンにすることを許可します。

    b. [プロジェクトを作成] を選択します。

    c. [プロジェクト名] に、このビルドプロジェクトの名前を入力します。

    d. [環境イメージ] で、[Managed image (マネージド型イメージ)] を選択します。[Operating system] で、[Ubuntu] を選択します。

    e. [ランタイム] で、[Standard (標準)] を選択します。[イメージ] で、[aws/codebuild/standard:5.0] を選択します。

    f. [サービスロール] で、[New service role (新しいサービスロール)] を選択します。

> **Note**
>
> CodeBuild サービスロールの名前を書き留めます。このチュートリアルの最後のステップでは、ロール名が必要になります。

g. [Buildspec] の Build specifications (ビルド仕様) で、[Insert build commands] (ビルドコマンドの挿入) を選択します。エディタに切り替え を選択し、ビルドコマンド に以下を貼り付けます。buildspec では、カスタム変数 $CUSTOM_VAR1 を使用してパイプライン変数をビルドログに出力します。次のステップでは、$CUSTOM_VAR1 出力変数を環境変数として作成します。

```
version: 0.2
#env:
  #variables:
     # key: "value"
     # key: "value"
  #parameter-store:
     # key: "value"
     # key: "value"
  #git-credential-helper: yes
phases:
  install:
    #If you use the Ubuntu standard image 2.0 or later, you must specify
 runtime-versions.
    #If you specify runtime-versions and use an image other than Ubuntu
 standard image 2.0, the build fails.
    runtime-versions:
      nodejs: 12
    #commands:
      # - command
      # - command
  #pre_build:
    #commands:
      # - command
      # - command
  build:
    commands:
      - echo $CUSTOM_VAR1
  #post_build:
    #commands:
      # - command
```

```
      # - command
artifacts:
  files:
     - '*'
    # - location
  name: $(date +%Y-%m-%d)
  #discard-paths: yes
  #base-directory: location
#cache:
  #paths:
    # - paths
```

h. [Continue to CodePipeline] (CodePipeline に進む) を選択します。CodePipeline コンソールに戻り、ビルドコマンドを使用して設定する CodeBuild プロジェクトが作成されます。ビルドプロジェクトでは、サービスロールを使用して AWS のサービス アクセス許可を管理します。このステップには数分かかる場合があります。

i. [環境変数 - オプション] で、パイプラインレベルの変数によって解決されるビルドアクションの入力変数として環境変数を作成するには、[環境変数の追加] を選択します。これにより、buildspec で指定した変数が $CUSTOM_VAR1 として作成されます。[名前] に「CUSTOM_VAR1」と入力します。[値] には「#{variables.timeout}」と入力します。[タイプ] で、[Plaintext] を選択します。

   環境変数の#{variables.timeout}値は、パイプラインレベルの変数名前空間variablesと、ステップ 7 でパイプライン用にtimeout作成されたパイプラインレベルの変数に基づいています。

j. [次へ] を選択します。

11. [ステップ 5: テストステージの追加] で、[テストステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

    [次へ] を選択します。

12. [ステップ 6: デプロイステージの追加] ページで、[デプロイステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。[次へ] を選択します。

13. ステップ 7: 確認で、パイプラインの作成を選択します。

ステップ 2: 変更をリリースしてログを表示する

1. パイプラインが正常に実行されたら、成功したビルドステージで [詳細を表示] を選択します。

詳細ページで、[ログ] タブを選択します。CodeBuild ビルド出力を表示します。このコマンドは、入力された変数の値を出力します。

2. 左側のナビゲーションで、[履歴] を選択します。

   最近の実行を選択し、[変数] タブを選択します。パイプライン変数の解決された値を表示します。

チュートリアル: シンプルなパイプラインを作成する (S3 バケット)

パイプラインを作成する最も簡単な方法は、 AWS CodePipeline コンソールでパイプラインの作成ウィザードを使用することです。

このチュートリアルでは、バージョン管理された S3 ソースバケットおよび CodeDeploy を使用してサンプルアプリケーションをリリースする 2 ステージのパイプラインを作成します。

> **Note**
>
> Amazon S3 がパイプラインのソースプロバイダーである場合、ソースファイルを 1 つの .zip に圧縮し、その .zip をソースバケットにアップロードできます。解凍されたファイルを 1 つアップロードすることもできます。ただし、.zip ファイルを想定するダウンストリームアクションは失敗します。

> **Important**
>
> パイプライン作成の一環として、CodePipeline は、ユーザーが指定した S3 アーティファクトバケットをアーティファクトとして使用します (これは S3 ソースアクションで使用するバケットとは異なります)。S3 アーティファクトバケットがパイプラインのアカウントとは異なるアカウントにある場合は、S3 アーティファクトバケットが によって所有 AWS アカウント されており、安全で信頼できることを確認してください。

このシンプルなパイプラインを作成したら、別のステージを追加し、ステージ間の移行を無効化または有効化します。

> **Important**
>
> この手順でパイプラインに追加するアクションの多くには、パイプラインを作成する前に作成する必要がある AWS リソースが含まれます。ソースアクションの AWS リソースは常に、パイプラインを作成するのと同じ AWS リージョンで作成する必要があります。例えば、米国東部 (オハイオ) リージョンにパイプラインを作成している場合、CodeCommit リポジトリは米国東部 (オハイオ) リージョンにある必要があります。
> パイプラインの作成時にクロスリージョンアクションを追加できます。クロスリージョンアクションの AWS リソースは、アクションを実行する予定のリージョンと同じ AWS リージョンに存在する必要があります。詳細については、「CodePipeline でクロスリージョンアクションを追加する」を参照してください。

開始する前に、「CodePipeline の使用開始」の前提条件を完了する必要があります。

トピック

- ステップ 1: アプリケーションの S3 バケットを作成する
- ステップ 2: Amazon EC2 Windows インスタンスを作成し、CodeDeploy エージェントをインストールします。
- ステップ 3: CodeDeploy でアプリケーションを作成する
- ステップ 4: CodePipeline で最初のパイプラインを作成する
- (オプション) ステップ 5: 別のステージをパイプラインに追加する
- (オプション) ステップ 6: CodePipeline でステージ間の移行を有効または無効にする
- ステップ 7: リソースをクリーンアップする

ステップ 1: アプリケーションの S3 バケットを作成する

ソースファイルまたはアプリケーションをバージョニングされた場所に保存します。このチュートリアルでは、サンプルアプリケーションファイルの S3 バケットを作成し、そのバケットでバージョニングを有効にします。バージョニングを有効化したら、サンプルアプリケーションをそのバケットにコピーします。

S3 バケットを作成するには

1. コンソールにサインインします AWS マネジメントコンソール。S3 コンソールを開きます。
2. [バケットを作成] を選択します。

3. [バケット名] に、バケットの名前 (**awscodepipeline-demobucket-example-date** など) を入力します。

   > **Note**
   >
   > Amazon S3 内のすべてのバケット名は一意になる必要があるため、例に示す名前ではなく、独自のバケット名を使用してください。例に示す名前は、日付を追加するだけでも変更できます。このチュートリアルの残りの部分で必要となるため、この名前を書き留めます。

   [リージョン] で、パイプラインを作成するリージョン [米国西部 (オレゴン)] などを選択し、[バケットの作成] を選択します。

4. バケットが作成されると、成功バナーが表示されます。[バケットの詳細に移動] を選択します。
5. [プロパティ] タブで、[バージョニング] を選択します。[バージョニングの有効化] を選択し、[保存] を選択します。

   バージョニングが有効になったら、Amazon S3 によって各オブジェクトのすべてのバージョンがバケットに保存されます。
6. [アクセス許可] タブは、デフォルト設定のままにします。S3 バケットおよびオブジェクトへのアクセス許可に関する詳細については、「ポリシーでのアクセス許可の指定」を参照してください。
7. 次に、サンプル をダウンロードし、ローカルコンピュータのフォルダまたはディレクトリに保存します。

   a. 次のいずれかを選択します 。Windows Server インスタンスについて、このチュートリアルのステップに従う場合は、`SampleApp_Windows.zip` を選択します。

      - CodeDeploy を使用して Amazon Linux インスタンスにデプロイする場合は、サンプルアプリケーションを SampleApp_Linux.zip からダウンロードします。
      - CodeDeploy を使用して Windows Server インスタンスにデプロイする場合は、サンプルアプリケーションを SampleApp_Windows.zip からダウンロードします。

      サンプルアプリケーションには、CodeDeploy を使用してデプロイするための以下のファイルが含まれています。

- `appspec.yml` - 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 形式のファイルです。AppSpec ファイルの詳細については、AWS CodeDeploy ユーザーガイドの「CodeDeploy AppSpec ファイルリファレンス」を参照してください。
- `index.html` - インデックスファイルには、デプロイされたサンプルアプリケーションのホームページが含まれています。
- `LICENSE.txt` - ライセンスファイルには、サンプルアプリケーションのライセンス情報が含まれています。
- スクリプトのファイル - サンプルアプリケーションはスクリプトを使用して、インスタンス上の場所にテキストファイルを書き込みます。以下のように、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。
  - (Linux サンプルのみ) `scripts` フォルダ - このフォルダに入っているのはシェルスクリプト `install_dependencies`、`start_server`、`stop_server` です。依存関係をインストールし、自動デプロイのサンプルアプリケーションを起動および停止するために使用されます。
  - (Windows サンプルのみ) `before-install.bat` - `BeforeInstall` デプロイライフサイクルイベントのバッチスクリプトです。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場所をインスタンス上に作成するために実行されます。

b. 圧縮 (zip) ファイルをダウンロードします。このファイルを解凍しないでください。

8. Amazon S3 コンソールで、バケットに次のファイルをアップロードします。

   a. アップロード を選択します。

   b. ファイルをドラッグアンドドロップするか、[ファイルを追加] を選択してファイルを参照します。

   c. アップロード を選択します。

ステップ 2: Amazon EC2 Windows インスタンスを作成し、CodeDeploy エージェントをインストールします。

> **Note**
>
> このチュートリアルでは、Amazon EC2 Windows インスタンスを作成するサンプル手順を示します。Amazon EC2 Linux インスタンスを作成するサンプルステップについては、「ス

テップ 3: Amazon EC2 Linux インスタンスを作成して 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。作成するインスタンスの数の入力を求められたら、2 つのインスタンスを指定します。

このステップでは、サンプルアプリケーションをデプロイする Windows Server Amazon EC2 インスタンスを作成します。このプロセスの一環として、インスタンス上で CodeDeploy エージェントのインストールと管理を許可するポリシーを関連付けたインスタンスロールを作成します。CodeDeploy エージェントは、CodeDeploy デプロイでインスタンスを使用できるようにするソフトウェアパッケージです。また、CodeDeploy エージェントによってアプリケーションのデプロイに使用されるファイルを取得すること、SSM によって管理されることを、インスタンスに許可するポリシーをアタッチします。

インスタンス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. コンソールダッシュボードで [ロール] を選択します。
3. [ロールの作成] を選択してください。
4. [信頼されたエンティティのタイプを選択] で、[AWS のサービス] を選択します。[ユースケースの選択] で [EC2] を選択し、[次の手順: アクセス許可] を選択します。
5. **AmazonEC2RoleforAWSCodeDeploy** という名前のマネージドポリシーを検索して選択します。
6. **AmazonSSMManagedInstanceCore** という名前のマネージドポリシーを検索して選択します。[Next: Tags (次へ: タグ)] を選択します。
7. [次へ: レビュー] を選択します。ロールの名前を入力します (例: **EC2InstanceRole**)。

> **Note**
>
> 次のステップのロール名をメモしておきます。このロールは、インスタンスの作成時に選択します。

[ロールの作成] を選択してください。

インスタンスを起動するには

1. Amazon EC2 コンソールの https://console.aws.amazon.com/ec2/ を開いてください。
2. サイドナビゲーションから [インスタンス] を選択し、ページの上部から [インスタンスの起動] を選択します。
3. [名前とタグ] で、[名前] に「**MyCodePipelineDemo**」と入力します。これにより、インスタンスにはキーが **Name** で、値が **MyCodePipelineDemo** というタグがが割り当てられます。後で、そのインスタンスにサンプルアプリケーションをデプロイする CodeDeploy アプリケーションを作成します。CodeDeploy は、タグに基づいてデプロイするインスタンスを選択します。
4. [アプリケーションと OS イメージ (Amazon マシンイメージ)] で、[Windows] オプションを選択します。(この AMI は Microsoft Windows Server 2019 Base と呼ばれ、「無料利用枠対象」というラベルが付けられ、クイックスタートにあります) 。
5. [インスタンスタイプ] で、インスタンスのハードウェア構成として無料利用枠対象となる `t2.micro` タイプを選択します。
6. [キーペア (ログイン)] で、キーペアを選択するか作成します。

   [キーペアなしで続行] を選択することもできます。

   > **Note**
   >
   > このチュートリアルでは、キーペアを使用せずに続行できます。SSH を使用してインスタンスに接続するには、キーペアを作成または使用します。

7. [ネットワーク設定] で、次の操作を行います。

   [パブリック IP の自動割り当て] で、ステータスが [有効] になっていることを確認します。

   - [セキュリティグループの割り当て] の横にある [新規セキュリティグループを作成] を選択します。
   - [SSH] の行で、[ソースタイプ] の [マイ IP] を選択します。
   - [セキュリティグループの追加]、[HTTP] の順に選択し、[ソースタイプ] で [マイ IP] を選択します。
8. [Advanced Details] (高度な詳細) を展開します。[IAM インスタンスプロファイル] で、前の手順で作成した IAM ロール (**EC2InstanceRole** など) を選択します。
9. [概要] の [インスタンス数] に 2 と入力します。

10. [インスタンスを起動] を選択します。
11. [View all instances] (すべてのインスタンスの表示) を選択して確認ページを閉じ、コンソールに戻ります。
12. [インスタンス] ページで、起動のステータスを表示できます。インスタンスを起動すると、その初期状態は `pending` です。インスタンスを起動した後は、状態が `running` に変わり、パブリック DNS 名を受け取ります ([パブリック DNS] 列が表示されていない場合は、[表示/非表示] アイコンを選択してから、[パブリック DNS] を選択します)。
13. インスタンスに接続可能になるまでには、数分かかることがあります。インスタンスのステータスチェックが成功していることを確認します。この情報は、[ステータスチェック] 列で確認できます。

ステップ 3: CodeDeploy でアプリケーションを作成する

CodeDeploy ではアプリケーションは、デプロイするコードの識別子で、名前の形式です。CodeDeploy はこの名前を使用して、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。このチュートリアルの後半でパイプラインを作成する際、このステップで作成した CodeDeploy アプリケーションの名前を選択します。

まず、CodeDeploy が使用するサービスロールを作成します。既にサービスロールを作成している場合は、別のサービスロールを作成する必要はありません。

CodeDeploy サービスロールの作成するために

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. コンソールダッシュボードで [ロール] を選択します。
3. [ロールの作成] を選択してください。
4. [信頼されたエンティティを選択] で、[AWS のサービス] を選択します。[ユースケース] で、[CodeDeploy] を選択します。示されたオプションから [CodeDeploy] を選択します。[次へ] を選択します。`AWSCodeDeployRole` マネージドポリシーはロールにアタッチ済みです。
5. [次へ] を選択します。
6. ロールの名前 (例: **`CodeDeployRole`**) を入力し、[ロールの作成] を選択します。

CodeDeploy でアプリケーションを作成するには

1. https://console.aws.amazon.com/codedeploy で、CodeDeploy コンソールを開きます。

2. アプリケーションページが表示されない場合は、 AWS CodeDeploy メニューでアプリケーションを選択します。
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、「`MyDemoApplication`」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。

CodeDeploy でデプロイグループを作成するには

1. アプリケーションが表示されるページで、[Create deployment group (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「**`MyDemoDeploymentGroup`**」と入力します。
3. [サービスロール] で、先ほど作成したサービスロールを選択します。少なくとも AWS CodeDeploy のサービスロールの作成で説明されている信頼とアクセス許可を使用して、CodeDeploy を信頼するサービスロールを使用する必要があります。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)」を参照してください。
4. [Deployment type] (デプロイタイプ) で、[In-place] (インプレース) を選択します。
5. [環境設定] で、[Amazon EC2 インスタンス] を選択します。[名前] を [キー] フィールドに入力し、[値] フィールドに **`MyCodePipelineDemo`** を入力します。

   > **Important**
   >
   > [Name (名前)] キーには、EC2 インスタンスの作成時にインスタンスに割り当てたのと同じ値を選択する必要があります。インスタンスに **`MyCodePipelineDemo`** 以外のタグを付けた場合は、ここでもそのタグを使用してください。

6. AWS Systems Manager を使用したエージェント設定で、今すぐ を選択し、更新をスケジュールします。これにより、インスタンスにエージェントがインストールされます。Windows インスタンスは既に SSM エージェントで設定されており、これから CodeDeploy エージェントで更新されます。
7. [デプロイ設定] で `CodeDeployDefault.OneAtaTime` を選択します。
8. [ロードバランサー] で、[ロードバランシングの有効化] ボックスが選択されていないことを確認してください。この例では、ロードバランサーを設定したり、ターゲットグループを選択したり

する必要はありません。チェックボックスの選択を解除すると、ロードバランサーのオプションが表示されません。

9. [詳細設定] セクションでは、既定のままにしておきます。
10. デプロイグループの作成 を選択します。

ステップ 4: CodePipeline で最初のパイプラインを作成する

チュートリアルのこの部分では、パイプラインを作成します。サンプルは、パイプラインを通して自動的に実行されます。

CodePipeline 自動リリースプロセスを作成するには

1. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。
3. [ステップ 1: 作成オプションを選択する] ページの [作成オプション] で、[カスタムパイプラインを構築する] オプションを選択します。[次へ] を選択します。
4. [ステップ 2: パイプラインの設定を選択する] で、[パイプライン名] に「**MyFirstPipeline**」と入力します。

   > **Note**
   >
   > パイプラインに別の名前を選択した場合は、このチュートリアルの残りの部分で **MyFirstPipeline** の代わりにその名前を使用してください。パイプラインを作成したら、その名前を変更することはできません。パイプラインの名前にはいくつかの制限がある場合があります。詳細については、「AWS CodePipeline のクォータ」を参照してください。

5. CodePipeline は、特徴と料金が異なる V1 タイプと V2 タイプのパイプラインを提供しています。V2 タイプは、コンソールで選択できる唯一のタイプです。詳細については、「パイプラインタイプ」を参照してください。CodePipeline の料金については、料金を参照してください。
6. [Service role (サービスロール)] で、次のいずれかの操作を行います。

   - New service role を選択して、CodePipeline に IAM での新しいサービスロールの作成を許可します。

- IAM で作成済みのサービスロールを使用するには、[Existing service role (既存のサービスロール)] を選択します。[ロール名] で、リストからサービスロールを選択します。

7. [詳細設定] をデフォルト設定のままにし、[次へ] を選択します。
8. [ステップ 3: ソースステージの追加] の [ソースプロバイダー] で、[Amazon S3] を選択します。[バケット] に、「ステップ 1: アプリケーションの S3 バケットを作成する」で作成した S3 バケットの名前を入力します。S3 オブジェクトキーで、ファイルパスの有無にかかわらずオブジェクトキーを入力し、必ずファイル拡張子を含めます。たとえば、`SampleApp_Windows.zip` の場合、次の例に示すように、サンプルファイル名を入力します。

   ```
   SampleApp_Windows.zip
   ```

   [次のステップ] を選択します。

   [Change detection options] で、デフォルト値のままにします。CodePipeline は Amazon CloudWatch Events を使用して、ソースバケットの変更を検出します。

   [次へ] を選択します。
9. [ステップ 4: ビルドステージの追加] で、[ビルドステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。[次へ] を選択します。
10. [ステップ 5: テストステージの追加] で、[テストステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

    [次へ] を選択します。
11. [ステップ 6: デプロイステージの追加] の [デプロイプロバイダー] で、[CodeDeploy] を選択します。リージョンフィールドのデフォルトはパイプライン AWS リージョン と同じです。[アプリケーション名] に `MyDemoApplication` を入力するか、更新ボタンを選択してリストからそのアプリケーション名を選択します。[デプロイグループ] に「**`MyDemoDeploymentGroup`**」と入力するか、リストからデプロイグループを選択して [次へ] を選択します。

    > **Note**
    >
    > 「Deploy」は、[ステップ 4: デプロイステージの追加] ステップで作成したステージにデフォルトで付けられる名前です。パイプラインの最初のステージに付けられる「Source」という名前も同様です。

12. ステップ 7: 情報を確認してから、パイプラインの作成を選択します。

13. パイプラインの実行が開始されます。CodePipeline サンプルがウェブページを CodeDeploy デプロイの各 Amazon EC2 インスタンスにデプロイしている間、進行状況と成功/失敗メッセージを表示できます。

お疲れ様でした。シンプルなパイプラインが CodePipeline に作成されました。パイプラインには 2 つのステージがあります。

- [Source] という名前のソースステージ。このステージでは、S3 バケットに保存したバージョニング済みのサンプルアプリケーションの変更を検出し、これらの変更をパイプライン内にプルします。
- デプロイ ステージでは、CodeDeploy を使用して、これらの変更を EC2 インスタンスにデプロイします。

ここで、結果を確認します。

パイプラインが正常に実行されたことを確認するには

1. パイプラインの最初の進行状況を表示します。各ステージのステータスは、[まだ実行はありません] から [進行中] に変わり、その後、[Succeeded (成功)] または [Failed (失敗)] のいずれかに変わります。パイプラインの最初の実行は数分で完了します。
2. アクションのステータスに [Succeeded (成功)] が表示されたら、[Deploy (デプロイ)] ステージのステータス領域で [Details (詳細)] を選択します。これにより、CodeDeploy コンソールが開きます。
3. [デプロイグループ] タブの [Deployment lifecycle events (デプロイライフサイクルイベント)] の下で、インスタンス ID を選択します。これにより、EC2 コンソールが開きます。
4. [Description] タブの [Public DNS] でアドレスをコピーし、ウェブブラウザーのアドレスバーに貼り付けます。S3 バケットにアップロードしたサンプルアプリケーションのインデックスページを表示します。

   S3 バケットにアップロードしたサンプルアプリケーションのウェブページが表示されます。

ステージ、アクション、パイプラインの仕組みの詳細については、「CodePipeline の概念 」を参照してください。

(オプション) ステップ 5: 別のステージをパイプラインに追加する

次に、別のステージをパイプラインに追加し、CodeDeploy を使用してステージングサーバーから本稼働サーバーにデプロイできるようにします。まず、CodeDeploy の CodePipelineDemoApplication に別のデプロイグループを作成します。その後、このデプロイグループを使用するアクションを含むステージを追加します。別のステージを追加するには、CodePipeline コンソールまたは AWS CLI を使用して、JSON ファイル内のパイプラインの構造を取得して手動で編集し、 update-pipeline コマンドを実行してパイプラインを変更で更新します。

トピック

- CodeDeploy で 2 番目のデプロイグループを作成するには
- パイプラインの別のステージとしてデプロイグループを追加する

CodeDeploy で 2 番目のデプロイグループを作成するには

> **Note**
>
> チュートリアルのこの部分では、2 番目のデプロイグループを作成しますが、以前と同じ Amazon EC2 インスタンスにデプロイします。このウォークスルーは、デモンストレーションのみを目的としています。CodePipeline でエラーを表示する方法を示すために、意図的に失敗するように設計されています。

CodeDeploy で 2 番目のデプロイグループを作成するには

1. 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] を選択し、アプリケーションのリストで [MyDemoApplication] を選択します。
3. [デプロイグループ] タブを選択して、[Create deployment group (デプロイグループの作成)] を選びます。
4. [Create deployment group (デプロイグループの作成)] ページの [Deployment group name (デプロイグループ名)] に、2 番目のデプロイグループの名前 (たとえば、**`CodePipelineProductionFleet`**) を入力します。
5. [サービスロール] で、最初のデプロイに使用したのと同じ CodeDeploy サービスロール (CodePipeline サービスロールではない) を選択します。
6. [Deployment type] (デプロイタイプ) で、[In-place] (インプレース) を選択します。

7. [環境設定] で、[Amazon EC2 インスタンス] を選択します。[名前] を [キー] ボックスから選択し、[値] ボックスで MyCodePipelineDemo をリストから選択します。[デプロイ設定] のデフォルト設定をそのままにします。
8. [デプロイ設定] で、[CodeDeployDefault.OneAtaTime] を選択します。
9. [Load Balancer (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。
10. デプロイグループの作成 を選択します。

パイプラインの別のステージとしてデプロイグループを追加する

別のデプロイグループが追加されたため、このデプロイグループを使用するステージを追加して、前に使用したのと同じ EC2 インスタンスにデプロイできます。CodePipeline コンソールまたは を使用して AWS CLI 、このステージを追加できます。

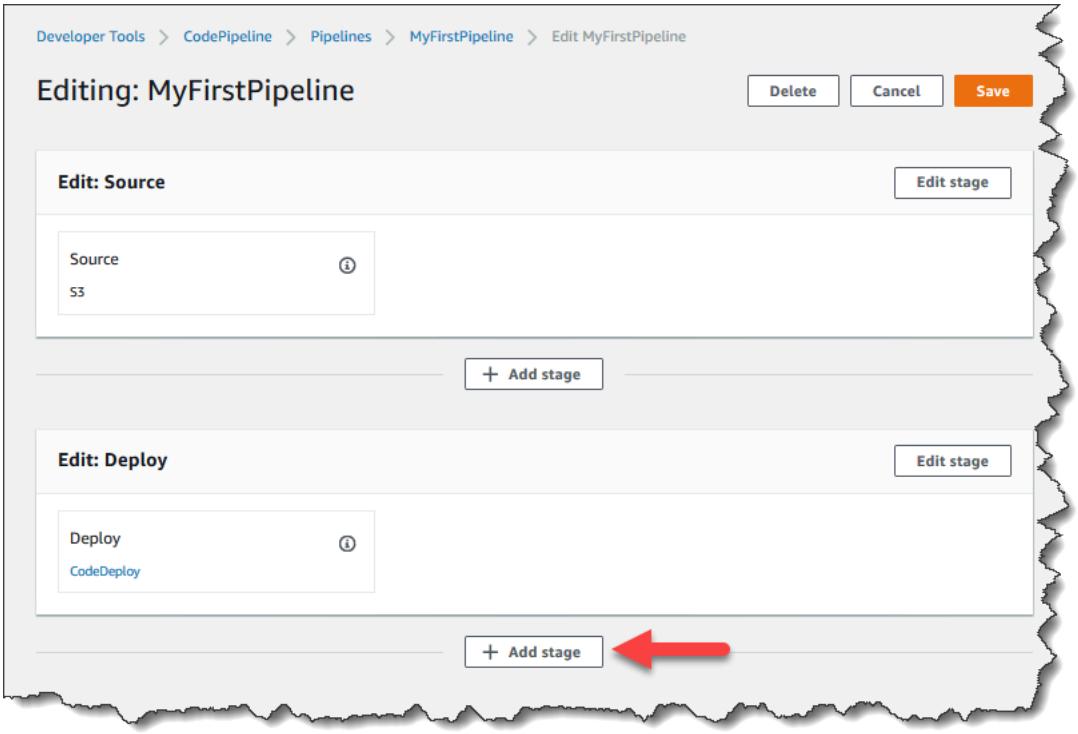
トピック

- 3 番目のステージを追加する (コンソール)
- 3 番目のステージを追加する (CLI)

3 番目のステージを追加する (コンソール)

CodePipeline コンソールを使用して、新しいデプロイグループを使用する新しいステージを追加できます。このデプロイグループのデプロイ先は、すでに使用した EC2 インスタンスであるため、このステージのデプロイアクションは失敗します。

1. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [名前] で、作成したパイプラインの名前 MyFirstPipeline を選択します。
3. パイプライン詳細ページで、[編集] を選択します。
4. [Edit (編集)] ページで [+ Add stage (+ ステージの追加)] を選択して、[Deploy] ステージの直後にステージを追加します。



5. [Add stage (ステージの追加)] で、[Stage name (ステージ名)] に、**Production** を入力します。[Add stage (ステージの追加)] を選択します。
6. 新しいステージで、[+ Add action group (+ アクショングループの追加)] を選択します。
7. [アクションの編集] の、[アクション名] に、**Deploy-Second-Deployment** を入力します。 [アクションプロバイダー] の [デプロイ] で、[CodeDeploy] を選択します。
8. CodeDeploy セクションの [アプリケーション名] で、パイプラインの作成時と同様に、ドロップダウンリストから `MyDemoApplication` を選択します。[デプロイグループ] で、先ほど作成したデプロイグループ **CodePipelineProductionFleet** を選択します。[入力アーティファクト] で、ソースアクションから入力アーティファクトを選択します。[保存] を選択します。
9. [Edit (編集)] ページで [Save (保存)] を選択します。[パイプラインの変更を保存] で、[Save (保存)] を選択します。
10. 新しいステージがパイプラインに追加されていますが、パイプラインの別の実行をトリガーした変更がないため、[まだ実行はありません] というステータスが表示されます。最新のリビジョンを手動で再度実行して、編集されたパイプラインの実行度を確認する必要があります。パイプラインの詳細ページで、[Release change (リリースの変更)] を選択し、プロンプトが表示されたら [Release (リリース)] を選択します。これにより、ソースアクションで指定した各ソース場所における最新のリビジョンがパイプラインで実行されます。

または、 AWS CLI を使用してパイプラインを再実行するには、ローカル Linux、macOS、Unix マシンのターミナル、またはローカル Windows マシンのコマンドプロンプトから、パイプラインの名前を指定して start-pipeline-execution コマンドを実行します。これにより、ソースバケット内のアプリケーションの 2 回目の実行がパイプラインで実行されます。

```
aws codepipeline start-pipeline-execution --name MyFirstPipeline
```

このコマンドは `pipelineExecutionId` オブジェクトを返します。

11. CodePipeline コンソールに戻り、パイプラインのリストで [MyFirstPipeline] を選択してビューページを開きます。

    パイプラインには、3 つのステージがあり、それらの各ステージのアーティファクトの状態が示されます。パイプラインがすべてのステージを実行するまでに最大 5 分かかることがあります。前回と同じように、最初の 2 つのステージではデプロイが成功しますが、[Production (本番稼働用)] ステージでは [Deploy-Second-Deployment (2 番目のデプロイをデプロイ)] アクションが失敗したことが示されます。

12. [Deploy-Second-Deployment] アクションで、[Details] を選択します。CodeDeploy デプロイのページにリダイレクトされます。この場合、最初のインスタンスグループがすべての EC2 インスタンスにデプロイされ、2 番目のデプロイグループ用のインスタンスが残っていないために失敗しています。

> **Note**
>
> この失敗は、パイプラインのステージにエラーがある場合にどうなるかを示すために、意図的に起こしたものです。

3 番目のステージを追加する (CLI)

を使用してパイプラインにステージ AWS CLI を追加するのは、 コンソールを使用するよりも複雑ですが、パイプラインの構造をより可視化できます。

パイプラインの 3 番目のステージを作成するには

1. ローカル Linux、macOS、または Unix マシンのターミナルセッションを開くか、ローカル Windows マシンのコマンドプロンプトを開き、get-pipeline コマンドを実行して、先ほど作成し

たパイプラインの構造を表示します。**MyFirstPipeline** に対して、以下のコマンドを入力します。

```
aws codepipeline get-pipeline --name "MyFirstPipeline"
```

このコマンドは、MyFirstPipeline の構造を返します。出力の最初の部分は以下のようになります。

```
{
    "pipeline": {
        "roleArn": "arn:aws:iam::80398EXAMPLE:role/AWS-CodePipeline-Service",
        "stages": [
    ...
```

出力の最後のパートにはパイプラインのメタデータが含まれており、次のようになります。

```
    ...
        ],
        "artifactStore": {
            "type": "S3"
            "location": "amzn-s3-demo-bucket",
        },
        "name": "MyFirstPipeline",
        "version": 4
    },
    "metadata": {
        "pipelineArn": "arn:aws:codepipeline:us-
east-2:80398EXAMPLE:MyFirstPipeline",
        "updated": 1501626591.112,
        "created": 1501626591.112
    }
}
```

2. この構造をコピーしてプレーンテキストエディタに貼り付け、ファイルを **pipeline.json** として保存します。便利なように、aws codepipeline コマンドを実行する同じディレクトリにこのファイルを保存します。

> **Note**
>
> 以下のように、get-pipeline コマンドを使用して、パイプ処理で JSON をファイルに渡すことができます。

```
aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
```

3. [Deploy (デプロイ)] ステージセクションをコピーし、最初の 2 つのステージの後に貼り付けます。これはデプロイステージであるため、[Deploy (デプロイ)] ステージと同様に、3 番目のステージのテンプレートとして使用します。
4. ステージの名前とデプロイグループの詳細を変更します。

   以下の例では、[デプロイ] ステージの後に pipeline.json ファイルに追加する JSON を示しています。強調表示された要素を新しい値で編集します。[Deploy (デプロイ)] と [Production (本番稼働用)] のステージ定義を区切るには、必ずカンマを使用してください。

```
,
{
    "name": "Production",
     "actions": [
        {
         "inputArtifacts": [
             {
              "name": "MyApp"
             }
          ],
          "name": "Deploy-Second-Deployment",
          "actionTypeId": {
              "category": "Deploy",
              "owner": "AWS",
              "version": "1",
              "provider": "CodeDeploy"
              },
          "outputArtifacts": [],
          "configuration": {
               "ApplicationName": "CodePipelineDemoApplication",
               "DeploymentGroupName": "CodePipelineProductionFleet"
                },
          "runOrder": 1
        }
    ]
}
```

5. get-pipeline コマンドを使用して取得したパイプライン構造を使用している場合、JSON ファイルから metadata 行を削除する必要があります。それ以外の場合

は、update-pipeline コマンドで使用することはできません。"metadata": { } 行と、"created"、"pipelineARN"、"updated" フィールドを削除します。

例えば、構造から以下の行を削除します。

```
"metadata": {
  "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
  "created": "date",
  "updated": "date"
  }
```

ファイルを保存します。

6. 以下のようにパイプライン JSON ファイルを指定して、update-pipeline コマンドを実行します。

```
aws codepipeline update-pipeline --cli-input-json file://pipeline.json
```

このコマンドは、更新されたパイプラインの構造全体を返します。

> **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。

7. パイプラインの名前を指定して、start-pipeline-execution コマンドを実行します。これにより、ソースバケット内のアプリケーションの 2 回目の実行がパイプラインで実行されます。

```
aws codepipeline start-pipeline-execution --name MyFirstPipeline
```

このコマンドは pipelineExecutionId オブジェクトを返します。

8. CodePipeline コンソールを開き、パイプラインのリストから [MyFirstPipeline] を選択します。

パイプラインには、3 つのステージがあり、それらの各ステージのアーティファクトの状態が示されます。パイプラインがすべてのステージを実行するまでに最大 5 分かかることがあります。前回と同じように、最初の 2 つのステージではデプロイが成功しますが、[Production] ステージでは [Deploy-Second-Deployment] アクションが失敗したことが示されます。

9. [Deploy-Second-Deployment] アクションで、[Details] を選択すると、その失敗の詳細が表示されます。CodeDeploy デプロイの詳細ページにリダイレクトされます。この場合、最初のインス

タンスグループがすべての EC2 インスタンスにデプロイされ、2 番目のデプロイグループ用のインスタンスが残っていないために失敗しています。

> **Note**
>
> この失敗は、パイプラインのステージにエラーがある場合にどうなるかを示すために、意図的に起こしたものです。

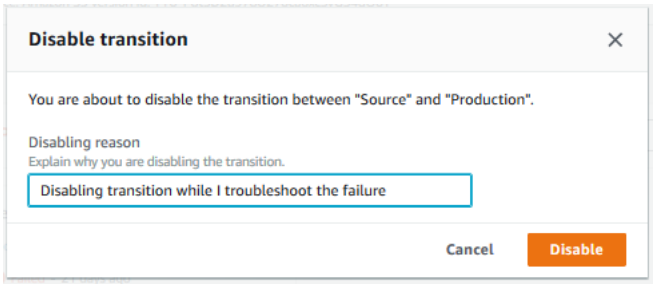
(オプション) ステップ 6: CodePipeline でステージ間の移行を有効または無効にする

パイプラインのステージ間の移行を有効化または無効化することができます。ステージ間の移行を無効にすると、ステージ間の移行を手動で制御できるようになります。たとえば、パイプラインの最初の 2 つのステージを実行するが、本番環境にデプロイする準備ができるまで、または問題のトラブルシューティング中か、そのステージが失敗するまで、3 番目のステージへの移行を無効化します。

CodePipeline パイプラインのステージ間の移行を無効/有効にするには

1. CodePipeline コンソールを開き、パイプラインのリストから [MyFirstPipeline] を選択します。
2. パイプラインの詳細ページで、2 番目のステージ (Deploy) と前のセクションで追加した 3 番目のステージ (Production) との間で [移行を無効にする] ボタンを選択します。
3. [移行を無効にする] で、ステージ間の移行を無効にする理由を入力し、[無効化] を選択します。

   ステージ間の矢印では、アイコンと色の変化、および、[移行を有効にする] ボタンが表示されます。



4. サンプルをもう一度 S3 バケットにアップロードします。バケットのバージョニングが有効になっているため、この変更によってパイプラインが開始します。

5. パイプラインの詳細ページに戻り、ステージの状態を監視します。パイプラインビューでは、最初の 2 つのステージで進行状況が示されて成功に変わりますが、3 番目のステージで変更はありません。このプロセスには数分かかることがあります。
6. 2 つのステージの間の [移行を有効にする] ボタンを選択して、遷移を有効にします。[Enable transition] ダイアログボックスで、[Enable] を選択します。3 番目のステージの実行は数分で開始し、パイプラインの最初の 2 つのステージですでに実行されているアーティファクトの処理を試みます。

> **Note**
>
> この 3 番目のステージを成功させるには、遷移を有効にする前に CodePipelineProductionFleet デプロイグループを編集し、アプリケーションのデプロイ先として別の一連の EC2 インスタンスを指定します。そのための方法の詳細については、「デプロイグループの設定を変更する」を参照してください。追加の EC2 インスタンスを作成すると、追加のコストが発生する場合があります。

ステップ 7: リソースをクリーンアップする

チュートリアル: 4 ステージのパイプラインを作成する 用にこのチュートリアルで作成したリソースの一部を使用することができます。たとえば、CodeDeploy アプリケーションおよびデプロイメントは再利用できます。クラウド上の完全マネージド型のビルドサービスである、CodeBuild などのプロバイダを 使用してビルドアクションを設定できます。また、Jenkins など、ビルドサーバーまたはシステムと備えたプロバイダを使用するビルドアクションを設定することもできます。

ただし、これらのチュートリアルの完了後、これらのリソースに対する継続利用料金が発生しないよう、使用したパイプラインおよびリソースを削除する必要があります。まずパイプラインを削除し、次に CodeDeploy アプリケーションおよびそれに関連付けられている Amazon EC2 インスタンスを削除します。最後に S3 バケットを削除します。

このチュートリアルで使用されているリソースをクリーンアップするには

1. CodePipeline リソースをクリーンアップするには、「AWS CodePipelineでパイプラインを削除する」の手順に従います。
2. CodeDeploy リソースをクリーンアップするには、「リソースをクリーンアップするには (コンソール)」の手順に従います。

3. S3 バケットを削除するには、「バケットを削除するか空にする」の手順に従います。追加のパイプラインを作成しない場合は、パイプラインのアーティファクトの保存用に作成した S3 バケットを削除します。このバケットの詳細については、「CodePipeline の概念」を参照してください。

チュートリアル: シンプルなパイプラインを作成する (CodeCommit リポジトリ)

このチュートリアルでは、CodePipeline を使用して、CodeCommit リポジトリに保持されているコードを 1 つの Amazon EC2 インスタンスにデプロイします。CodeCommit リポジトリに変更をプッシュすると、パイプラインがトリガーされます。パイプラインは、デプロイサービスとして CodeDeploy を使用して Amazon EC2 インスタンスに変更をデプロイします。

> **Important**
>
> パイプライン作成の一環として、CodePipeline は、ユーザーが指定した S3 アーティファクトバケットをアーティファクトとして使用します (これは S3 ソースアクションで使用するバケットとは異なります)。S3 アーティファクトバケットがパイプラインのアカウントとは異なるアカウントにある場合は、S3 アーティファクトバケットが によって所有 AWS アカウント されており、安全で信頼できることを確認してください。

パイプラインには 2 つのステージがあります。

- ソースステージ (ソース) を CodeCommit ソースアクションに指定します。
- デプロイステージ (デプロイ) を CodeDeploy デプロイアクションで指定します。

の使用を開始する最も簡単な方法は、CodePipeline コンソールでパイプラインの作成ウィザードを使用すること AWS CodePipeline です。

> **Note**
>
> 開始する前に、CodeCommit で Git クライアントを使用するようにセットアップされていることを確認します。手順については、CodeCommit のセットアップ を参照してください。

ステップ 1: CodeCommit リポジトリを作成する

まず、CodeCommit でリポジトリを作成します。パイプラインを実行すると、このリポジトリからソースコードが取得されます。また、CodeCommit リポジトリにプッシュする前に、コードを管理および更新するローカルリポジトリも作成します。

CodeCommit リポジトリを作成するには

1. https://console.aws.amazon.com/codecommit/ で CodeCommit コンソールを開きます。
2. リージョンセレクタで、リポジトリとパイプライン AWS リージョン を作成する を選択します。詳細については、「AWS リージョン およびエンドポイント」を参照してください。
3. [Repositories (リポジトリ)] ページで、[Create repository (リポジトリの作成)] を選択します。
4. [リポジトリの作成] ページの [リポジトリ名] に、新しいリポジトリの名前を入力します (例: **MyDemoRepo**)。
5. [作成] を選択します。

> **Note**
>
> このチュートリアルの残りのステップでは、CodeCommit リポジトリの名前として **MyDemoRepo** を使用します。別の名前を選択した場合は、このチュートリアル全体でそれを使用してください。

ローカルリポジトリをセットアップするには

このステップでは、ローカルリポジトリをセットアップしてリモート CodeCommit リポジトリに接続します。

> **Note**
>
> ローカルリポジトリを設定する必要はありません。ステップ 2: CodeCommit リポジトリにサンプルコードを追加する の説明に従って、コンソールを使用してファイルをアップロードすることもできます。

1. コンソールで新しいリポジトリを開き、ページの右上にある [URL のクローンを作成] を選択してから、[SSH のクローンを作成] を選択します。Git リポジトリのクローンを作成するアドレスがクリップボードにコピーされます。
2. ターミナルまたはコマンドラインで、ローカルリポジトリを保存するローカルディレクトリに移動します。このチュートリアルでは、/tmp を使用します。
3. 次のコマンドを実行してリポジトリをクローンし、SSH アドレスを前のステップでコピーしたものに置き換えます。このコマンドは、MyDemoRepo という名前のディレクトリを作成します。サンプルアプリケーションをこのディレクトリにコピーします。

```
git clone ssh://git-codecommit.us-west-2.amazonaws.com/v1/repos/MyDemoRepo
```

ステップ 2: CodeCommit リポジトリにサンプルコードを追加する

このステップでは、CodeDeploy サンプルチュートリアル用に作成したサンプルアプリケーションのコードをダウンロードし、CodeCommit リポジトリに追加します。

1. 次に、サンプル をダウンロードし、ローカルコンピュータのフォルダまたはディレクトリに保存します。

   a. 次のいずれかを選択します 。Linux インスタンスについて、このチュートリアルのステップに従う場合は、SampleApp_Linux.zip を選択します。

   - CodeDeploy を使用して Amazon Linux インスタンスにデプロイする場合は、サンプルアプリケーションを SampleApp_Linux.zip からダウンロードします。
   - CodeDeploy を使用して Windows Server インスタンスにデプロイする場合は、サンプルアプリケーションを SampleApp_Windows.zip からダウンロードします。

   サンプルアプリケーションには、CodeDeploy を使用してデプロイするための以下のファイルが含まれています。

   - appspec.yml - 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 形式のファイルです。AppSpec ファイルの詳細については、AWS CodeDeploy ユーザーガイドの「CodeDeploy AppSpec ファイルリファレンス」を参照してください。
   - index.html - インデックスファイルには、デプロイされたサンプルアプリケーションのホームページが含まれています。

- LICENSE.txt - ライセンスファイルには、サンプルアプリケーションのライセンス情報が含まれています。
- スクリプトのファイル - サンプルアプリケーションはスクリプトを使用して、インスタンス上の場所にテキストファイルを書き込みます。以下のように、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。
  - (Linux サンプルのみ) scripts フォルダ - このフォルダに入っているのはシェルスクリプト install_dependencies、start_server、stop_server です。依存関係をインストールし、自動デプロイのサンプルアプリケーションを起動および停止するために使用されます。
  - (Windows サンプルのみ) before-install.bat - BeforeInstall デプロイライフサイクルイベントのバッチスクリプトです。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場所をインスタンス上に作成するために実行されます。

b. 圧縮 (zip) ファイルをダウンロードします。

2. SampleApp_Linux.zip から先ほど作成したローカルディレクトリ (例: /tmp/MyDemoRepo や c:\temp\MyDemoRepo) にファイルを解凍します。

   それらのファイルはローカルリポジトリに直接配置してください。SampleApp_Linux フォルダーは含めないでください。例えば、ローカルの Linux、macOS、Unix マシンでは、ディレクトリとファイルの階層は次のようになります。

```
/tmp
   #-- MyDemoRepo
       #-- appspec.yml
       #-- index.html
       #-- LICENSE.txt
       #-- scripts
           #-- install_dependencies
           #-- start_server
           #-- stop_server
```

3. リポジトリにファイルをアップロードするには、次のいずれかの方法を使用します。

   a. CodeCommit コンソールを使用してファイルをアップロードするには:

      i. CodeCommit コンソールを開き、リポジトリ リストから自分のリポジトリを選択します。

      ii. [Add file]、[Upload file] の順に選択します。

iii. [ファイルの選択] を選択し、ファイルを参照します。フォルダにファイルを追加するには、ファイルの作成 を選択してから、scripts/install_dependencies のようなファイル名でフォルダ名を入力します。ファイルの内容を新しいファイルに貼り付けます。

ユーザー名とメールアドレスを入力して、変更をコミットします。

[Commit changes] (変更のコミット) を選択します。

iv. ファイルごとにこの手順を繰り返します。

リポジトリの内容は次のようになります。

```
#-- appspec.yml
#-- index.html
#-- LICENSE.txt
#-- scripts
    #-- install_dependencies
    #-- start_server
    #-- stop_server
```

b. git コマンドを使用してファイルをアップロードするには:

i. ディレクトリをローカルリポジトリに変更する:

```
(For Linux, macOS, or Unix) cd /tmp/MyDemoRepo
(For Windows) cd c:\temp\MyDemoRepo
```

ii. 以下のコマンドを実行して、すべてのファイルを一度にステージングします。

```
git add -A
```

iii. 以下のコマンドを実行して、コミットメッセージによりファイルをコミットします。

```
git commit -m "Add sample application files"
```

iv. 以下のコマンドを実行して、ローカルリポジトリから CodeCommit リポジトリにファイルをプッシュします。

```
git push
```

4. ダウンロードしてローカルリポジトリに追加したファイルは、CodeCommit `main` リポジトリの `MyDemoRepo` ブランチに追加され、パイプラインに含める準備ができています。

ステップ 3: Amazon EC2 Linux インスタンスを作成して CodeDeploy エージェントをインストールする

このステップでは、サンプルアプリケーションをデプロイする先の Amazon EC2 インスタンスを作成します。このプロセスの一環として、インスタンス上での CodeDeploy エージェントのインストールと管理を許可するインスタンスロールを作成します。CodeDeploy エージェントは、CodeDeploy デプロイでインスタンスを使用できるようにするソフトウェアパッケージです。また、CodeDeploy エージェントによってアプリケーションのデプロイに使用されるファイルを取得すること、SSM によって管理されることを、インスタンスに許可するポリシーをアタッチします。

インスタンス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. コンソールダッシュボードで [ロール] を選択します。
3. [ロールの作成] を選択してください。
4. [信頼されたエンティティのタイプを選択] で、[AWS のサービス] を選択します。[ユースケースの選択] で、[EC2] を選択します。[Select your use case (ユースケースを選択)] で、[EC2] を選択します。[Next: Permissions] (次のステップ: 許可) を選択します。
5. **`AmazonEC2RoleforAWSCodeDeploy`** という名前のマネージドポリシーを検索して選択します。
6. **`AmazonSSMManagedInstanceCore`** という名前のマネージドポリシーを検索して選択します。[Next: Tags (次へ: タグ)] を選択します。
7. [次へ: レビュー] を選択します。ロールの名前を入力します (例: **`EC2InstanceRole`**)。

   > **Note**
   >
   > 次のステップのロール名をメモしておきます。このロールは、インスタンスの作成時に選択します。

   [ロールの作成] を選択してください。

インスタンスを起動するには

1. Amazon EC2 コンソールの https://console.aws.amazon.com/ec2/ を開いてください。
2. サイドナビゲーションから [インスタンス] を選択し、ページの上部から [インスタンスの起動] を選択します。
3. [名前] に「**MyCodePipelineDemo**」と入力します。これにより、インスタンスにはキーが **Name** で、値が **MyCodePipelineDemo** というタグがが割り当てられます。後で、このインスタンスにサンプルアプリケーションをデプロイする CodeDeploy アプリケーションを作成します。CodeDeploy は、タグに基づいてデプロイするインスタンスを選択します。
4. アプリケーションイメージと OS イメージ (Amazon マシンイメージ) で、 AWS ロゴが付いた Amazon Linux AMI オプションを見つけ、選択されていることを確認します。(この AMI は Amazon Linux 2 AMI (HVM) と表記され、「無料利用枠対象」と表示されています。)
5. [インスタンスタイプ] で、インスタンスのハードウェア構成として無料利用枠対象となる `t2.micro` タイプを選択します。
6. [キーペア (ログイン)] で、キーペアを選択するか作成します。

   [キーペアなしで続行] を選択することもできます。

   > **Note**
   >
   > このチュートリアルでは、キーペアを使用せずに続行できます。SSH を使用してインスタンスに接続するには、キーペアを作成または使用します。

7. [ネットワーク設定] で、次の操作を行います。

   [パブリック IP の自動割り当て] で、ステータスが [有効] になっていることを確認します。

   作成したセキュリティグループで、[HTTP] を選択肢、[ソースタイプ] で [マイ IP] を選択します。
8. [Advanced Details] (高度な詳細) を展開します。[IAM インスタンスプロファイル] で、前の手順で作成した IAM ロール (**EC2InstanceRole** など) を選択します。
9. [概要] の [インスタンス数] に 1 と入力します。
10. [インスタンスを起動] を選択します。
11. [インスタンス] ページで、起動のステータスを表示できます。インスタンスを起動すると、その初期状態は `pending` です。インスタンスを起動した後は、状態が `running` に変わり、パブ

リック DNS 名を受け取ります ([パブリック DNS] 列が表示されていない場合は、[表示/非表示] アイコンを選択してから、[パブリック DNS] を選択します)。

ステップ 4: CodeDeploy でアプリケーションを作成する

CodeDeploy では、アプリケーション は、デプロイするソフトウェアアプリケーションを含むリソースです。後で、このアプリケーションを CodePipeline とともに使用して、サンプルアプリケーションの Amazon EC2 インスタンスへのデプロイを自動化します。

最初に、CodeDeploy がデプロイを実行できるようにするロールを作成します。次に、CodeDeploy アプリケーションを作成します。

CodeDeploy サービスロールの作成するために

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. コンソールダッシュボードで [ロール] を選択します。
3. [ロールの作成] を選択してください。
4. [信頼されたエンティティを選択] で、[AWS のサービス] を選択します。[ユースケース] で、[CodeDeploy] を選択します。示されたオプションから [CodeDeploy] を選択します。[次へ] を選択します。`AWSCodeDeployRole` マネージドポリシーはロールにアタッチ済みです。
5. [次へ] を選択します。
6. ロールの名前 (例: **`CodeDeployRole`**) を入力し、[ロールの作成] を選択します。

CodeDeploy でアプリケーションを作成するには

1. 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] ページが表示されない場合は、メニューで [アプリケーション] を選択します。
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、「**`MyDemoApplication`**」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。

CodeDeploy でデプロイグループを作成するには

デプロイグループは、デプロイ先のインスタンスやデプロイの速度など、デプロイ関連の設定を定義するリソースです。

1. アプリケーションが表示されるページで、[Create deployment group (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「**MyDemoDeploymentGroup**」と入力します。
3. [サービスロール] で、先ほど作成したサービスロールの ARN を選択します (**arn:aws:iam::*account_ID*:role/CodeDeployRole** など)。
4. [Deployment type] (デプロイタイプ) で、[In-place] (インプレース) を選択します。
5. [環境設定] で、[Amazon EC2 インスタンス] を選択します。キー フィールドに **Name** と入力します。値 フィールドに、(**MyCodePipelineDemo** のような) インスタンスのタグ付けに使用した名前を入力します。
6. AWS Systems Manager を使用したエージェント設定で、今すぐ を選択し、更新をスケジュールします。これにより、インスタンスにエージェントがインストールされます。Linux インスタンスは既に SSM エージェントで設定されており、これから CodeDeploy エージェントで更新されます。
7. [デプロイ設定] で、[CodeDeployDefault.OneAtaTime] を選択します。
8. ロードバランサー で、ロードバランシングの有効化 が選択されていないことを確認します。この例では、ロードバランサーを設定したり、ターゲットグループを選択したりする必要はありません。
9. デプロイグループの作成 を選択します。

ステップ 5: CodePipeline で最初のパイプラインを作成する

これで、最初のパイプラインを作成および実行する準備ができました。このステップでは、コードが CodeCommit リポジトリにプッシュされたときに自動的に実行されるパイプラインを作成します。

CodePipeline でパイプラインを作成するには

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

   CodePipeline コンソール (https://console.aws.amazon.com/codepipeline/) を開きます。

2. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。
3. [ステップ 1: 作成オプションを選択する] ページの [作成オプション] で、[カスタムパイプラインを構築する] オプションを選択します。[次へ] を選択します。
4. [ステップ 2: パイプラインの設定を選択する] で、[パイプライン名] に「**MyFirstPipeline**」と入力します。
5. CodePipeline は、特徴と料金が異なる V1 タイプと V2 タイプのパイプラインを提供しています。V2 タイプは、コンソールで選択できる唯一のタイプです。詳細については、「パイプラインタイプ」を参照してください。CodePipeline の料金については、料金を参照してください。
6. サービスロール で、新しいサービスロール を選択して、CodePipeline が IAM でサービスロールを作成できるようにします。
7. [詳細設定] をデフォルト設定のままにし、[次へ] を選択します。
8. [ステップ 3: ソースステージの追加] の [ソースプロバイダー] で、[CodeCommit] を選択します。リポジトリ名 で、ステップ 1: CodeCommit リポジトリを作成する で作成した CodeCommit リポジトリの名前を選択します。[ブランチ名] で、[main] を選択し、[次のステップ] を選択します。

   リポジトリ名とブランチを選択すると、このパイプライン用に作成される Amazon CloudWatch Events ルールがメッセージに表示されます。

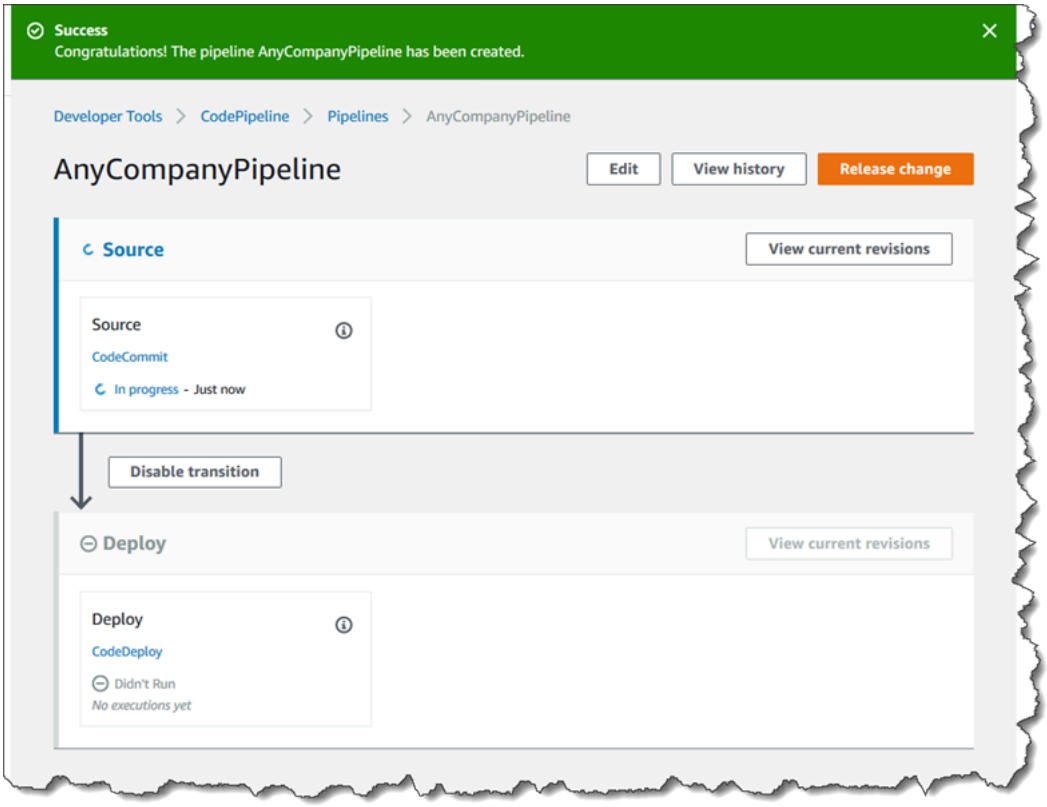
   [Change detection options] で、デフォルト値のままにします。これにより、CodePipelineは Amazon CloudWatch Events を使用して、ソースリポジトリの変更を検出できます。

   [次へ] を選択します。
9. [ステップ 4: ビルドステージの追加] で、[ビルドステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。[次へ] を選択します。

   > **Note**
   >
   > このチュートリアルでは、ビルドサービスを必要としないコードをデプロイするため、このステップは省略できます。ただし、インスタンスにデプロイする前にソースコードを構築する必要がある場合は、このステップで CodeBuild を設定できます。
10. [ステップ 5: テストステージの追加] で、[テストステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

    [次へ] を選択します。

11. [ステップ 6: デプロイステージの追加] の [デプロイプロバイダー] で、[CodeDeploy] を選択します。[アプリケーション名] に、「**MyDemoApplication**」を選択します。[デプロイグループ] で、[**MyDemoDeploymentGroup**]、[次のステップ] の順に選択します。
12. ステップ 7: 情報を確認してから、パイプラインの作成を選択します。
13. パイプラインは、作成後に実行を開始します。CodeCommit リポジトリからコードをダウンロードし、EC2 インスタンスへの CodeDeploy デプロイを作成します。CodePipeline サンプルが CodeDeploy デプロイで Amazon EC2 インスタンスにウェブページをデプロイするときに、進行状況と成功および失敗のメッセージを表示できます。



お疲れ様でした。CodePipeline で単純なパイプラインを作成しました。

次に、結果を確認します。

パイプラインが正常に実行されたことを確認するには

1. パイプラインの最初の進行状況を表示します。各ステージのステータスは、[まだ実行はありません] から [進行中] に変わり、その後、[Succeeded (成功)] または [Failed (失敗)] のいずれかに変わります。パイプラインの最初の実行は数分で完了します。

2. パイプラインのステータスが [成功] と表示されたら、[デプロイ] ステージのステータス領域で [CodeDeploy] を選択します。これにより、CodeDeploy コンソールが開きます。Succeeded が表示されない場合は、「」を参照してくださいCodePipeline のトラブルシューティング。
3. [Deployments (デプロイ)] タブで、デプロイ ID を選択します。デプロイのページの [Deployment lifecycle events (デプロイライフサイクルイベント)] で、インスタンス ID を選択します。これにより、EC2 コンソールが開きます。
4. [説明] タブの [パブリック DNS] でアドレス (例: `ec2-192-0-2-1.us-west-2.compute.amazonaws.com`) をコピーし、ウェブブラウザのアドレスバーに貼り付けます。

   ダウンロードして CodeCommit リポジトリにプッシュしたサンプルアプリケーションのウェブページが表示されます。

ステージ、アクション、パイプラインの仕組みの詳細については、「CodePipeline の概念 」を参照してください。

ステップ 6: CodeCommit リポジトリ内のコードを変更する

CodeCommit リポジトリのコードが変更されるとパイプラインが実行されるように設定されています。このステップでは、CodeCommit リポジトリ内のサンプル CodeDeploy アプリケーションのパートである HTML ファイルに変更を加えます。これらの変更をプッシュすると、パイプラインが再度実行され、変更内容は先ほどアクセスしたウェブアドレスに表示されます。

1. ディレクトリをローカルリポジトリに変更する:

```
(For Linux, macOS, or Unix) cd /tmp/MyDemoRepo
(For Windows) cd c:\temp\MyDemoRepo
```

2. 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。

```
(For Linux or Unix)gedit index.html
(For OS X)open -e index.html
(For Windows)notepad index.html
```

3. `index.html` ファイルのコンテンツを変更して、背景色およびウェブページのテキストの一部を変更してから、ファイルを保存します。

```
<!DOCTYPE html>
<html>
```

```
<head>
  <title>Updated Sample Deployment</title>
  <style>
    body {
      color: #000000;
      background-color: #CCFFCC;
      font-family: Arial, sans-serif;
      font-size:14px;
    }

    h1 {
      font-size: 250%;
      font-weight: normal;
      margin-bottom: 0;
    }

    h2 {
      font-size: 175%;
      font-weight: normal;
      margin-bottom: 0;
    }
  </style>
</head>
<body>
  <div align="center"><h1>Updated Sample Deployment</h1></div>
  <div align="center"><h2>This application was updated using CodePipeline,
 CodeCommit, and CodeDeploy.</h2></div>
  <div align="center">
    <p>Learn more:</p>
    <p><a href="https://docs.aws.amazon.com/codepipeline/latest/
userguide/">CodePipeline User Guide</a></p>
    <p><a href="https://docs.aws.amazon.com/codecommit/latest/
userguide/">CodeCommit User Guide</a></p>
    <p><a href="https://docs.aws.amazon.com/codedeploy/latest/
userguide/">CodeDeploy User Guide</a></p>
  </div>
</body>
</html>
```

4. 次のコマンドを一度に 1 つずつ実行して、変更をコミットして CodeCommit リポジトリにプッシュします。

```
git commit -am "Updated sample application files"
```

```
git push
```

パイプラインが正常に実行されたことを確認するには

1. パイプラインの最初の進行状況を表示します。各ステージのステータスは、[まだ実行はありません] から [進行中] に変わり、その後、[Succeeded (成功)] または [Failed (失敗)] のいずれかに変わります。パイプラインの実行は数分以内に完了します。
2. アクションステータスが [成功] と表示されたら、ブラウザで先ほどアクセスしたデモページを更新します。

   更新されたウェブページが表示されます。

ステップ 7: リソースをクリーンアップする

このガイドの他のチュートリアルでは、このチュートリアルで作成したリソースの一部を使用できます。例えば、CodeDeploy アプリケーションを再利用してデプロイできます。ただし、これらのチュートリアルの完了後、これらのリソースに対する継続利用料金が発生しないよう、使用したパイプラインおよびリソースを削除する必要があります。最初にパイプラインを削除し、次に CodeDeploy アプリケーションとそれに関連する Amazon EC2 インスタンスを削除し、最後に CodeCommit リポジトリを削除します。

このチュートリアルで使用されているリソースをクリーンアップするには

1. CodePipeline リソースをクリーンアップするには、「AWS CodePipelineでパイプラインを削除する」の手順に従います。
2. CodeDeploy リソースをクリーンアップするには、「チュートリアルのデプロイリソースのクリーンアップ」の手順に従います。
3. CodeCommit リポジトリを削除するには、「CodeCommit リポジトリの削除」の手順に従います。

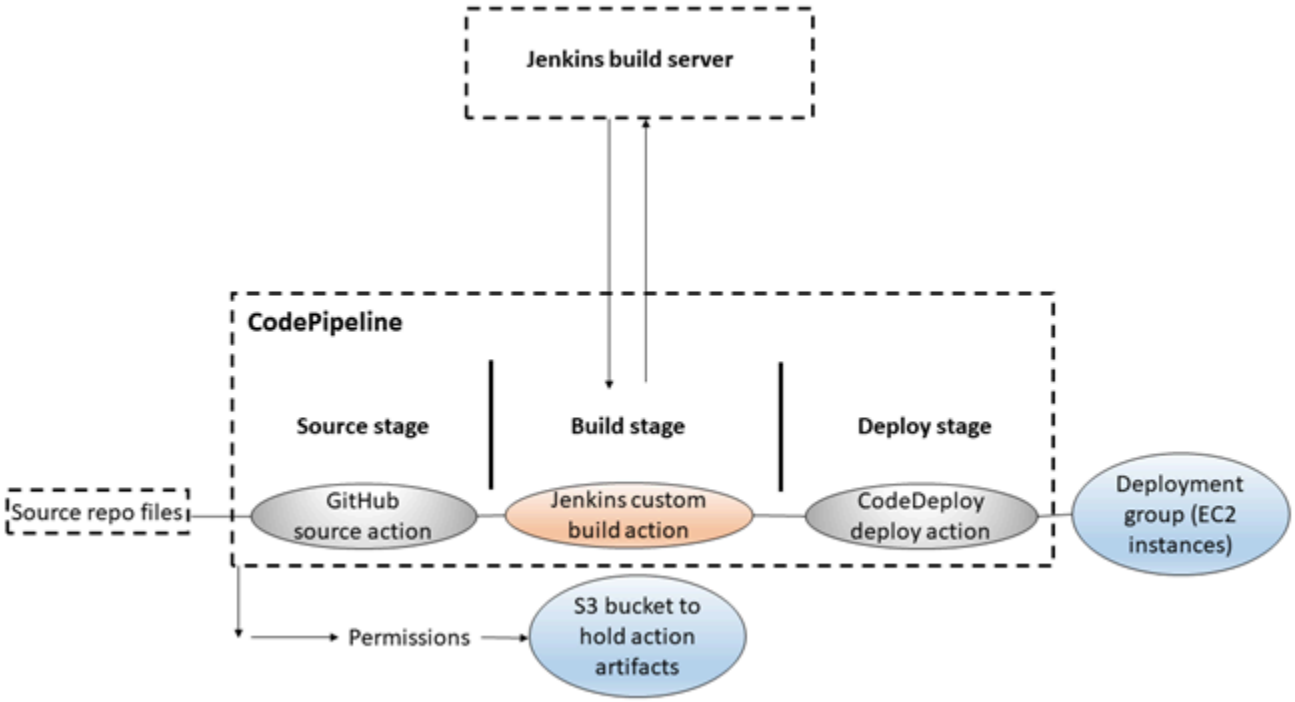
ステップ 8: 詳細情報

CodePipeline の仕組みの詳細：

- ステージ、アクション、パイプラインの仕組みの詳細については、「CodePipeline の概念 」を参照してください。
- CodePipeline を使用して実行できるアクションの詳細については、CodePipeline アクションタイプとの統合 を参照してください。
- この詳細なチュートリアル「チュートリアル: 4 ステージのパイプラインを作成する」をお試しください。デプロイ前にコードをビルドする手順を含むマルチステージパイプラインが作成されます。

チュートリアル: 4 ステージのパイプラインを作成する

これで、「チュートリアル: シンプルなパイプラインを作成する (S3 バケット)」または「チュートリアル: シンプルなパイプラインを作成する (CodeCommit リポジトリ)」に最初のパイプラインが作成されたため、より複雑なパイプラインを作成できるようになりました。このチュートリアルでは、4 ステージパイプラインを作成する方法について説明します。このパイプラインでは、ソースとして GitHub リポジトリを使用し、プロジェクトをビルドするための Jenkins ビルドサーバーおよびビルドしたコードをステージングサーバーにデプロイするための CodeDeploy アプリケーションを使用します。以下の図は初期の 3 ステージのパイプラインを示しています。



パイプラインが作成されたら、これを編集して、テストアクションを含むステージを追加してコードをテストします。この際、Jenkins も使用します。

このパイプラインを作成する前に、必要なリソースを設定する必要があります。例えば、ソースコードに GitHub リポジトリを使用する場合は、パイプラインに追加する前にリポジトリを作成する必要があります。このチュートリアルでは、セットアップの一部として EC2 インスタンスに Jenkins を設定する方法をデモ目的で示します。

> **Important**
>
> この手順でパイプラインに追加するアクションの多くには、パイプラインを作成する前に作成する必要がある AWS リソースが含まれます。ソースアクションの AWS リソースは常に、パイプラインを作成するのと同じ AWS リージョンで作成する必要があります。例えば、米国東部 (オハイオ) リージョンにパイプラインを作成している場合、CodeCommit リポジトリは米国東部 (オハイオ) リージョンにある必要があります。
> パイプラインの作成時にクロスリージョンアクションを追加できます。クロスリージョンアクションの AWS リソースは、アクションを実行する予定のリージョンと同じ AWS リージョンに存在する必要があります。詳細については、「CodePipeline でクロスリージョンアクションを追加する」を参照してください。

> **Important**
>
> パイプライン作成の一環として、CodePipeline は、ユーザーが指定した S3 アーティファクトバケットをアーティファクトとして使用します (これは S3 ソースアクションで使用するバケットとは異なります)。S3 アーティファクトバケットがパイプラインのアカウントとは異なるアカウントにある場合は、S3 アーティファクトバケットが によって所有 AWS アカウント されており、安全で信頼できることを確認してください。

このチュートリアルを開始するには、「CodePipeline の使用開始」の一般的な前提条件を満たしている必要があります。

トピック

- ステップ 1: の前提条件を満たす
- ステップ 2: CodePipeline でパイプラインを作成する
- ステップ 3: パイプラインに別のステージを追加する
- ステップ 4: リソースをクリーンアップする

ステップ 1: の前提条件を満たす

を Jenkins と統合するには、CodePipeline で使用する Jenkins のインスタンスに CodePipeline Plugin for Jenkins をインストール AWS CodePipeline する必要があります。また、専用の IAM ユーザーまたはロールを設定して、Jenkins プロジェクトと CodePipeline の間でアクセス許可を使用する必要があります。Jenkins と CodePipeline を統合する最も簡単な方法としては、Jenkins の統合用に作成した IAM インスタンスロールを使用する EC2 インスタンスに Jenkins をインストールします。Jenkins アクション用のパイプラインのリンクを正常に接続するには、Jenkins プロジェクトで使用するポートへのインバウンド接続を許可するように、サーバーまたは EC2 インスタンスのプロキシおよびファイアウォール設定を構成する必要があります。これらのポートに接続する前に、Jenkins でユーザーを認証してアクセス制御するように設定されていることを確認します (HTTPS 接続のみ使用できるように Jenkins のセキュリティを確保するには 443 および 8443、HTTP 接続できるようにするには 80 および 8080)。詳細については、「Jenkins のセキュリティ確保」を参照してください。

> **Note**
>
> このチュートリアルでは、コードサンプルを使用して、Haml から HTML に変換するビルドステップを設定します。GitHub リポジトリからオープンソースのサンプルコードをダウンロードするには、「サンプルのコピーまたはクローンを GitHub リポジトリに作成する」のステップに従います。GitHub リポジトリ内の .zip ファイルだけではなく、sample 全体が必要です。
>
> このチュートリアルでは、以下を前提としています。
>
> - Jenkins のインストールと管理および Jenkins プロジェクトの作成に慣れている
> - Ruby の Rake と Haml gem が、同一コンピュータ、または Jenkins プロジェクトをホストするインスタンス上にインストールされていること。
> - Rake コマンドを端末またはコマンドラインから実行できるように、必要なシステム環境変数が設定されていること (例えば、Windows システムで、Rake をインストールしたディレクトリが追加されるように PATH 変数を変更する)。

トピック

- サンプルのコピーまたはクローンを GitHub リポジトリに作成する
- Jenkins 統合に使用する IAM ロールを作成する
- Jenkins および Jenkins 用 CodePipeline プラグインのインストールと設定

サンプルのコピーまたはクローンを GitHub リポジトリに作成する

サンプルを複製して GitHub リポジトリにプッシュするには

1. サンプルコードを GitHub リポジトリからダウンロードするか、リポジトリをローカルコンピュータに複製します。2 つのサンプルパッケージがあります。
   - サンプルを Amazon Linux、RHEL、または Ubuntu Server インスタンスにデプロイする場合は、[codepipeline-jenkins-aws-codedeploy_linux.zip] を選択します。
   - サンプルを Windows Server インスタンスにデプロイする場合は、[CodePipeline-Jenkins-AWSCodeDeploy_Windows.zip] を選択します。
2. リポジトリから Fork を選択して、サンプルリポジトリを GitHub アカウントのリポジトリにクローンします。詳細については、GitHub のドキュメントを参照してください。

Jenkins 統合に使用する IAM ロールを作成する

ベストプラクティスとして、EC2 インスタンスを起動して Jenkins サーバーをホストし、IAM ロールを使用して CodePipeline とのやり取りに必要なアクセス許可をインスタンスに付与することを検討します。

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/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]、[ロールを作成する] の順に選択します。
3. [Select type of trusted entity] (信頼されたエンティティの種類を選択) で、[AWS のサービス] を選択します。[Choose the service that will use this role (このロールを使用するサービスを選択)] で、[EC2] を選択します。[Select your use case (ユースケースを選択)] で、[EC2] を選択します。
4. [Next: Permissions] (次のステップ: 許可) を選択します。[Attach permissions policies (アクセス許可ポリシーをアタッチする)] ページで、`AWSCodePipelineCustomActionAccess` 管理ポリシーを選択し、次に、[Next: Tags (次の手順: タグ)] を選択します。[次へ: レビュー] を選択します。
5. [確認] ページの [ロール名] に、Jenkins の統合専用として作成するロールの名前 (*`JenkinsAccess`*など) を入力し、[ロールの作成] を選択します。

Jenkins をインストールする先の EC2 インスタンスを作成する場合は、「ステップ 3: インスタンスの詳細を設定する」で、インスタンスロール (*`JenkinsAccess`*など) を必ず選択します。

インスタンスロールおよび Amazon EC2 の詳細については、「Amazon EC2 の IAM ロール」、「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用してアクセス許可を付与する」、「AWS の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。

Jenkins および Jenkins 用 CodePipeline プラグインのインストールと設定

Jenkins および Jenkins 用 CodePipeline プラグインのインストールをするには、

1. Jenkins をインストールする先の EC2 インスタンスを作成し、「ステップ 3: インスタンスの詳細を設定する」で、作成したインスタンスロール (*JenkinsAccess*など) を必ず選択します。EC2 インスタンス作成の詳細については、「Amazon EC2 ユーザーガイド」の 「Amazon EC2 インスタンスの起動」を参照してください。

   > **Note**
   >
   > 使用する Jenkins リソースがすでにある場合は、特別な IAM ユーザーを作成し、そのユーザーに `AWSCodePipelineCustomActionAccess` 管理ポリシーを適用してから、Jenkins リソースに対してそのユーザーのアクセス認証情報を設定して使用する必要があります。Jenkins UI を使用して認証情報を指定する場合は、HTTPS のみを許可するように Jenkins を設定します。詳細については、「CodePipeline のトラブルシューティング」を参照してください。

2. EC2 インスタンスに Jenkins をインストールします。詳細については、Jenkins のドキュメントの「Jenkins のインストール」と「Jenkins の開始とアクセス」のほか、「CodePipeline との製品とサービスの統合」の「details of integration with Jenkins」を参照してください。
3. Jenkins を起動し、ホーム ページで [Manage Jenkins] (Jenkins の管理) を選択します。
4. [Jenkins の管理] ページで、[プラグインの管理] を選択します。
5. [Available] タブを選択し、[Filter] 検索ボックスに「**`AWS CodePipeline`**」と入力します。リストから [CodePipeline Plugin for Jenkins] を選択し、次に [ダウンロードして再起動後にインストール] を選択します。
6. [プラグイン/アップグレードのインストール] ページで、[インストール完了後、実行中のジョブがなければ Jenkins を再起動する] を選択します。
7. [ダッシュボードに戻る] を選択します。
8. メインページで、[New Item] (新しい項目) を選択します。
9. [Item Name] に、Jenkins プロジェクトの名前 (*`MyDemoProject`* など) を入力します。[Freestyle project] (フリースタイルプロジェクト)、[OK] の順に選択します。

> **Note**
>
> プロジェクトの名前が CodePipeline の要件を満たしていることを確認します。詳細については、「AWS CodePipeline のクォータ」を参照してください。

10. プロジェクトの設定ページで、[Execute concurrent builds if necessary] (必要な場合に複数のビルドを並列実行する) チェックボックスをオンにします。[ソースコードの管理] で、[AWS CodePipeline] を選択します。EC2 インスタンスに Jenkins をインストールし、CodePipeline と Jenkins の統合用に作成した IAM ユーザーのプロファイル AWS CLI で を設定した場合は、他のすべてのフィールドを空のままにします。
11. [Advanced (詳細)] を選択し、[プロバイダ] に、CodePipeline に表示されるアクションのプロバイダーの名前 (*MyJenkinsProviderName*など) を入力します。この名前が一意で覚えやすいものであることを確認します。このチュートリアルの後半でパイプラインにビルドアクションを追加するときと、テストアクションを追加するときに使用します。

> **Note**
>
> このアクション名は、CodePipeline のアクションの命名要件を満たしている必要があります。詳細については、「AWS CodePipeline のクォータ」を参照してください。

12. [Build Triggers] (トリガーのビルド) で、チェックボックスをすべてオフにし、[Poll SCM] (SCM のポーリング) を選択します。[Schedule] に、以下のようにスペースで区切ってアスタリスクを 5 つ入力します。

```
* * * * *
```

これは 1 分ごとに CodePipeline をポーリングします。

13. [ビルド] で、[Add build step] (ビルドステップの追加) を選択します。[シェルの実行] (Amazon Linux、RHEL、または Ubuntu Server) [バッチコマンドの実行] (Windows Server) を選択し、次のように入力します。

```
rake
```

> **Note**
>
> rake の実行に必要な変数と設定が環境で定義されていることを確認します。定義されていないと、ビルドは失敗します。

14. [Add post-build action]、[AWS CodePipeline Publisher] の順に選択します。[Add] を選択し、[Build Output Locations] でこの場所は空白のままにします。この設定はデフォルトです。ビルドプロセスの最後に圧縮ファイルが作成されます。
15. [保存] を選択して、Jenkins プロジェクトを保存します。

ステップ 2: CodePipeline でパイプラインを作成する

チュートリアルのこの部分では、[Create Pipeline] ウィザードを使用してパイプラインを作成します。

CodePipeline 自動リリースプロセスを作成するには

1. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. 必要に応じて、リージョンセレクターを使用し、パイプラインリソースの配置先のリージョンに切り替えます。例えば、前のチュートリアルでリソースを `us-east-2` に作成した場合は、リージョンセレクターを必ず米国東部 (オハイオ) に設定します。

   CodePipeline で使用できるリージョンとエンドポイントの詳細については、「AWS CodePipeline エンドポイントとクォータ」を参照してください。
3. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。
4. [ステップ 1: 作成オプションを選択する] ページの [作成オプション] で、[カスタムパイプラインを構築する] オプションを選択します。[次へ] を選択します。
5. [ステップ 2: パイプラインの設定を選択する] ページで、[パイプライン名] にパイプラインの名前を入力します。
6. CodePipeline は、特徴と料金が異なる V1 タイプと V2 タイプのパイプラインを提供しています。V2 タイプは、コンソールで選択できる唯一のタイプです。詳細については、「パイプラインタイプ」を参照してください。CodePipeline の料金については、料金を参照してください。

7. [サービスロール] で、[新しいサービスロール] を選択して、CodePipeline の IAM でのサービスロールの作成を許可します。
8. [詳細設定] をデフォルト設定のままにし、[次へ] を選択します。
9. [ステップ 3: ソースステージの追加] ページの [ソースプロバイダー] で、[GitHub] を選択します。
10. 接続 で、既存の接続を選択するか、新規の接続を作成します。GitHub ソースアクション用の接続を作成または管理するには、「GitHub コネクション」を参照してください。
11. [ステップ 4: ビルドステージ を追加する] で、[Jenkins の追加] を選択します。[プロバイダー名]で、Jenkins の CodePipeline プラグインで指定したアクションの名前 (*MyJenkinsProviderName*など) を入力します。この名前は、Jenkins用の CodePipeline プラグインの名前と正確に一致する必要があります。[サーバー URL] に、Jenkins がインストールされている EC2 インスタンスの URL を入力します。[プロジェクト名] に、Jenkins で作成したプロジェクトの名前 (*MyDemoProject* など) を入力し、[次へ] を選択します。
12. [ステップ 5: テストステージの追加] で、[テストステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

    [次へ] を選択します。
13. [ステップ 6: デプロイステージの追加] では、チュートリアル: シンプルなパイプラインを作成する (S3 バケット) で作成した CodeDeploy アプリケーションおよびデプロイグループを再利用します。[プロバイダをデプロイする]で、[CodeDeploy] を選択します。[アプリケーション名] に「**CodePipelineDemoApplication**」と入力するか、更新ボタンを選択してリストからアプリケーション名を選択します。[デプロイグループ] に「**CodePipelineDemoFleet**」と入力するか、リストからデプロイグループを選択して [次へ] を選択します。

    > **Note**
    >
    > 独自の CodeDeploy リソースを使用することも、新しいリソースを作成することもできますが、追加のコストが発生する場合があります。
14. ステップ 7: 情報を確認してから、パイプラインの作成を選択します。
15. パイプラインが自動的に開始され、パイプラインによりサンプルが実行されます。パイプラインが Haml サンプルを HTML にビルドし、CodeDeploy デプロイの各 Amazon EC2 インスタンスにウェブページとしてデプロイすると、進行状況、成功、失敗のメッセージを表示できます。

ステップ 3: パイプラインに別のステージを追加する

次に、テストステージ、テストアクションの順で、サンプルに含まれている Jenkins テストを使用するステージに追加し、ウェブページにコンテンツが含まれているかどうかを確認します。このテストは、デモンストレーションのみを目的としています。

> **Note**
>
> 他のステージをパイプラインに追加しない場合は、パイプラインのステージ (Staging) へテストアクションを追加します。その前後にデプロイアクションを行います。

パイプラインにテストステージを追加する

トピック

- インスタンスの IP アドレスを検索する
- デプロイのテスト用に Jenkins プロジェクトを作成する
- 4 番目のステージを作成する

インスタンスの IP アドレスを検索する

コードをデプロイしたインスタンスの IP アドレスを確認するには

1. パイプラインのステータスが "Succeeded" と表示されたら、[Staging] ステージのステータス領域で [詳細] を選択します。
2. [デプロイの詳細] (デプロイの詳細) セクションの [インスタンス ID] で、正常にデプロイされたいずれかのインスタンスの ID を選択します。
3. インスタンスの IP アドレス (*192.168.0.4* など) をコピーします。この IP アドレスは Jenkins テストで使用します。

デプロイのテスト用に Jenkins プロジェクトを作成する

Jenkins プロジェクトを作成するには

1. Jenkins をインストールしたインスタンスで、Jenkins を開き、メインページから [New Item] (新しい項目) を選択します。

2. [Item Name] に、Jenkins プロジェクトの名前 (*MyTestProject* など) を入力します。[Freestyle project] (フリースタイルプロジェクト)、[OK] の順に選択します。

> **Note**
>
> プロジェクトの名前が CodePipeline の要件を満たしていることを確認します。詳細については、「AWS CodePipeline のクォータ」を参照してください。

3. プロジェクトの設定ページで、[Execute concurrent builds if necessary] (必要な場合に複数のビルドを並列実行する) チェックボックスをオンにします。[ソースコードの管理] で、[AWS CodePipeline] を選択します。EC2 インスタンスに Jenkins をインストールし、CodePipeline と Jenkins の統合用に作成した IAM ユーザーのプロファイル AWS CLI で を設定した場合は、他のすべてのフィールドを空のままにします。

> **Important**
>
> Jenkins プロジェクトを設定していて、それが Amazon EC2 インスタンスにインストールされていないか、Windows オペレーティングシステムを実行している EC2 インスタンスにインストールされている場合は、プロキシホストとポートの設定に従ってフィールドに入力し、Jenkins と CodePipeline の統合用に設定した IAM ユーザーまたはロールの認証情報を指定します。

4. [詳細設定] を選択してから、[カテゴリ] で [テスト] を選択します。
5. [プロバイダ] に、ビルドプロジェクトで使用したのと同じ名前 (*MyJenkinsProviderName* など) を入力します。この名前は、このチュートリアルの後半でパイプラインにテストアクションを追加するときに使用します。

> **Note**
>
> この名前は、CodePipeline のアクションの命名要件を満たしている必要があります。詳細については、「AWS CodePipeline のクォータ」を参照してください。

6. [Build Triggers] (トリガーのビルド) で、チェックボックスをすべてオフにし、[Poll SCM] (SCM のポーリング) を選択します。[Schedule] に、以下のようにスペースで区切ってアスタリスクを 5 つ入力します。

```
* * * * *
```

これは 1 分ごとに CodePipeline をポーリングします。

7. [ビルド] で、[Add build step] (ビルドステップの追加) を選択します。Amazon Linux、RHEL、または Ubuntu Server インスタンスにデプロイする場合は、[シェルの実行] を選択します。次に、以下のように入力します。IP アドレスは、先ほどコピーした EC2 インスタンスのアドレスです。

```
TEST_IP_ADDRESS=192.168.0.4 rake test
```

Windows Server インスタンスにデプロイする場合は、[Execute batch command (バッチコマンドの実行)] を選択し、以下のように入力します。ここで、IP アドレスは、先ほどコピーした EC2 インスタンスのアドレスです。

```
set TEST_IP_ADDRESS=192.168.0.4 rake test
```

> **Note**
>
> このテストでは、デフォルトのポート 80 を想定しています。別のポートを指定する場合は、以下のように test port ステートメントを追加します。
>
> ```
> TEST_IP_ADDRESS=192.168.0.4 TEST_PORT=8000 rake test
> ```

8. [Add post-build action]、[AWS CodePipeline Publisher] の順に選択します。[追加] は選択しないでください。
9. [保存] を選択して、Jenkins プロジェクトを保存します。

4 番目のステージを作成する

Jenkins テストアクションを含むステージをパイプラインに追加するには

1. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [名前] で、作成したパイプラインの名前を選択します。
3. パイプライン詳細ページで、[編集] を選択します。
4. [編集] ページで [+ Stage (+ ステージの追加)] を選択して、ビルドステージの直後にステージを追加します。

5. 新しいステージの名前フィールドに名前 (**Testing** など) を入力し、[+ Add action group (+ アクショングループの追加)] を選択します。
6. [アクション名] に「*MyJenkinsTest-Action*」と入力します。[テストプロバイダ] で、Jenkins で指定したプロバイダ名 (*MyJenkinsProviderName* など) を選択します。[プロジェクト名] に、Jenkins で作成したプロジェクトの名前 (*MyTestProject* など) を入力します。[入力アーティファクト] で、デフォルト名が *BuildArtifact* の Jenkins ビルドからアーティファクトを選択し、[完了] を選択します。

   > **Note**
   >
   > Jenkins テストアクションは Jenkins ビルドステップで構築されたアプリケーションで動作するため、テストアクションへの入力アーティファクトのビルドアーティファクトを使用します。

   入力アーティファクトと出力アーティファクト、およびパイプラインの構造の詳細については、「CodePipeline パイプライン構造リファレンス」を参照してください。
7. [編集] ページで、[パイプラインの変更を保存] を選択します。[パイプラインの変更を保存] ダイアログボックスで、[保存して続行] を選択します。
8. パイプラインに新しいステージが追加されましたが、パイプラインの別の実行をトリガーした変更がないため、そのステージのステータスは [まだ実行はありません] と表示されます。修正したパイプラインによりサンプルを実行するには、パイプラインの詳細ページで [リリースの変更] を選択します。

   パイプラインビューには、パイプラインのステージとアクション、それらの 4 つのステージを実行しているリビジョンの状態が表示されます。パイプラインがすべてのステージを実行するのにかかる時間は、アーティファクトのサイズ、ビルドとテストのアクションの複雑さ、その他の要因によって異なります。

ステップ 4: リソースをクリーンアップする

これらのチュートリアルが完了したら、使用したパイプラインおよびリソースを削除する必要があるため、このリソースに対する継続利用料金がかかることはありません。今後、CodePipeline を使用しない場合は、パイプラインを削除してから、CodeDeploy アプリケーション、関連付けられている Amazon EC2 インスタンス、アーティファクトの保存に使用した Amazon S3 バケットの順に削除し

ます。今後使用しない場合は、GitHub リポジトリなどの他のリソースを削除するかどうかを検討することも必要です。

このチュートリアルで使用されているリソースをクリーンアップするには

1. ローカル Linux、macOS または Unix マシンでターミナルセッションを開くか、ローカル Windows マシンでコマンドプロンプトを開き、delete-pipeline コマンドを実行して、作成したパイプラインを削除します。**MySecondPipeline** の場合は、次のコマンドを入力します。

   ```
   aws codepipeline delete-pipeline --name "MySecondPipeline"
   ```

   このコマンドは何も返しません。
2. CodeDeploy リソースをクリーンアップするには、「クリーンアップ」の手順に従います。
3. インスタンスリソースをクリーンアップするには、Jenkins をインストールした EC2 インスタンスを削除します。詳細については、「インスタンスのクリーンアップ」を参照してください。
4. 追加のパイプラインを作成したり、CodePipeline を再利用したりしない場合は、パイプラインのアーティファクトの保存に使用した Amazon S3 バケットを削除します。バケットを削除するには、「バケットの削除」の手順に従います。
5. このパイプラインに他のリソースを再利用しない場合は、それらのリソースを該当するガイダンスに従って削除することを検討してください。例えば、GitHub リポジトリを削除する場合は、GitHub ウェブサイトの「リポジトリの削除」の指示に従います。

チュートリアル: CloudWatch Events ルールをセットアップし、パイプラインの状態の変更の E メール通知を送信します。

でパイプラインを設定したら AWS CodePipeline、パイプラインの実行状態、またはパイプラインのステージやアクションに変更があるたびに通知を送信するように CloudWatch Events ルールを設定できます。CloudWatch Events を使用してパイプラインの状態の変更の通知をセットアップする方法の詳細は、CodePipeline イベントのモニタリング を参照してください。

このチュートリアルでは、パイプラインの状態が失敗に変わったら E メールを送信する通知を設定します。このチュートリアルでは、CloudWatch Events ルールを作成するときの入力変換方法を使用します。メッセージスキーマの詳細を変換し、人間が読み取れるテキストでメッセージを配信します。

> **Note**
>
> Amazon SNS 通知や CloudWatch Events ルールなど、このチュートリアルのリソースを作成するときは、リソースがパイプラインと同じ AWS リージョンに作成されていることを確認します。

トピック

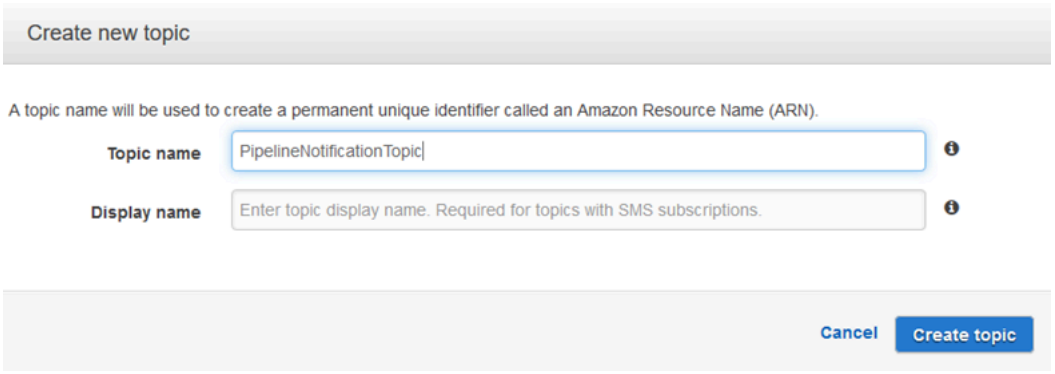
- ステップ 1 : Amazon SNS を使用して E メール通知をセットアップします。
- ステップ 2: ルールを作成し SNS トピックをターゲットとして追加する
- ステップ 3: リソースをクリーンアップする

ステップ 1 : Amazon SNS を使用して E メール通知をセットアップします。

Amazon SNS は、トピックの使用を調整して、サブスクライブしているエンドポイントやクライアントへのメッセージを配信します。Amazon SNS を使用して通知トピックを作成してから、E メールアドレスを使用してトピックをサブスクライブします。Amazon SNS トピックが CloudWatch Events ルールにターゲットとして追加されます。詳細については、「Amazon Simple Notification Service デベロッパーガイド」を参照してください。

Amazon SNS でトピックを作成または識別します。CodePipeline は CloudWatch Events を使用して、Amazon SNS を介してこのトピックに通知を送信します。トピックを作成するには:

1. Amazon SNS コンソール (https://console.aws.amazon.com/sns) を開きます。
2. [トピックの作成] を選択します。
3. [Create new topic (新しいトピックの作成)] ダイアログボックスの [Topic name (トピック名)] で、トピックの名前 (例: **`PipelineNotificationTopic`**) を入力します。



4. [トピックを作成] を選択します。

   詳細については、Amazon SNS デベロッパーガイド の トピックの作成 を参照してください。

1 つかそれ以上の受信者にトピックをサブスクライブさせ、E メール通知を受け取ります。受信者にトピックをサブスクライブさせるには:

1. Amazon SNS コンソールで、トピック リストから、新しいトピックの横にあるチェックボックスを選択します。[Actions, Subscribe to topic] を選択します。
2. [Create subscription] ダイアログボックスで、ARN が [Topic ARN] に表示されていることを確認します。
3. [Protocol (プロトコル)] として [Email (E メール)] を選択してください。
4. [Endpoint] に、新しい受信者の完全な E メールアドレスを入力します。
5. [Create Subscription] (サブスクリプションの作成) を選択します。
6. Amazon SNS は受信者にサブスクリプション確認の E メールを送信します。E メール通知を受信するには、受信者は、この E メールで [サブスクリプションを確認] リンクを選択する必要があります。受信者がリンクをクリックした後、正常にサブスクライブされたら、Amazon SNS により受信者のウェブブラウザに確認メッセージが表示されます。

   詳細については、Amazon SNS デベロッパーガイド の トピックのサブスクライブ を参照してください。

ステップ 2: ルールを作成し SNS トピックをターゲットとして追加する

CodePipeline でイベント出典として CloudWatch Events 通知ルールを作成します。

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の Events] を選択します。
3. Create rule を選択します。[イベントソース] で、[AWS CodePipeline] を選択します。[イベントタイプ] で、パイプライン実行の状態変更を選択します。
4. [特定の状態] を選択し、[**FAILED**] を選択します。
5. [編集] を選択し、[イベントパターンのプレビュー] ペインで JSON テキストエディタを開きます。**pipeline** パラメータを、次の例 (「myPipeline」という名前のパイプライン) に示すように、パイプラインの名前とともに追加します。

   ここでイベントパターンをコピーしてコンソールに貼り付けることができます。

```
{
  "source": [
    "aws.codepipeline"
  ],
  "detail-type": [
    "CodePipeline Pipeline Execution State Change"
  ],
  "detail": {
    "state": [
      "FAILED"
    ],
    "pipeline": [
      "myPipeline"
    ]
  }
}
```

6. [Targets] で、[Add target] を選択します。
7. ターゲットのリストで、[SNS トピック] を選択します。[トピック] に、作成したトピックを入力します。
8. [入力の設定] を展開して、[インプットトランスフォーマー] を閉じます。
9. [入力パス] ボックスに、次のキーと値のペアを入力します。

   ```
   { "pipeline" : "$.detail.pipeline" }
   ```

   [入力テンプレート] ボックスに、以下のように入力します。

   ```
   "The Pipeline <pipeline> has failed."
   ```

10. [詳細の設定] を選択します。
11. [ルールの詳細を設定する] ページで、名前とオプションの説明を入力します。[状態] では、[有効] ボックスをオンのままにします。
12. [ルールを作成] を選択します。
13. CodePipeline が構築通知を現在送信していることを確認します。たとえば、ビルド通知 E メールが受信トレイにあるかどうかを確認します。
14. ルールの動作を変更するには、CloudWatch コンソールで、ルールを選択してから、アクション、編集 の順に選択します。ルールを編集し、[詳細設定] を選択し、[Update] を選択します。

ルールがビルド通知を送信するのを停止するには、CloudWatch コンソールで、ルールを選択してから、アクション、無効化 を選択します。

ルールを削除するには、 CloudWatch コンソールで、ルールを選択してから、アクション、削除の順に選択します。

ステップ 3: リソースをクリーンアップする

これらのチュートリアルが完了したら、使用したパイプラインおよびリソースを削除する必要があるため、このリソースに対する継続利用料金がかかることはありません。

SNS 通知をクリーンアップして Amazon CloudWatch Events ルールを削除する方法については、クリーンアップ (Amazon SNSトピックからの退会) および Amazon CloudWatch Events API リファレンス の `DeleteRule` リファレンスを参照してください。

チュートリアル: を使用して Android アプリを構築およびテストするパイプラインを作成する AWS Device Farm

AWS CodePipeline を使用して、コミットがプッシュされるたびにアプリが構築およびテストされる継続的な統合フローを設定できます。このチュートリアルでは、GitHub リポジトリのソースコードを使って Android アプリをビルドしてテストするパイプラインを作成して設定する方法を説明します。パイプラインは新しい GitHub コミットの到着を検出し、CodeBuild を使用してアプリケーションを構築し、Device Farm を使用してテストします。

> **Important**
>
> コンソールでのパイプライン作成の一環として、CodePipeline は S3 アーティファクトバケットをアーティファクトとして使用します (これは S3 ソースアクションで使用するバケットとは異なります)。S3 アーティファクトバケットがパイプラインのアカウントとは異なるアカウントにある場合は、S3 アーティファクトバケットが によって所有 AWS アカウントされており、安全で信頼できることを確認してください。

> **Important**
>
> この手順でパイプラインに追加するアクションの多くには、パイプラインを作成する前に作成する必要がある AWS リソースが含まれます。ソースアクションの AWS リソースは常

に、パイプラインを作成するのと同じ AWS リージョンで作成する必要があります。例えば、米国東部 (オハイオ) リージョンにパイプラインを作成している場合、CodeCommit リポジトリは米国東部 (オハイオ) リージョンにある必要があります。
パイプラインの作成時にクロスリージョンアクションを追加できます。クロスリージョンアクションの AWS リソースは、アクションを実行する予定のリージョンと同じ AWS リージョンに存在する必要があります。詳細については、「CodePipeline でクロスリージョンアクションを追加する」を参照してください。

既存の Android アプリとテスト定義を使用してこれを試すか、 Device Farmが提供したサンプルアプリケーションとテスト定義 を使用できます。

[開始する前に]

1. AWS Device Farm コンソールにサインインし、新しいプロジェクトの作成を選択します。
2. プロジェクトを選択します。ブラウザで、新しいプロジェクトの URL をコピーします。URL には、プロジェクト ID が含まれます。
3. プロジェクト ID をコピーしてメモしておきます。CodePipeline でパイプラインを作成するときに、それを使用します。

   以下は、プロジェクトの URL の例です。プロジェクト ID を抽出するには、`projects/` 後の値をコピーします。この例では、プロジェクト ID は `eec4905f-98f8-40aa-9afc-4c1cfexample` です。

   ```
   https://<region-URL>/devicefarm/home?region=us-west-2#/projects/
   eec4905f-98f8-40aa-9afc-4c1cfexample/runs
   ```

Device Farm テストを使用するように CodePipeline を設定します。

1. アプリケーションのコードのルートに `buildspec.yml` という名前のファイルを追加してコミットし、リポジトリにプッシュします。CodeBuild は、このファイルを使用してコマンドを実行し、アプリケーションを構築するために必要なアーティファクトにアクセスします。

   ```
   version: 0.2

   phases:
     build:
   ```

```
    commands:
      - chmod +x ./gradlew
      - ./gradlew assembleDebug
artifacts:
  files:
     - './android/app/build/outputs/**/*.apk'
  discard-paths: yes
```

2. (オプション) Calabash または Appium を使用してアプリケーションをテストする場合は、テスト定義ファイルをリポジトリに追加します。後のステップで、定義を使用してテストスイートを実行するように Device Farm を設定できます。

   Device Farm の組み込みのテストを使用する場合は、このステップを省略できます。

3. パイプラインを作成してソースステージを追加するには、以下の手順を実行します。

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

   b. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。

   c. [ステップ 1: 作成オプションを選択する] ページの [作成オプション] で、[カスタムパイプラインを構築する] オプションを選択します。[次へ] を選択します。

   d. [ステップ 2: パイプラインの設定を選択する] ページで、[パイプライン名] にパイプラインの名前を入力します。

   e. CodePipeline は、特徴と料金が異なる V1 タイプと V2 タイプのパイプラインを提供しています。V2 タイプは、コンソールで選択できる唯一のタイプです。詳細については、「パイプラインタイプ」を参照してください。CodePipeline の料金については、料金を参照してください。

   f. [サービスロール] で、[New service role (新しいサービスロール)] は選択したままにして、[Role name (ロール名)] は変更しません。既存のサービスロール (ある場合) を使用することもできます。

   > **Note**
   >
   > 2018 年 7 月以前に作成した CodePipeline サービスロールを使用している場合、Device Farm の許可を追加する必要があります。これを行うには、IAM コンソールを開き、ロールを見つけて、ロールのポリシーに次の許可を追加します。詳細については、「CodePipeline サービスロールにアクセス許可を追加する」を参照してください。

```
{
     "Effect": "Allow",
     "Action": [
        "devicefarm:ListProjects",
        "devicefarm:ListDevicePools",
        "devicefarm:GetRun",
        "devicefarm:GetUpload",
        "devicefarm:CreateUpload",
        "devicefarm:ScheduleRun"
     ],
     "Resource": "*"
}
```

g. [詳細設定] をデフォルト設定のままにし、[次へ] を選択します。

h. [ステップ 3: ソースステージの追加] ページの [ソースプロバイダー] で、[GitHub (GitHub アプリ経由)] を選択します。

i. 接続 で、既存の接続を選択するか、新規の接続を作成します。GitHub ソースアクション用の接続を作成または管理するには、「GitHub コネクション」を参照してください。

j. [リポジトリ] で、ソースリポジトリを選択します。

k. [ブランチ] で、使用するブランチを選択します。

l. ソースアクションの残りの設定はデフォルトのままにします。[次へ] を選択します。

4. [ステップ 4: ビルドステージの追加] で、ビルドステージを追加します。

a. [プロバイダーを構築する] で、[その他のビルドプロバイダー]、[AWS CodeBuild] の順に選択します。[リージョン] をデフォルトでパイプラインのリージョンにすることを許可します。

b. [プロジェクトを作成] を選択します。

c. [プロジェクト名] に、このビルドプロジェクトの名前を入力します。

d. [環境イメージ] で、[Managed image (マネージド型イメージ)] を選択します。[Operating system] で、[Ubuntu] を選択します。

e. [ランタイム] で、[Standard (標準)] を選択します。[イメージ] で、[aws/codebuild/standard:5.0] を選択します。

CodeBuild は、Android Studio がインストールされているこの OS イメージを使用してアプリケーションを構築します。

f. サービスロール で、既存の CodeBuild サービスロールを選択するか、新しいサービスロールを作成します。

g. [ビルド仕様] で、[Use a buildspec file (ビルド仕様ファイルの使用)] を選択します。

h. [Continue to CodePipeline] (CodePipeline に進む) を選択します。これにより、CodePipeline コンソールに戻り、設定用にリポジトリ内の `buildspec.yml` を使用する CodeBuild プロジェクトが作成されます。ビルドプロジェクトでは、サービスロールを使用して AWS のサービス のアクセス許可を管理します。このステップには数分かかる場合があります。

i. [次へ] を選択します。

5. [ステップ 5: テストステージの追加] で、[テストステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

   [次へ] を選択します。

6. [ステップ 6: デプロイステージの追加] ページで、[デプロイステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。[次へ] を選択します。

7. ステップ 7: 確認で、パイプラインの作成を選択します。ソースとビルドステージを示す図が表示されます。

8. パイプラインに Device Farm テストアクションを追加します。

   a. 右上の [編集] を選択します。

   b. 図の最下部で [+ Add stage] (+ ステージの追加) を選択します。[ステージ名] に名前 (**`Test`** など) を入力します。

   c. [+ Add action group (+ アクションの追加)] を選択します。

   d. [アクション名] に名前を入力します。

   e. アクションプロバイダ で、AWS Device Farm を選択します。[リージョン] をデフォルトでパイプラインのリージョンにすることを許可します。

   f. [入力アーティファクト] で、テストステージに先立つステージの出力アーティファクトと一致する入力アーティファクト (`BuildArtifact` など) を選択します。

      AWS CodePipeline コンソールでは、パイプライン図の情報アイコンにカーソルを合わせると、各ステージの出力アーティファクトの名前を確認できます。パイプラインでアプリケーションを [ソース] ステージから直接テストする場合は、[SourceArtifact] を選択します。パイプラインに [ビルド] ステージが含まれている場合は、[BuildArtifact] を選択します。

   g. ProjectId に、Device Farm プロジェクト ID を入力します。このチュートリアルの最初の手順に従い、プロジェクト ID を取得します。

h. [DevicePoolArn] に、デバイスプールの ARN を入力します。上位デバイスの ARNs など、プロジェクトの使用可能なデバイスプール ARN を取得するには、 CLI AWS を使用して次のコマンドを入力します。

```
aws devicefarm list-device-pools --arn arn:aws:devicefarm:us-
west-2:account_ID:project:project_ID
```

i. [AppType] に、「Android」と入力します。

[AppType] の有効な値は次のとおりです。

- iOS
- Android
- Web

j. [デプロイ] に、コンパイルされたアプリケーションパッケージのパスを入力します。パスは、テストステージの入力アーティファクトのルートを基準とする相対パスです。このパスは `app-release.apk` に似ています。

k. TestType にテストのタイプを入力し、Test にテスト定義ファイルのパスを入力します。パスは、テストの入力アーティファクトのルートに関連します。

[TestType] の有効な値は次のとおりです。

- APPIUM_JAVA_JUNIT
- APPIUM_JAVA_TESTNG
- APPIUM_NODE
- APPIUM_RUBY
- APPIUM_PYTHON
- APPIUM_WEB_JAVA_JUNIT
- APPIUM_WEB_JAVA_TESTNG
- APPIUM_WEB_NODE
- APPIUM_WEB_RUBY
- APPIUM_WEB_PYTHON
- BUILTIN_FUZZZ
- INSTRUMENTATION
- XCTEST

- XCTEST_UI

> **Note**
>
> カスタム環境ノードはサポートされていません。

l. 残りのフィールドにはテストおよびアプリケーションタイプに適した構成を入力します。

m. (オプション) [アドバンスト] で、テストランの情報の設定を行います。

n. [保存] を選択します。

o. 編集中のステージで、[完了] を選択します。 AWS CodePipeline のペインで [保存] を選択し、警告メッセージで [保存] を選択します。

p. 変更を送信してパイプラインのビルドを開始するには、[変更をリリース]、[リリース] の順に選択します。

チュートリアル: で iOS アプリをテストするパイプラインを作成する AWS Device Farm

AWS CodePipeline を使用して、ソースバケットが変更されるたびにアプリをテストする継続的な統合フローを簡単に設定できます。このチュートリアルでは、S3 バケットからビルドした iOS アプリをテストするためのパイプラインを作成して設定する方法を示します。パイプラインは保存された変更の到着を Amazon CloudWatch Events を介して検出し、構築したアプリケーションをテストするために Device Farm を使用します。

> **Important**
>
> パイプライン作成の一環として、CodePipeline は、ユーザーが指定した S3 アーティファクトバケットをアーティファクトとして使用します (これは S3 ソースアクションで使用するバケットとは異なります)。S3 アーティファクトバケットがパイプラインのアカウントとは異なるアカウントにある場合は、S3 アーティファクトバケットが によって所有 AWS アカウント されており、安全で信頼できることを確認してください。

> **Important**
>
> この手順でパイプラインに追加するアクションの多くには、パイプラインを作成する前に作成する必要がある AWS リソースが含まれます。ソースアクションの AWS リソースは常に、パイプラインを作成するのと同じ AWS リージョンで作成する必要があります。例えば、米国東部 (オハイオ) リージョンにパイプラインを作成している場合、CodeCommit リポジトリは米国東部 (オハイオ) リージョンにある必要があります。
> パイプラインの作成時にクロスリージョンアクションを追加できます。クロスリージョンアクションの AWS リソースは、アクションを実行する予定のリージョンと同じ AWS リージョンに存在する必要があります。詳細については、「CodePipeline でクロスリージョンアクションを追加する」を参照してください。

既存の iOS アプリを使用して試してみるか、サンプル iOS アプリを使用できます。

[開始する前に]

1. AWS Device Farm コンソールにサインインし、新しいプロジェクトの作成を選択します。
2. プロジェクトを選択します。ブラウザで、新しいプロジェクトの URL をコピーします。URL には、プロジェクト ID が含まれます。
3. プロジェクト ID をコピーしてメモしておきます。CodePipeline でパイプラインを作成するときに、それを使用します。

   以下は、プロジェクトの URL の例です。プロジェクト ID を抽出するには、`projects/` 後の値をコピーします。この例では、プロジェクト ID は `eec4905f-98f8-40aa-9afc-4c1cfexample` です。

   ```
   https://<region-URL>/devicefarm/home?region=us-west-2#/projects/
   eec4905f-98f8-40aa-9afc-4c1cfexample/runs
   ```

Device Farm テスト(例 Amazon S3) を使用するように CodePipeline を設定する

1. バージョニングが有効になっている S3 バケットを作成するか、使用します。「ステップ 1: アプリケーションの S3 バケットを作成する」の手順に従って、S3 バケットを作成します。

2. バケットの Amazon S3 コンソールで、アップロード を選択し、指示に従って .zip ファイルをアップロードします。

   サンプルアプリケーションは、.zip ファイルにパッケージ化する必要があります。

3. パイプラインを作成してソースステージを追加するには、以下の手順を実行します。

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

   b. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。

   c. [ステップ 1: 作成オプションを選択する] ページの [作成オプション] で、[カスタムパイプラインを構築する] オプションを選択します。[次へ] を選択します。

   d. [ステップ 2: パイプラインの設定を選択する] ページで、[パイプライン名] にパイプラインの名前を入力します。

   e. CodePipeline は、特徴と料金が異なる V1 タイプと V2 タイプのパイプラインを提供しています。V2 タイプは、コンソールで選択できる唯一のタイプです。詳細については、「パイプラインタイプ」を参照してください。CodePipeline の料金については、料金を参照してください。

   f. [サービスロール] で、[New service role (新しいサービスロール)] は選択したままにして、[Role name (ロール名)] は変更しません。既存のサービスロール (ある場合) を使用することもできます。

   > **Note**
   >
   > 2018 年 7 月以前に作成した CodePipeline サービスロールを使用している場合、Device Farm に対する許可を追加する必要があります。これを行うには、IAM コンソールを開き、ロールを見つけて、ロールのポリシーに次の許可を追加します。詳細については、「CodePipeline サービスロールにアクセス許可を追加する」を参照してください。
   >
   > ```
   > {
   >     "Effect": "Allow",
   >     "Action": [
   >        "devicefarm:ListProjects",
   >        "devicefarm:ListDevicePools",
   >        "devicefarm:GetRun",
   >        "devicefarm:GetUpload",
   >        "devicefarm:CreateUpload",
   > ```

```
            "devicefarm:ScheduleRun"
        ],
        "Resource": "*"
    }
```

g. [詳細設定] をデフォルト設定のままにし、[次へ] を選択します。

h. [ステップ 3: ソースステージの追加] ページの [ソースプロバイダー] で、[Amazon S3] を選択します。

i. [Amazon S3 の場所] に、.zip ファイルのバケット (`my-storage-bucket` など) とオブジェクトキー (`s3-ios-test-1.zip` など) を入力します。

j. [次へ] を選択します。

4. [ステップ 4: ビルドステージの追加] で、パイプラインのプレースホルダービルドステージを作成します。これにより、ウィザードでパイプラインを作成することができます。ウィザードを使用して 2 ステージパイプラインを作成した後は、このプレースホルダービルドステージは不要になります。パイプラインが完了した後、この第 2 ステージが削除され、ステップ 5 で新しいテストステージが追加されます。

   a. [ビルドプロバイダ] で、[Jenkins の追加] を選択します。このビルド選択はプレースホルダーです。それは使用されていません。

   b. [プロバイダ名] に名前を入力します。名前はプレースホルダーです。それは使用されていません。

   c. [サーバー URL] にテキストを入力します。テキストはプレースホルダーです。それは使用されていません。

   d. [プロジェクト名] に名前を入力します。名前はプレースホルダーです。それは使用されていません。

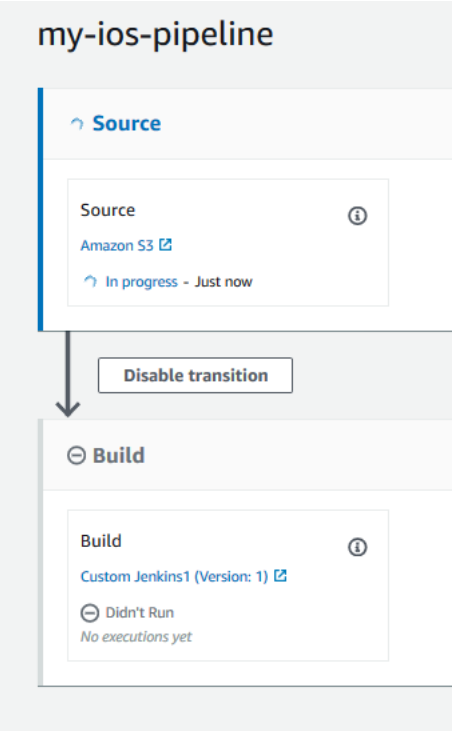
   e. [次へ] を選択します。

   f. [ステップ 5: テストステージの追加] で、[テストステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

      [次へ] を選択します。

   g. [ステップ 6: デプロイステージの追加] ページで、[デプロイステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

   h. ステップ 7: 確認で、パイプラインの作成を選択します。ソースとビルドステージを示す図が表示されます。



5. 次のようにして、Device Farm テストアクションをパイプラインに追加します。

   a. 右上の [編集] を選択します。

   b. [Edit stage (ステージの編集)] を選択します。[削除] を選択します。これにより、パイプライン作成のためには使用しないプレースホルダーステージが削除されます。

   c. 図の最下部で [+ Add stage] (+ ステージの追加) を選択します。

   d. [ステージ名] にステージ名 (Test など) を入力し、[Add stage (ステージの追加)] を選択します。

   e. [+ Add action group (+ アクションの追加)] を選択します。

   f. [アクション名] に、名前を入力します (DeviceFarmTest など)。

   g. アクションプロバイダ で、AWS Device Farm を選択します。[リージョン] をデフォルトでパイプラインのリージョンにすることを許可します。

   h. [入力アーティファクト] で、テストステージに先立つステージの出力アーティファクトと一致する入力アーティファクト (`SourceArtifact` など) を選択します。

      AWS CodePipeline コンソールでは、パイプライン図の情報アイコンにカーソルを合わせると、各ステージの出力アーティファクトの名前を確認できます。パイプラインでアプリケーションを [ソース] ステージから直接テストする場合は、[SourceArtifact] を選択します。パイプラインに [ビルド] ステージが含まれている場合は、[BuildArtifact] を選択します。

i. ProjectId で、Device Farm のプロジェクト ID を選択します。このチュートリアルの最初の手順に従い、プロジェクト ID を取得します。

j. [DevicePoolArn] に、デバイスプールの ARN を入力します。上位デバイスの ARNs など、プロジェクトの使用可能なデバイスプール ARN を取得するには、 CLI AWS を使用して次のコマンドを入力します。

```
aws devicefarm list-device-pools --arn arn:aws:devicefarm:us-
west-2:account_ID:project:project_ID
```

k. [AppType] に、「iOS」と入力します。

[AppType] の有効な値は次のとおりです。

- iOS
- Android
- Web

l. [デプロイ] に、コンパイルされたアプリケーションパッケージのパスを入力します。パスは、テストステージの入力アーティファクトのルートを基準とする相対パスです。このパスは `ios-test.ipa` に似ています。

m. TestType にテストのタイプを入力し、Test にテスト定義ファイルのパスを入力します。パスは、テストの入力アーティファクトのルートに関連します。

Device Farm の組み込みテストのいずれかを使用している場合は、BUILTIN_FUZZ など Device Farm プロジェクトに設定されたテストのタイプを入力します。[FuzzEventCount] に、時間をミリ秒単位で入力します (6000 など)。[FuzzEventThrottle] に、時間をミリ秒単位で入力します (50 など)。

Device Farm の組み込みテストのいずれも使用していない場合は、テストのタイプを入力し、テスト にテスト定義ファイルのパスを入力します。パスは、テストの入力アーティファクトのルートに関連します。

[TestType] の有効な値は次のとおりです。

- APPIUM_JAVA_JUNIT
- APPIUM_JAVA_TESTNG
- APPIUM_NODE
- APPIUM_RUBY

- APPIUM_PYTHON
- APPIUM_WEB_JAVA_JUNIT
- APPIUM_WEB_JAVA_TESTNG
- APPIUM_WEB_NODE
- APPIUM_WEB_RUBY
- APPIUM_WEB_PYTHON
- BUILTIN_FUZZZ
- INSTRUMENTATION
- XCTEST
- XCTEST_UI

> **Note**
> カスタム環境ノードはサポートされていません。

n. 残りのフィールドにはテストおよびアプリケーションタイプに適した構成を入力します。

o. (オプション) [アドバンスト] で、テストランの情報の設定を行います。

p. [保存] を選択します。

q. 編集中のステージで、[完了] を選択します。 AWS CodePipeline のペインで [保存] を選択し、警告メッセージで [保存] を選択します。

r. 変更を送信してパイプラインの実行を開始するには、[変更のリリース]、[リリース] の順に選択します。

チュートリアル: Service Catalog にデプロイするパイプラインを作成する

Service Catalog を使用すると、 AWS CloudFormation テンプレートに基づいて製品を作成およびプロビジョニングできます。

> **Important**
> パイプライン作成の一環として、CodePipeline は、ユーザーが指定した S3 アーティファクトバケットをアーティファクトとして使用します (これは S3 ソースアクションで使用するバ

ケットとは異なります)。S3 アーティファクトバケットがパイプラインのアカウントとは異なるアカウントにある場合は、S3 アーティファクトバケットが によって所有 AWS アカウント されており、安全で信頼できることを確認してください。

このチュートリアルでは、製品テンプレートを Service Catalog にデプロイするパイプラインを作成して設定し、(GitHub、CodeCommit、Amazon S3 で作成済みの) ソースリポジトリで行った変更を送信する方法を示します。

> **Note**
>
> Amazon S3 がパイプラインのソースプロバイダーである場合、すべてのソースファイルを 1 つの .zip ファイルとしてパッケージ化してバケットにアップロードする必要があります。それ以外の場合、ソースアクションは失敗します。

まず Service Catalog で製品を作成し、次に AWS CodePipelineでパイプラインを作成します。このチュートリアルでは、デプロイ設定を指定するための 2 つのオプションを取り上げます。

- Service Catalog で製品を作成し、テンプレートファイルをソースリポジトリにアップロードします。(個別の設定ファイルではなく) CodePipeline コンソールで製品バージョンとデプロイ設定を指定します。「オプション 1: 設定ファイルを使用しないで Service Catalog にデプロイする」を参照してください。

  > **Note**
  >
  > テンプレートファイルは YAML または JSON 形式で作成できます。

- Service Catalog で製品を作成し、テンプレートファイルをソースリポジトリにアップロードします。個別の設定ファイルを使用して製品バージョンとデプロイ設定を指定します。「オプション 2: 設定ファイルを使用して Service Catalog にデプロイする」を参照してください。

オプション 1: 設定ファイルを使用しないで Service Catalog にデプロイする

この例では、S3 バケットのサンプル AWS CloudFormation テンプレートファイルをアップロードし、Service Catalog で製品を作成します。次に、CodePipeline コンソールで、パイプラインを作成し、デプロイ設定を指定します。

ステップ 1: サンプルテンプレートファイルをソースリポジトリにアップロードする

1. テキストエディタを開きます。以下のコードをファイルに貼り付けて、サンプルテンプレートを作成します。S3_template.json という名前でファイルを保存します。

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "CloudFormation Sample Template S3_Bucket: Sample template showing
 how to create a privately accessible S3 bucket. **WARNING** This template creates
 an S3 bucket. You will be billed for the resources used if you create a stack from
 this template.",
  "Resources": {
    "S3Bucket": {
      "Type": "AWS::S3::Bucket",
      "Properties": {}
    }
  },
  "Outputs": {
    "BucketName": {
      "Value": {
        "Ref": "S3Bucket"
      },
      "Description": "Name of Amazon S3 bucket to hold website content"
    }
  }
}
```

   このテンプレートにより AWS CloudFormation 、 は Service Catalog で使用できる S3 バケットを作成できます。

2. S3_template.json ファイルを AWS CodeCommit リポジトリにアップロードします。

ステップ 2: Service Catalog で製品を作成する

1. IT 管理者として、Service Catalog コンソールにサインインし、[製品] ページに移動して、[新しい製品のアップロード] を選択します。
2. [新しい製品のアップロード] ページで、以下の手順を実行します。
   a. [製品名] に、新しい製品に使用する名前を入力します。
   b. [Description (説明)] に製品カタログの説明を入力します。この説明は、製品リストでユーザーが正しい製品を選択できるように表示されます。
   c. [提供元] に IT 部門または管理者の名前を入力します。
   d. [次へ] を選択します。
3. (オプション) [サポート詳細の入力] に製品サポートの連絡先情報を入力し、[次へ] を選択します。
4. [バージョンの詳細] に以下の情報を入力します。
   a. [Upload a template file (テンプレートファイルをアップロード)] を選択します。`S3_template.json` ファイルを見つけ、アップロードします。
   b. [バージョンタイトル] に、製品バージョンの名前 (**`devops S3 v2`** など) を入力します。
   c. [Description (説明)] に、このバージョンと他のバージョンを区別するための詳細を入力します。
   d. [次へ] を選択します。
5. [確認] ページで、情報が正しいことを確認し、[作成] を選択します。
6. ブラウザの [製品] ページで、新しい製品の URL をコピーします。これには製品 ID が含まれています。この製品 ID をコピーして保持します。CodePipeline でパイプラインを作成するときに、それを使用します。

   以下に示しているのは、`my-product` という製品の URL です。製品 ID を抽出するには、等号 (=) とアンパサンド (&) との間の値をコピーします。この例では、製品 ID は `prod-example123456` です。

```
https://<region-URL>/servicecatalog/home?region=<region>#/admin-products?
productCreated=prod-example123456&createdProductTitle=my-product
```

> **Note**
>
> ページから移動する前に、製品の URL をコピーします。このページから移動したら、CLI を使用して製品 ID を取得する必要があります。

数秒後、製品が [製品] ページに表示されます。製品をリストに表示するには、ブラウザの更新が必要になる場合があります。

ステップ 3: パイプラインを作成する

1. パイプラインに名前を付け、パイプラインのパラメータを選択するには、以下の手順を実行します。

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

   b. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。

   c. [ステップ 1: 作成オプションを選択する] ページの [作成オプション] で、[カスタムパイプラインを構築する] オプションを選択します。[次へ] を選択します。

   d. [ステップ 2: パイプラインの設定を選択する] ページで、[パイプライン名] にパイプラインの名前を入力します。

   e. CodePipeline は、特徴と料金が異なる V1 タイプと V2 タイプのパイプラインを提供しています。V2 タイプは、コンソールで選択できる唯一のタイプです。詳細については、「パイプラインタイプ」を参照してください。CodePipeline の料金については、料金を参照してください。

   f. サービスロール で、新しいサービスロール を選択して、CodePipeline が IAM でサービスロールを作成できるようにします。

   g. [詳細設定] をデフォルト設定のままにし、[次へ] を選択します。

2. [ステップ 3: ソースステージの追加] ページにソースステージを追加するには、以下を行います。

   a. [ソースプロバイダ] で、AWS CodeCommit を選択します。

   b. [リポジトリ名] と [ブランチ名] に、ソースアクションに使用するリポジトリとブランチを入力します。

c. [次へ] を選択します。

3. [ステップ 4: ビルドステージの追加] で、[ビルドステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。
4. [ステップ 5: テストステージの追加] で、[テストステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

   [次へ] を選択します。
5. [ステップ 6: デプロイステージの追加] で、以下の手順を実行します。

   a. [デプロイプロバイダ] で、[AWS Service Catalog] を選択します。

   b. デプロイ設定で、[Enter deployment configuration (デプロイ設定の入力)] を選択します。

   c. [プロダクト ID] に、Service Catalog コンソールからコピーしたプロダクト ID を貼り付けます。

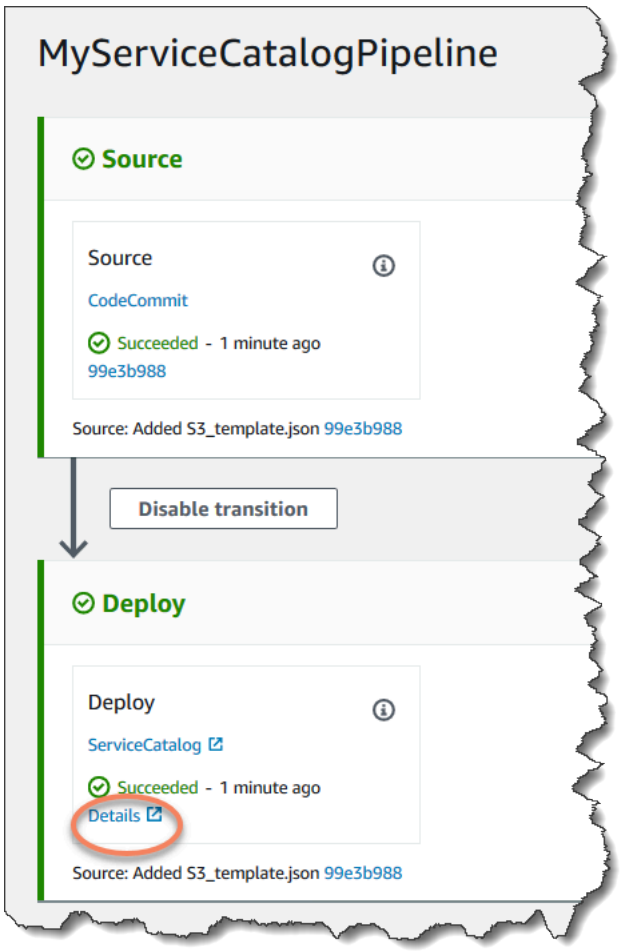
   d. [Template file path (テンプレートファイルパス)] に、テンプレートファイルが保存されている相対パスを入力します。

   e. [製品タイプ] で、[CloudFormation テンプレート] を選択します。

   f. [製品バージョン名] に、Service Catalog で指定した製品バージョンの名前を入力します。テンプレートの変更を新しい製品バージョンにデプロイする場合は、同じ製品の以前の製品バージョンで使用されていない製品バージョン名を入力します。

   g. [Input artifact (入力アーティファクト)] で、ソース入力アーティファクトを選択します。

   h. [次へ] を選択します。
6. [ステップ 7: レビュー] で、パイプライン設定を確認し、[作成] を選択します。
7. パイプラインが正常に実行されたら、デプロイステージで [Details (詳細)] を選択します。これにより、Service Catalog で製品が開きます。



8. 製品情報で、バージョン名を選択して製品テンプレートを開きます。テンプレートのデプロイを表示します。

ステップ 4: 変更をプッシュして Service Catalog で製品を確認する

1. CodePipeline コンソールでパイプラインを表示し、ソースステージで [詳細] を選択します。コンソールでソース AWS CodeCommit リポジトリが開きます。[Edit (編集)] を選択し、ファイルの内容 (説明など) を変更します。

```
"Description": "Name of Amazon S3 bucket to hold and version website content"
```

2. 変更をコミットし、プッシュします。変更をプッシュした後、パイプラインが開始されます。パイプラインの実行が完了したら、デプロイステージで [詳細] を選択して、製品を Service Catalog で開きます。
3. 製品情報で、新しいバージョン名を選択して製品テンプレートを開きます。デプロイされたテンプレートの変更を表示します。

オプション 2: 設定ファイルを使用して Service Catalog にデプロイする

この例では、S3 バケットのサンプル AWS CloudFormation テンプレートファイルをアップロードし、Service Catalog で製品を作成します。デプロイ設定を指定する個別の設定ファイルもアップロードします。次に、パイプラインを作成し、設定ファイルの場所を指定します。

ステップ 1: サンプルテンプレートファイルをソースリポジトリにアップロードする

1. テキストエディタを開きます。以下のコードをファイルに貼り付けて、サンプルテンプレートを作成します。S3_template.json という名前でファイルを保存します。

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "CloudFormation Sample Template S3_Bucket: Sample template showing
 how to create a privately accessible S3 bucket. **WARNING** This template creates
 an S3 bucket. You will be billed for the resources used if you create a stack from
 this template.",
  "Resources": {
    "S3Bucket": {
      "Type": "AWS::S3::Bucket",
      "Properties": {}
    }
  },
  "Outputs": {
    "BucketName": {
      "Value": {
        "Ref": "S3Bucket"
      },
      "Description": "Name of Amazon S3 bucket to hold website content"
    }
  }
}
```

   このテンプレートにより AWS CloudFormation 、 は Service Catalog で使用できる S3 バケットを作成できます。

2. S3_template.json ファイルを AWS CodeCommit リポジトリにアップロードします。

ステップ 2: 製品デプロイ設定ファイルを作成する

1. テキストエディタを開きます。製品の設定ファイルを作成します。設定ファイルは、Service Catalog デプロイパラメータ/設定を定義するために使用されます。パイプラインを作成するときに、このファイルを使用します。

   このサンプルでは、ProductVersionName を「devops S3 v2」、ProductVersionDescription を MyProductVersionDescription としています。テンプレートの変更を新しい製品バージョンにデプロイする場合は、同じ製品の以前の製品バージョンで使用されていない製品バージョン名を入力するだけです。

   sample_config.json という名前でファイルを保存します。

   ```
   {
       "SchemaVersion": "1.0",
       "ProductVersionName": "devops S3 v2",
       "ProductVersionDescription": "MyProductVersionDescription",
       "ProductType": "CLOUD_FORMATION_TEMPLATE",
       "Properties": {
           "TemplateFilePath": "/S3_template.json"
       }
   }
   ```

   このファイルにより、パイプラインが実行されるたびに製品バージョン情報が作成されます。

2. sample_config.json ファイルを AWS CodeCommit リポジトリにアップロードします。必ずこのファイルはソースリポジトリにアップロードしてください。

ステップ 3: Service Catalog で製品を作成する

1. IT 管理者として、Service Catalog コンソールにサインインし、[製品] ページに移動して、[新しい製品のアップロード] を選択します。
2. [新しい製品のアップロード] ページで、以下の手順を実行します。
   a. [製品名] に、新しい製品に使用する名前を入力します。
   b. [Description (説明)] に製品カタログの説明を入力します。この説明は製品リストに表示されて、ユーザーが正しい製品を選択するのに役立ちます。
   c. [提供元] に IT 部門または管理者の名前を入力します。
   d. [次へ] を選択します。

3. (オプション) [サポート詳細の入力] に、製品サポートの連絡先情報を入力し、[次へ] を選択します。
4. [バージョンの詳細] に以下の情報を入力します。
   a. [Upload a template file (テンプレートファイルをアップロード)] を選択します。`S3_template.json` ファイルを見つけ、アップロードします。
   b. [バージョンタイトル] に製品バージョンの名前 (devops S3 v2 など) を入力します。
   c. [Description (説明)] に、このバージョンと他のバージョンを区別するための詳細を入力します。
   d. [次へ] を選択します。
5. [Review (確認)] ページで、情報が正しいことを確認し、[Confirm and upload (確認してアップロード)] を選択します。
6. ブラウザの [製品] ページで、新しい製品の URL をコピーします。これには製品 ID が含まれています。この製品 ID をコピーして保持します。CodePipeline でパイプラインを作成するときに、それを使用します。

   以下に示しているのは、`my-product` という製品の URL です。製品 ID を抽出するには、等号 (=) とアンパサンド (&) との間の値をコピーします。この例では、製品 ID は `prod-example123456` です。

   ```
   https://<region-URL>/servicecatalog/home?region=<region>#/admin-products?
   productCreated=prod-example123456&createdProductTitle=my-product
   ```

   > **Note**
   >
   > ページから移動する前に、製品の URL をコピーします。このページから移動したら、CLI を使用して製品 ID を取得する必要があります。

   数秒後、製品が [製品] ページに表示されます。製品をリストに表示するには、ブラウザの更新が必要になる場合があります。

ステップ 4: パイプラインを作成する

1. パイプラインに名前を付け、パイプラインのパラメータを選択するには、以下の手順を実行します。

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

b. [開始方法] を選択します。[パイプラインの作成] を選択し、パイプラインの名前を入力します。

c. サービスロール で、新しいサービスロール を選択し、CodePipeline に IAM でのサービスロールの作成を許可します。

d. [詳細設定] をデフォルト設定のままにし、[次へ] を選択します。

2. ソースステージを追加するには、以下の手順を実行します。

a. [ソースプロバイダ] で、AWS CodeCommit を選択します。

b. [リポジトリ名] と [ブランチ名] に、ソースアクションに使用するリポジトリとブランチを入力します。

c. [次へ] を選択します。

3. [Add build stage (ビルドステージの追加)] で [Skip build stage (ビルドステージのスキップ)] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

4. [Add deploy stage (デプロイステージの追加)] で、以下の手順を実行します。

a. [デプロイプロバイダ] で、[AWS Service Catalog] を選択します。

b. [設定ファイルの使用] を選択します。

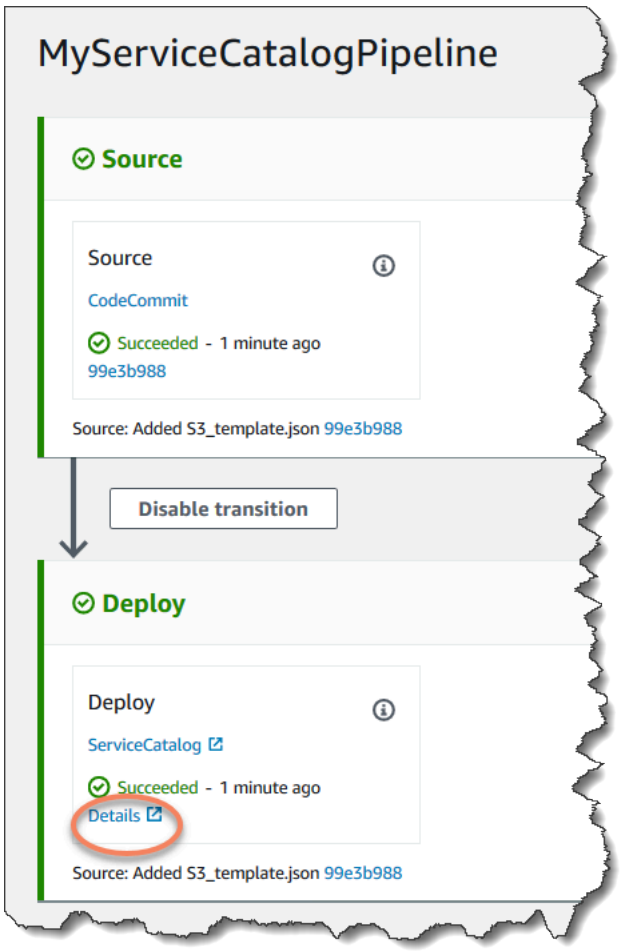
c. [プロダクト ID] に、Service Catalog コンソールからコピーしたプロダクト ID を貼り付けます。

d. [Configuration file path (設定ファイルのパス)] に、リポジトリ内の設定ファイルのファイルパスを入力します。

e. [次へ] を選択します。

5. [確認] で、パイプライン設定を確認し、[作成] を選択します。

6. パイプラインが正常に実行されたら、デプロイステージで [詳細] を選択して、製品を Service Catalog で開きます。



7. 製品情報で、バージョン名を選択して製品テンプレートを開きます。テンプレートのデプロイを表示します。

ステップ 5: 変更をプッシュして Service Catalog で製品を確認する

1. CodePipeline コンソールでパイプラインを表示し、ソースステージで [詳細] を選択します。コンソールでソース AWS CodeCommit リポジトリが開きます。[Edit (編集)] を選択して、ファイルの内容 (説明など) を変更します。

```
"Description": "Name of Amazon S3 bucket to hold and version website content"
```

2. 変更をコミットし、プッシュします。変更をプッシュした後、パイプラインが開始されます。パイプラインの実行が完了したら、デプロイステージで [詳細] を選択して、製品を Service Catalog で開きます。
3. 製品情報で、新しいバージョン名を選択して製品テンプレートを開きます。デプロイされたテンプレートの変更を表示します。

チュートリアル: を使用してパイプラインを作成する AWS CloudFormation

この例では、 AWS CloudFormation を使用して、ソースコードが変更されるたびにアプリケーションをインスタンスにデプロイするパイプラインを作成できるサンプルテンプレートを提供しています。このサンプルテンプレートでは、 AWS CodePipelineで表示できるパイプラインを作成します。パイプラインは Amazon CloudWatch Events を介して保存された変更の到着を検出します。

> **Important**
>
> パイプライン作成の一環として、CodePipeline は、ユーザーが指定した S3 アーティファクトバケットをアーティファクトとして使用します (これは S3 ソースアクションで使用するバケットとは異なります)。S3 アーティファクトバケットがパイプラインのアカウントとは異なるアカウントにある場合は、S3 アーティファクトバケットが によって所有 AWS アカウント されており、安全で信頼できることを確認してください。

トピック

- 例 1: を使用して AWS CodeCommit パイプラインを作成する AWS CloudFormation
- 例 2: を使用して Amazon S3 パイプラインを作成する AWS CloudFormation

例 1: を使用して AWS CodeCommit パイプラインを作成する AWS CloudFormation

このチュートリアルでは、 AWS CloudFormation コンソールを使用して、CodeCommit ソースリポジトリに接続されたパイプラインを含むインフラストラクチャを作成する方法を示します。このチュートリアルでは、提供されたサンプルテンプレートファイルを使用して、Amazon CloudWatch Events ルールなどのアーティファクトストア、パイプライン、変更検出リソースを含むリソーススタックを作成します。でリソーススタックを作成したら AWS CloudFormation、 AWS CodePipeline コンソールでパイプラインを表示できます。パイプラインは、CodeCommit サービスステージと CodeDeploy デプロイステージの 2 つのステージパイプラインになります。

前提条件:

AWS CloudFormation サンプルテンプレートで使用するには、次のリソースを作成しておく必要があります。

- ソースリポジトリを作成しておく必要があります。で作成した AWS CodeCommit リポジトリを使用できますチュートリアル: シンプルなパイプラインを作成する (CodeCommit リポジトリ)。
- CodeDeploy アプリケーションとデプロイグループを作成しておく必要があります。チュートリアル: シンプルなパイプラインを作成する (CodeCommit リポジトリ) で作成したCodeDeploy リソースを使用できます。
- パイプラインを作成するためのサンプル AWS CloudFormation テンプレートファイルをダウンロードするには、次のいずれかのリンクを選択します。YAML | JSON

  ファイルを解凍し、ローカルコンピュータに配置します。
- SampleApp_Linux.zip サンプルアプリケーションファイルをダウンロードします。

でパイプラインを作成する AWS CloudFormation

1. SampleApp_Linux.zip からファイルを解凍し、リポジトリにアップロードします AWS CodeCommit 。解凍したファイルをリポジトリのルートディレクトリにアップロードする必要があります。「ステップ 2: CodeCommit リポジトリにサンプルコードを追加する」の指示に従って、ファイルをリポジトリにプッシュできます。
2. AWS CloudFormation コンソールを開き、スタックの作成を選択します。[With new resources (standard)] (新しいリソースの使用 (標準)) を選択します。
3. [テンプレートの指定] で、[テンプレートのアップロード] を選択します。[ファイルを選択] を選択し、ローカルコンピュータからテンプレートファイルを選択します。[次へ] を選択します。
4. [スタック名] に、パイプラインの名前を入力します。サンプルテンプレートで指定されたパラメータが表示されます。以下のパラメータを入力します。

   a. ApplicationName に、CodeDeploy アプリケーションの名前を入力します。

   b. BetaFleet に CodeDeploy デプロイグループの名前を入力します。

   c. [BranchName] に、使用するリポジトリブランチを入力します。

   d. RepositoryName に CodeCommit サービスリポジトリの名前を入力します。
5. [次へ] を選択します。以下のページのデフォルト値を受け入れ、[次へ] を選択します。
6. 「機能」で、 が IAM リソースを作成する AWS CloudFormation 可能性があることを確認し、「スタックの作成」を選択します。
7. スタックの作成が完了したら、イベントリストを表示して、エラーがないか確認します。

   トラブルシューティング

でパイプラインを作成する IAM ユーザーには、パイプラインのリソースを作成するための追加のアクセス許可が必要になる AWS CloudFormation 場合があります。が CodeCommit パイプラインに必要な Amazon CloudWatch Events リソースを作成できるようにするには AWS CloudFormation 、ポリシーで次のアクセス許可が必要です。

```
{
     "Effect": "Allow",
     "Action": [
        "events:PutRule",
        "events:PutEvents",
        "events:PutTargets",
        "events:DeleteRule",
        "events:RemoveTargets",
        "events:DescribeRule"
     ],
     "Resource": "resource_ARN"
}
```

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

   [パイプライン] で、パイプラインを選択してから、[表示] を選択します。この図は、パイプラインのソースとデプロイのステージを示しています。

   > **Note**
   >
   > 作成されたパイプラインを表示するには、 CloudFormationのスタックの [リソース] タブで [論理 ID] 列を見つけます。パイプラインの [物理 ID] 列の名前をメモします。CodePipeline で、スタックを作成したリージョン内の同じ物理 ID (パイプライン名) のパイプラインを表示できます。

9. ソースリポジトリで、変更をコミットしてプッシュします。変更検出リソースが変更を受け取り、パイプラインが開始されます。

例 2: を使用して Amazon S3 パイプラインを作成する AWS CloudFormation

このチュートリアルでは、 AWS CloudFormation コンソールを使用して、Amazon S3 ソースバケットに接続されたパイプラインを含むインフラストラクチャを作成する方法を示します。このチュートリアルでは、提供されたサンプルテンプレートファイルを使用して、Amazon CloudWatch Events ルールや CloudTrail 証跡などのサービスバケット、アーティファクトストア、パイプライン、変更検出リソースを含むリソーススタックを作成します。でリソーススタックを作成したら AWS CloudFormation、 AWS CodePipeline コンソールでパイプラインを表示できます。パイプラインは、Amazon S3 ソースステージと CodeDeploy デプロイステージの 2 つのステージパイプラインになります。

前提条件:

AWS CloudFormation サンプルテンプレートで使用するには、次のリソースが必要です。

- Amazon EC2 インスタンスを作成して CodeDeploy エージェントをインスタンスにインストールしておく必要があります。CodeDeploy アプリケーションとデプロイグループを作成しておく必要があります。チュートリアル: シンプルなパイプラインを作成する (CodeCommit リポジトリ) で作成した Amazon EC2 リソースと CodeDeploy リソースを使用します。
- 次のリンクを選択して、Amazon S3 ソースでパイプラインを作成するためのサンプル AWS CloudFormation テンプレートファイルをダウンロードします。
  - パイプラインのサンプルテンプレートをダウンロードする: YAML | JSON
  - CloudTrail バケットおよび証跡のサンプルテンプレート (YAML | JSON) をダウンロードします。
  - ファイルを解凍し、ローカルコンピュータに配置します。
- SampleApp_Linux.zip からサンプルアプリケーションファイルをダウンロードします。

  .zip ファイルをローカルコンピューターに保存します。スタックの作成後、.zip ファイルをアップロードします。

でパイプラインを作成する AWS CloudFormation

1. AWS CloudFormation コンソールを開き、スタックの作成を選択します。[With new resources (standard)] (新しいリソースの使用 (標準)) を選択します。
2. [テンプレートの選択] で、[テンプレートのアップロード] を選択します。[ファイルの選択] を選択し、ローカルコンピュータからテンプレートファイルを選択します。[次へ] を選択します。

3. [スタック名] に、パイプラインの名前を入力します。サンプルテンプレートで指定されたパラメータが表示されます。以下のパラメータを入力します。

   a. ApplicationName に、CodeDeploy アプリケーションの名前を入力します。`DemoApplication` デフォルト名は置き換えることができます。

   b. BetaFleet に CodeDeploy デプロイグループの名前を入力します。`DemoFleet` デフォルト名は置き換えることができます。

   c. [SourceObjectKey] に `SampleApp_Linux.zip` と入力します。このファイルは、テンプレートによってバケットとパイプラインが作成された後に、バケットにアップロードします。

4. [次へ] を選択します。以下のページのデフォルト値を受け入れ、[次へ] を選択します。

5. 「機能」で、 が IAM リソースを作成する AWS CloudFormation 可能性があることを確認し、「スタックの作成」を選択します。

6. スタックの作成が完了したら、イベントリストを表示して、エラーがないか確認します。

   トラブルシューティング

   でパイプラインを作成している IAM ユーザーには、パイプラインのリソースを作成するための追加のアクセス許可が必要になる AWS CloudFormation 場合があります。が Amazon S3 パイプラインに必要な Amazon CloudWatch Events リソースを作成できるようにするには AWS CloudFormation 、ポリシーで次のアクセス許可が必要です。 Amazon S3

   ```
   {
        "Effect": "Allow",
        "Action": [
           "events:PutRule",
           "events:PutEvents",
           "events:PutTargets",
           "events:DeleteRule",
           "events:RemoveTargets",
           "events:DescribeRule"
        ],
        "Resource": "resource_ARN"
   }
   ```

7. スタックのリソースタブ CloudFormationで、スタック用に作成されたリソースを表示します。

> **Note**
>
> 作成されたパイプラインを表示するには、 CloudFormationのスタックの [リソース] タブで [論理 ID] 列を見つけます。パイプラインの [物理 ID] 列の名前をメモします。CodePipeline で、スタックを作成したリージョン内の同じ物理 ID (パイプライン名) のパイプラインを表示できます。

名前に `sourcebucket` ラベルが付いた S3 バケットを選択します (`s3-cfn-codepipeline-sourcebucket-y04EXAMPLE.` など)。パイプラインアーティファクトバケットは選択しないでください。

リソースは CloudFormationによって新しく作成されたため、ソースバケットは空です。Amazon S3 コンソールを開き、`sourcebucket` バケットを見つけます。[アップロード] を選択し、指示に従って `SampleApp_Linux.zip` .zip ファイルをアップロードします。

> **Note**
>
> Amazon S3 がパイプラインのサービスプロバイダである場合、すべてのサービスファイルを 1 つの .zip ファイルとしてパッケージ化したバケットにアップロードする必要があります。それ以外の場合、ソースアクションは失敗します。

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

   [パイプライン] で、パイプラインを選択してから、[表示] を選択します。この図は、パイプラインのソースとデプロイのステージを示しています。

9. AWS CloudTrail リソースを作成するには、以下の手順を実行します。

で AWS CloudTrail リソースを作成する AWS CloudFormation

1. AWS CloudFormation コンソールを開き、スタックの作成を選択します。
2. [テンプレートの選択] で、[テンプレートを Amazon S3 にアップロード] を選択します。参照を選択し、ローカルコンピュータから AWS CloudTrail リソースのテンプレートファイルを選択します。[次へ] を選択します。

3. [スタックの名前] にリソーススタックの名前を入力します。サンプルテンプレートで指定されたパラメータが表示されます。以下のパラメータを入力します。

   - SourceObjectKey では、サンプルアプリケーションの zip ファイルのデフォルトを受け入れます。

4. [次へ] を選択します。以下のページのデフォルト値を受け入れ、[次へ] を選択します。
5. 「機能」で、「 が IAM リソースを作成する AWS CloudFormation 可能性がある」を選択し、「作成」を選択します。
6. スタックの作成が完了したら、イベントリストを表示して、エラーがないか確認します。

   が Amazon S3 パイプラインに必要な CloudTrail リソース AWS CloudFormation を作成できるようにするには、ポリシーで次のアクセス許可が必要です。

```
{
     "Effect": "Allow",
     "Action": [
        "cloudtrail:CreateTrail",
        "cloudtrail:DeleteTrail",
        "cloudtrail:StartLogging",
        "cloudtrail:StopLogging",
        "cloudtrail:PutEventSelectors"
     ],
     "Resource": "resource_ARN"
}
```

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

   [パイプライン] で、パイプラインを選択してから、[表示] を選択します。この図は、パイプラインのソースとデプロイのステージを示しています。

8. ソースバケットで、変更をコミットしてプッシュします。変更検出リソースが変更を受け取り、パイプラインが開始されます。

チュートリアル: AWS CloudFormation デプロイアクションの変数を使用するパイプラインを作成する

このチュートリアルでは、 AWS CodePipeline コンソールを使用して、デプロイアクションを含むパイプラインを作成します。パイプラインが実行されると、テンプレートはスタックを作

成し、さらに outputs ファイルを作成します。スタックテンプレートによって生成された出力は、CodePipeline の AWS CloudFormation アクションによって生成された変数です。

テンプレートからスタックを作成するアクションに、変数の名前空間を指定します。outputs ファイルによって生成された変数は、以降のアクションで使用できます。この例では、 AWS CloudFormation アクションによって生成されたStackName変数に基づいて変更セットを作成します。手動承認の後で、変更セットを実行し、StackName 変数に基づいてスタックを削除するスタック削除アクションを作成します。

> **Important**
>
> パイプライン作成の一環として、CodePipeline は、ユーザーが指定した S3 アーティファクトバケットをアーティファクトとして使用します (これは S3 ソースアクションで使用するバケットとは異なります)。S3 アーティファクトバケットがパイプラインのアカウントとは異なるアカウントにある場合は、S3 アーティファクトバケットが によって所有 AWS アカウント されており、安全で信頼できることを確認してください。

トピック

- 前提条件: AWS CloudFormation サービスロールと CodeCommit リポジトリを作成する
- ステップ 1: サンプル AWS CloudFormation テンプレートをダウンロード、編集、アップロードする
- ステップ 2: パイプラインを作成する
- ステップ 3: CloudFormation デプロイアクションを追加して変更セットを作成する
- ステップ 4: 手動承認アクションを追加する
- ステップ 5: 変更セットを実行するための CloudFormation デプロイアクションを追加する
- ステップ 6: スタックを削除するための CloudFormation デプロイアクションを追加する

前提条件: AWS CloudFormation サービスロールと CodeCommit リポジトリを作成する

以下のものを用意しておく必要があります。

- CodeCommit リポジトリ。で作成した AWS CodeCommit リポジトリを使用できますチュートリアル: シンプルなパイプラインを作成する (CodeCommit リポジトリ)。

- 次の例では、テンプレートから Amazon DocumentDB スタックを作成します。 AWS Identity and Access Management (IAM) を使用して、Amazon DocumentDB の次のアクセス許可を持つ AWS CloudFormation サービスロールを作成する必要があります。

```
"rds:DescribeDBClusters",
"rds:CreateDBCluster",
"rds:DeleteDBCluster",
"rds:CreateDBInstance"
```

ステップ 1: サンプル AWS CloudFormation テンプレートをダウンロード、編集、アップロードする

サンプル AWS CloudFormation テンプレートファイルをダウンロードし、CodeCommit リポジトリにアップロードします。

1. リージョンのサンプルテンプレートに移動します。例えば、https://docs.aws.amazon.com/documentdb/latest/developerguide/quick_start_cfn.html#quick_start_cfn-launch_stack のテーブルを使用してリージョンを選択し、テンプレートをダウンロードします。Amazon DocumentDB クラスターのテンプレートをダウンロードします。ファイル名は `documentdb_full_stack.yaml` です。
2. `documentdb_full_stack.yaml` ファイルを解凍し、テキストエディタで開きます。以下の変更を加えます。

   a. 次の例では、テンプレートの `Parameters` セクションに次の `Purpose:` パラメータを追加します。

   ```
   Purpose:
     Type: String
     Default: testing
     AllowedValues:
       - testing
       - production
     Description: The purpose of this instance.
   ```

   b. 次の例では、テンプレートの `Outputs:` セクションに次の `StackName` 出力を追加します。

   ```
   StackName:
   ```

```
    Value: !Ref AWS::StackName
```

3. テンプレートファイルを AWS CodeCommit リポジトリにアップロードします。解凍および編集したテンプレートファイルを、リポジトリのルートディレクトリにアップロードする必要があります。

   CodeCommit コンソールを 使用して、ファイルをアップロードします。

   a. CodeCommit コンソールを開き、リポジトリ リストから自分のリポジトリを選択します。

   b. [Add file]、[Upload file] の順に選択します。

   c. [ファイルの選択] を選択し、ファイルを参照します。ユーザー名とメールアドレスを入力して、変更をコミットします。[Commit changes] (変更のコミット) を選択します。

   ファイルは、リポジトリのルートレベルに次のように表示されます。

```
documentdb_full_stack.yaml
```

ステップ 2: パイプラインを作成する

このセクションでは、次のアクションを使用してパイプラインを作成します。

- ソースアーティファクトがテンプレートファイルである CodeCommit アクションを含むソースステージ。
- デプロイアクションを含む CloudFormation デプロイステージ。

ウィザードによって作成されたソースステージとデプロイステージの各アクションには、変数の名前空間として `SourceVariables`、`DeployVariables` がそれぞれ割り当てられます。アクションには名前空間が割り当てられるため、この例で設定した変数はダウンストリームアクションで使用可能になります。詳細については、「変数リファレンス」を参照してください。

ウィザードを使用してパイプラインを作成するには

1. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。

3. [ステップ 1: 作成オプションを選択する] ページの [作成オプション] で、[カスタムパイプラインを構築する] オプションを選択します。[次へ] を選択します。
4. [ステップ 2: パイプラインの設定を選択する] で、[パイプライン名] に「**MyCFNDeployPipeline**」と入力します。
5. CodePipeline は、特徴と料金が異なる V1 タイプと V2 タイプのパイプラインを提供しています。V2 タイプは、コンソールで選択できる唯一のタイプです。詳細については、「パイプラインタイプ」を参照してください。CodePipeline の料金については、料金を参照してください。
6. [Service role (サービスロール)] で、次のいずれかの操作を行います。

   - 新しいサービスロール を選択して、CodePipeline に IAM でのサービスロールの作成を許可します。
   - [Existing service role (既存のサービスロール)] を選択します。[ロール名] で、リストからサービスロールを選択します。
7. アーティファクトストア:

   a. パイプライン用に選択したリージョンのパイプラインに、デフォルトとして指定された Amazon S3 アーティファクトバケットなどのデフォルトのアーティファクトストアを使用するには、デフォルトの場所 を選択します。

   b. Amazon S3 アーティファクトバケットなどのアーティファクトストアがパイプラインと同じリージョンに既に存在する場合は、カスタムの場所 を選択します。

   > **Note**
   >
   > これはソースコードのソースバケットではありません。パイプラインのアーティファクトストアです。パイプラインごとに S3 バケットなどの個別のアーティファクトストアが必要です。パイプラインを作成または編集するときは、パイプラインリージョンにアーティファクトバケットと、アクションを実行している AWS リージョンごとに 1 つのアーティファクトバケットが必要です。
   > 詳細については、「入力および出力アーティファクト」および「CodePipeline パイプライン構造リファレンス」を参照してください。

   [次へ] を選択します。
8. [ステップ 3: ソースステージの追加] で、次の操作を行います。

   a. [ソースプロバイダ] で、AWS CodeCommit を選択します。

b. リポジトリ名 で、ステップ 1: CodeCommit リポジトリを作成する で作成した CodeCommit リポジトリの名前を選択します。

c. [Branch name] で、最新のコード更新を含むブランチの名前を選択します。

リポジトリ名とブランチを選択した後、このパイプライン用に作成される Amazon CloudWatch Events ルールが表示されます。

[次へ] を選択します。

9. [ステップ 4: ビルドステージの追加] で、[ビルドステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

[次へ] を選択します。

10. [ステップ 5: テストステージの追加] で、[テストステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

[次へ] を選択します。

11. ステップ 6: デプロイステージの追加:

a. [アクション名] で、[デプロイ] をクリックします。[デプロイプロバイダー] で、[CloudFormation] を選択します。

b. [アクションモード] で、[スタックを作成または更新する] をクリックします。

c. [スタック名] に、スタックの名前を入力します。これは、テンプレートが作成するスタックの名前です。

d. [出力ファイル名] に、出力ファイルの名前 (**outputs** など) を入力します。これは、スタックの作成後にアクションによって作成されるファイルの名前です。

e. [Advanced] を展開します。[パラメータの上書き] で、テンプレートの上書きをキーと値のペアとして入力します。例えば、このテンプレートには、次の上書きが必要です。

```
{
"DBClusterName": "MyDBCluster",
"DBInstanceName": "MyDBInstance",
"MasterUser": "UserName",
"MasterPassword": "Password",
"DBInstanceClass": "db.r4.large",
"Purpose": "testing"}
```

上書きを入力しない場合、テンプレートはスタックをデフォルト値で作成します。

f. [次へ] を選択します。

g. ステップ 7: 確認で、パイプラインの作成を選択します。パイプラインステージを示す図が表示されます。パイプラインの実行を許可します。2 つのステージで構成されたパイプラインが完成し、他のステージを追加する準備が整いました。

ステップ 3: CloudFormation デプロイアクションを追加して変更セットを作成する

手動承認アクションの前に CloudFormation が変更セットを作成できるようにする次のアクションをパイプラインに作成します。

1. CodePipeline コンソール (https://console.aws.amazon.com/codepipeline/) を開きます。

   [パイプライン] で、パイプラインを選択してから、[表示] を選択します。この図は、パイプラインのソースとデプロイのステージを示しています。

2. [編集] モードで、パイプラインを編集するか、引き続きパイプラインを表示するかを選択します。

3. デプロイステージを編集することを選択します。

4. 前のアクションで作成したスタックに対する変更セットを作成するデプロイアクションを追加します。このアクションは、ステージ内の既存のアクションの後に追加します。

   a. [アクション名] に、「Change_Set」と入力します。[アクションプロバイダー] で、[AWS CloudFormation] を選択します。

   b. [入力アーティファクト] で、[SourceArtifact] を選択します。

   c. [Action mode] (アクションモード) で [Create or replace a change set] (変更セットの作成または置換) を選択します。

   d. [スタック名] に、次のように変数の構文を入力します。これは、変更セットを作成する対象のスタックの名前です。アクションには、デフォルトの名前空間 `DeployVariables` が割り当てられます。

   ```
   #{DeployVariables.StackName}
   ```

   e. [変更セット名] に、変更セットの名前を入力します。

   ```
   my-changeset
   ```

f. [パラメータの上書き] で、Purpose パラメータを testing から production に変更します。

```
{
"DBClusterName": "MyDBCluster",
"DBInstanceName": "MyDBInstance",
"MasterUser": "UserName",
"MasterPassword": "Password",
"DBInstanceClass": "db.r4.large",
"Purpose": "production"}
```

g. [完了] をクリックしてアクションを保存します。

ステップ 4: 手動承認アクションを追加する

パイプラインで手動承認アクションを作成します。

1. [編集] モードで、パイプラインを編集するか、引き続きパイプラインを表示するかを選択します。
2. デプロイステージを編集することを選択します。
3. 変更セットを作成するデプロイアクションの後に、手動承認アクションを追加します。このアクションにより、パイプラインが変更セットを実行する CloudFormation 前に、 で作成されたリソース変更セットを確認できます。

ステップ 5: 変更セットを実行するための CloudFormation デプロイアクションを追加する

手動承認アクションの後に CloudFormation が変更セットを実行できるようにする次のアクションをパイプラインに作成します。

1. CodePipeline コンソール (https://console.aws.amazon.com/codepipeline/) を開きます。

   [パイプライン] で、パイプラインを選択してから、[表示] を選択します。この図は、パイプラインのソースとデプロイのステージを示しています。
2. [編集] モードで、パイプラインを編集するか、引き続きパイプラインを表示するかを選択します。

3. デプロイステージを編集することを選択します。
4. 前の手動アクションで承認した変更セットを実行するデプロイアクションを追加します。

   a. [アクション名] に、「Execute_Change_Set」と入力します。[アクションプロバイダー] で、[AWS CloudFormation] を選択します。

   b. [入力アーティファクト] で、[SourceArtifact] を選択します。

   c. [Action mode] (アクションモード) で、 [Execute a change set] (変更セットの実行) を選択します。

   d. [スタック名] に、次のように変数の構文を入力します。これは、変更セットを作成する対象のスタックの名前です。

   ```
   #{DeployVariables.StackName}
   ```

   e. [変更セット名] に、前のアクションで作成した変更セットの名前を入力します。

   ```
   my-changeset
   ```

   f. [完了] をクリックしてアクションを保存します。

   g. パイプラインの実行を続行します。

ステップ 6: スタックを削除するための CloudFormation デプロイアクションを追加する

が出力ファイルの 変数からスタック名を取得し、スタックを削除 CloudFormation できるようにする最終アクションをパイプラインに作成します。

1. CodePipeline コンソール (https://console.aws.amazon.com/codepipeline/) を開きます。

   [パイプライン] で、パイプラインを選択してから、[表示] を選択します。この図は、パイプラインのソースとデプロイのステージを示しています。
2. パイプラインの編集を選択します。
3. デプロイステージを編集することを選択します。
4. スタックを削除するデプロイアクションを追加します。

   a. [アクション名] で、[DeleteStack] を選択します。[デプロイプロバイダー] で、[CloudFormation] を選択します。

b. [アクションモード] で、[スタックを削除する] をクリックします。

c. [スタック名] に、次のように変数の構文を入力します。これは、アクションで削除するスタックの名前です。

d. [完了] をクリックしてアクションを保存します。

e. [保存] をクリックしてポリシーを保存します。

パイプラインは保存すると実行されます。

チュートリアル: CodePipeline を使用した Amazon ECS 標準デプロイ

このチュートリアルでは、CodePipeline を使用して Amazon ECS で完全なエンドツーエンドの継続的デプロイメント (CD) パイプラインを作成する方法を説明します。

Important

コンソールでのパイプライン作成の一環として、CodePipeline は S3 アーティファクトバケットをアーティファクトとして使用します (これは S3 ソースアクションで使用するバケットとは異なります)。S3 アーティファクトバケットがパイプラインのアカウントとは異なるアカウントにある場合は、S3 アーティファクトバケットが によって所有 AWS アカウントされており、安全で信頼できることを確認してください。

Note

このトピックとチュートリアルでは、CodePipeline の Amazon ECS 標準デプロイアクションについて説明します。CodePipeline で Amazon ECS から CodeDeploy の blue/green デプロイアクションを使用するチュートリアルは、チュートリアル: Amazon ECR ソース、ECS - CodeDeploy 間のデプロイでパイプラインを作成する を参照してください。

Note

このトピックとチュートリアルでは、ソースアクションを含む CodePipeline の Amazon ECS 標準デプロイアクションについて説明します。Amazon ECS 標準デプロイアクション

と CodePipeline の ECRBuildAndPublish ビルドアクションを使用してイメージをプッシュするチュートリアルについては、「」を参照してくださいチュートリアル: CodePipeline (V2 タイプ) を使用して Docker イメージを構築し Amazon ECR にプッシュする。

前提条件

このチュートリアルで CD パイプラインを作成する前に、いつくかのリソースを用意する必要があります。使用を開始するために必要なものは以下のとおりです。

> **Note**
>
> これらのリソースはすべて、同じ AWS リージョン内に作成する必要があります。

- Dockerfile およびアプリケーションリソースを使用するソースコントロールリポジトリ (このチュートリアルでは CodeCommit を使用します)。詳細については、AWS CodeCommit ユーザーガイド の「CodeCommit リポジトリの作成」を参照してください。
- Dockerfile およびアプリケーションソースから作成したイメージを含む Docker イメージリポジトリ (このチュートリアルでは Amazon ECR を使用します)。詳細については、Amazon Elastic Container Registry ユーザーガイド の「リポジトリの作成」と「イメージをプッシュする」を参照してください。
- イメージリポジトリでホストされた Docker イメージを参照する Amazon ECS タスク定義。詳細については、Amazon Elastic Container Service デベロッパーガイド の「タスク定義の作成」を参照してください。

> **Important**
>
> CodePipeline の Amazon ECS 標準デプロイアクションは、Amazon ECS サービスで使用されるリビジョンに基づいて、タスク定義の独自のリビジョンを作成します。Amazon ECS サービスを更新せずにタスク定義の新しいリビジョンを作成した場合、デプロイアクションはそれらのリビジョンを無視します。

このチュートリアルで使用するタスク定義の例を以下に示します。`name` と `family` に使用する値は、ビルド仕様ファイルのために次のステップで使用します。

```
{
  "ipcMode": null,
  "executionRoleArn": "role_ARN",
  "containerDefinitions": [
    {
      "dnsSearchDomains": null,
      "environmentFiles": null,
      "logConfiguration": {
        "logDriver": "awslogs",
        "secretOptions": null,
        "options": {
          "awslogs-group": "/ecs/hello-world",
          "awslogs-region": "us-west-2",
          "awslogs-stream-prefix": "ecs"
        }
      },
      "entryPoint": null,
      "portMappings": [
        {
          "hostPort": 80,
          "protocol": "tcp",
          "containerPort": 80
        }
      ],
      "command": null,
      "linuxParameters": null,
      "cpu": 0,
      "environment": [],
      "resourceRequirements": null,
      "ulimits": null,
      "dnsServers": null,
      "mountPoints": [],
      "workingDirectory": null,
      "secrets": null,
      "dockerSecurityOptions": null,
      "memory": null,
      "memoryReservation": 128,
      "volumesFrom": [],
      "stopTimeout": null,
      "image": "image_name",
      "startTimeout": null,
      "firelensConfiguration": null,
      "dependsOn": null,
```

```
      "disableNetworking": null,
      "interactive": null,
      "healthCheck": null,
      "essential": true,
      "links": null,
      "hostname": null,
      "extraHosts": null,
      "pseudoTerminal": null,
      "user": null,
      "readonlyRootFilesystem": null,
      "dockerLabels": null,
      "systemControls": null,
      "privileged": null,
      "name": "hello-world"
    }
  ],
  "placementConstraints": [],
  "memory": "2048",
  "taskRoleArn": null,
  "compatibilities": [
    "EC2",
    "FARGATE"
  ],
  "taskDefinitionArn": "ARN",
  "family": "hello-world",
  "requiresAttributes": [],
  "pidMode": null,
  "requiresCompatibilities": [
    "FARGATE"
  ],
  "networkMode": "awsvpc",
  "cpu": "1024",
  "revision": 1,
  "status": "ACTIVE",
  "inferenceAccelerators": null,
  "proxyConfiguration": null,
  "volumes": []
}
```

- 前に説明したタスク定義を使用するサービスを実行する Amazon ECS クラスター。詳細については、Amazon Simple Queue Serviceデベロッパーガイド の クラスターの作成 と サービスの作成 を参照してください。

これらの前提条件を満たした後、チュートリアルに進んで CD パイプラインを作成できます。

ステップ 1: ビルド仕様ファイルをソースリポジトリに追加する

このチュートリアルでは、CodeBuild を使用して Docker イメージを構築し、Amazon ECR にイメージをプッシュします。buildspec.yml ファイルをソースコードリポジトリに追加して CodeBuild に処理方法を指示します。ビルド仕様の以下の例では、次のように動作します。

- プレビルドステージ:
  - Amazon ECR にログインします。
  - リポジトリ URI を ECR イメージに設定して、ソースの Git コミット ID の最初の 7 文字を使用するイメージタグを追加します。
- ビルドステージ
  - Docker イメージを作成し、イメージに latest と Git コミット ID の両方をタグ付けします。
- ポストビルドステージ:
  - 両方のタグを持った ECR リポジトリにイメージをプッシュします。
  - Amazon ECS サービスのコンテナ名およびイメージとタグがあるビルドのルートに imagedefinitions.json という名前のファイルを作成します。CD パイプラインのデプロイステージでこの情報を使用してサービスのタスク定義の新しいリビジョンを作成し、新しいタスク定義を使用してサービスを更新します。imagedefinitions.json ファイルは ECS ジョブワーカーに必須です。

このサンプルテキストを貼り付けて、buildspec.yml ファイルを使用して、イメージとタスク定義の値を置き換えます。このテキストでは、例としてアカウント ID 111122223333 を使用しています。

```
version: 0.2

phases:
  pre_build:
    commands:
      - echo Logging in to Amazon ECR...
      - aws --version
      - aws ecr get-login-password --region $AWS_DEFAULT_REGION | docker login --
username AWS --password-stdin 111122223333.dkr.ecr.us-west-2.amazonaws.com
      - REPOSITORY_URI=012345678910.dkr.ecr.us-west-2.amazonaws.com/hello-world
      - COMMIT_HASH=$(echo $CODEBUILD_RESOLVED_SOURCE_VERSION | cut -c 1-7)
      - IMAGE_TAG=${COMMIT_HASH:=latest}
```

```
  build:
    commands:
      - echo Build started on `date`
      - echo Building the Docker image...
      - docker build -t $REPOSITORY_URI:latest .
      - docker tag $REPOSITORY_URI:latest $REPOSITORY_URI:$IMAGE_TAG
  post_build:
    commands:
      - echo Build completed on `date`
      - echo Pushing the Docker images...
      - docker push $REPOSITORY_URI:latest
      - docker push $REPOSITORY_URI:$IMAGE_TAG
      - echo Writing image definitions file...
      - printf '[{"name":"hello-world","imageUri":"%s"}]' $REPOSITORY_URI:$IMAGE_TAG >
 imagedefinitions.json
artifacts:
    files: imagedefinitions.json
```

このチュートリアルで使用する Amazon ECS サービスで、前提条件 で提供されているサンプルタスクの定義に合わせてビルド仕様が書き込まれています。REPOSITORY_URI 値は image リポジトリ (イメージタグなし) に対応し、ファイルの末尾近くの *hello-world* 値はサービスのタスク定義のコンテナ名に対応します。

ソースリポジトリに **buildspec.yml** ファイルを追加するには

1. テキストエディタを開き、上記のビルド仕様をコピーして新しいファイルに貼り付けます。
2. REPOSITORY_URI の値 (*012345678910.dkr.ecr.us-west-2.amazonaws.com/hello-world*) を、Docker イメージの自分の Amazon ECR リポジトリ URI (イメージタグなし) に置き換えます。*hello-world* を、Docker イメージを参照するサービスのタスク定義のコンテナ名に置き換えます。
3. ソースリポジトリに buildspec.yml ファイルをコミットし、プッシュします。

   a. ファイルを追加します。

   ```
   git add .
   ```

   b. 変更をコミットします。

   ```
   git commit -m "Adding build specification."
   ```

   c. コミットをプッシュします。

```
git push
```

ステップ 2: 継続的デプロイパイプラインを作成する

CodePipeline ウィザードを使用してパイプラインステージを作成し、ソースリポジトリを ECS サービスに接続します。

パイプラインを作成するには

1. CodePipeline コンソール (https://console.aws.amazon.com/codepipeline/) を開きます。
2. [Welcome (ようこそ)] ページで、[Create pipeline (パイプラインの作成)] を選択します。

   CodePipeline を初めて使用する場合は、[Welcome (ようこそ)] の代わりに紹介ページが表示されます。[今すぐ始める] を選択します。
3. [ステップ 1: 作成オプションを選択する] ページの [作成オプション] で、[カスタムパイプラインを構築する] オプションを選択します。[次へ] を選択します。
4. [ステップ 2: パイプラインの設定を選択する] ページで、[パイプライン名] にパイプラインの名前を入力します。このチュートリアルでは、パイプライン名は hello-world です。
5. [パイプラインのタイプ] で、デフォルトの選択を [V2] のままにします。パイプラインのタイプによって特徴および価格が異なります。詳細については、「パイプラインのタイプ」を参照してください。[Next] (次へ) を選択します。
6. ステップ 3: ソースステージの追加ページで、ソースプロバイダーで AWS CodeCommitを選択します。

   a. [Repository name (リポジトリ名)] で、パイプラインのソース場所として使用する リポジトリの名前を選択します。

   b. [ブランチ名] で使用するブランチを選択し、[Next (次へ)] を選択します。
7. ステップ 4: ビルドステージの追加ページで、ビルドプロバイダーで を選択しAWS CodeBuild、プロジェクトの作成を選択します。

   a. [Project name] では、ビルドプロジェクトに一意の名前を選択します。このチュートリアルでは、プロジェクト名は hello-world です。

   b. [環境イメージ] で、[Managed image (マネージド型イメージ)] を選択します。

   c. [オペレーティングシステム] で、[Amazon Linux 2] を選択します。

   d. [ランタイム] で、[Standard (標準)] を選択します。

e. [イメージ] で、[**aws/codebuild/amazonlinux2-x86_64-standard:3.0**] を選択します。

f. [イメージバージョン] と [環境タイプ] には、既定値を使用します。

g. [Enable this flag if you want to build Docker images or want your builds to get elevated privileges (Docker イメージを構築する場合、またはビルドで昇格された権限を取得する場合は、このフラグを有効にする)] を選択します。

h. [CloudWatch logs] の選択を解除します。アドバンスト の拡張を必要とする場合があります。

i. [Continue to CodePipeline] (CodePipeline に進む) を選択します。

j. [次へ] を選択します。

> **Note**
>
> ウィザードによって、codebuild-***build-project-name***-service-role という名前のビルドプロジェクト用の CodeBuild サービスロールが作成されます。このロール名を書き留めます。これには後で Amazon ECR アクセス権限を追加します。

8. [Step 5: デプロイステージの追加] の [デプロイプロバイダ] で [Amazon ECS] を選択します。

a. [Cluster name (クラスター名)] で、サービスが実行されている Amazon ECS クラスターを選択します。このチュートリアルでは、クラスターは default です。

b. [サービス名] で更新するサービスを選択し、[Next (次へ)] を選択します。このチュートリアルでは、サービス名は hello-world です。

9. [Step 6: Review] ページで、パイプラインの設定を確認し、[Create pipeline] を選択してパイプラインを作成します。

> **Note**
>
> これでパイプラインが作成され、さまざまなパイプラインステージを通して実行を試みます。ただし、ウィザードによって作成されたデフォルトの CodeBuild ロールには、`buildspec.yml` ファイルに含まれるコマンドのすべてを実行するアクセス権限がないため、ビルドステージは失敗します。次のセクションで、ビルドステージのアクセス権限を追加します。

ステップ 3: CodeBuild ロールに Amazon ECR 権限を追加する

CodePipeline ウィザードによって、codebuild-***build-project-name***-service-role という名前の CodeBuild ビルドプロジェクト用の IAM ロールが作成されます。このチュートリアルで名前は codebuild-hello-world-service-role です。`buildspec.yml` ファイルは Amazon ECR API オペレーションの呼び出しを実行するため、これらの Amazon ECR コールを行うアクセス権限を許可するポリシーがロールに必要です。以下の手順では、適切なアクセス権限をロールにアタッチします。

ステップ 3: CodeBuild ロールに Amazon ECR 権限を追加する

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左のナビゲーションペインで、[ロール] を選択してください。
3. 検索ボックスに「codebuild-」と入力し、CodePipeline ウィザードによって作成されたロールを選択します。このチュートリアルでは、ロール名は codebuild-hello-world-service-role です。
4. [Summary (概要)] ページで、[Attach policy (ポリシーのアタッチ)] を選択します。
5. [AmazonEC2ContainerRegistryPowerUser] ポリシーの左にあるボックスをオンにし、[Attach policy] を選択します。

ステップ 4: パイプラインのテスト

パイプラインには、end-to-endのネイティブ AWS 継続的デプロイを実行するためのすべてが必要です。次は、コードの変更をソースリポジトリにプッシュすることで機能をテストします。

パイプラインをテストするには

1. 設定済みソースリポジトリにコード変更を行い、変更をコミットしてプッシュします。
2. CodePipeline コンソール (https://console.aws.amazon.com/codepipeline/) を開きます。
3. リストからパイプラインを選択します。
4. ステージを通してパイプラインの進行状況を監視します。パイプラインが終了し、Amazon ECS サービスがコード変更から作成された Docker イメージを実行することを確認します。

チュートリアル: Amazon ECR ソース、ECS - CodeDeploy 間のデプロイでパイプラインを作成する

このチュートリアルでは、Docker イメージをサポートする Blue/Green デプロイを使用して AWS CodePipeline コンテナアプリケーションをデプロイするパイプラインを で設定します。Blue/Green デプロイでは、古いバージョンと一緒に新しいバージョンのアプリケーションを起動し、トラフィックを再ルーティングする前に新しいバージョンをテストできます。また、デプロイプロセスをモニタリングし、問題がある場合は迅速にロールバックすることもできます。

> **Important**
>
> パイプライン作成の一環として、CodePipeline は、ユーザーが指定した S3 アーティファクトバケットをアーティファクトとして使用します (これは S3 ソースアクションで使用するバケットとは異なります)。S3 アーティファクトバケットがパイプラインのアカウントとは異なるアカウントにある場合は、S3 アーティファクトバケットが によって所有 AWS アカウント されており、安全で信頼できることを確認してください。

> **Note**
>
> このチュートリアルでは、CodePipeline の Amazon ECS から CodeDeploy Blue/Green デプロイアクションについて説明します。CodePipeline で Amazon ECS スタンダードデプロイアクションを使用するチュートリアルについては、チュートリアル: CodePipeline を使用した Amazon ECS 標準デプロイ を参照してください。

完了したパイプラインは、Amazon ECR などのイメージリポジトリに保存されているイメージへの変更を検出し、CodeDeploy を使用してトラフィックを AmazonECS クラスターとロードバランサーにルーティングおよびデプロイします。CodeDeploy は、リスナーを使用して、AppSpec ファイルで指定し更新されたコンテナのポートにトラフィックを再ルーティングします。ロードバランサー、製造リスナー、ターゲットグループ、Amazon ECS アプリケーションが青/緑のデプロイでどのように使用されるかについては、チュートリアル：Amazon ECS サービスのデプロイ を参照してください。

また、Amazon ECS タスク定義が保存されている CodeCommit リポジトリなどのソース場所を使用するようにパイプラインを設定することもできます。このチュートリアルでは、これらの各 AWS リソースを設定し、各リソースのアクションを含むステージでパイプラインを作成します。

ソースコードが変更されたり、新しいベースイメージが Amazon ECR にアップロードしたときは、継続的デリバリーパイプラインが自動的にコンテナイメージを構築してデプロイします。

このフローでは、次のアーティファクトを使用します。

- Amazon ECR イメージリポジトリのコンテナ名とリポジトリ URI を指定する Docker イメージファイル。
- Docker イメージ名、コンテナ名、Amazon ECS サービス名、およびロードバランサーの設定を一覧表示する Amazon ECS タスク定義。
- Amazon ECS タスク定義ファイルの名前、更新されたアプリケーションのコンテナの名前、および CodeDeploy が製造トラフィックを再ルーティングするコンテナポートを指定する CodeDeployAppSpec ファイル。デプロイライフサイクルイベントフック中に実行できるオプションのネットワーク設定と Lambda 関数も指定できます。

> **Note**
>
> Amazon ECR イメージリポジトリへの変更をコミットすると、パイプラインソースアクションはそのコミット用に `imageDetail.json` ファイルを作成します。`imageDetail.json` ファイルの詳細については、「Amazon ECS Blue/Green デプロイアクション用の imageDetail.json ファイル」を参照してください。

パイプラインを作成または編集し、デプロイステージのソースアーティファクトを更新または指定するときは、使用する最新の名前とバージョンのソースアーティファクトを必ず指してください。パイプラインを設定した後、イメージまたはタスク定義を変更したら、リポジトリ内のソースアーティファクトファイルを更新してから、パイプラインのデプロイステージを編集する必要があります。

トピック

- 前提条件
- ステップ 1: イメージを作成して Amazon ECR リポジトリにプッシュする
- ステップ 2: タスク定義ソースファイルと AppSpec ソースファイルを作成して、CodeCommit リポジトリにプッシュする
- ステップ 3: Application Load Balancer とターゲットグループを作成する
- ステップ 4: Amazon ECS クラスターとサービスを作成する
- ステップ 5: CodeDeploy アプリケーションとデプロイグループ (ECS コンピューティングプラットフォーム) を作成する

- ステップ 6: パイプラインを作成する
- ステップ 7: パイプラインに変更を加えてデプロイを確認する

前提条件

以下のリソースがすでに作成されている必要があります。

- CodeCommit リポジトリ。で作成した AWS CodeCommit リポジトリを使用できますチュートリアル: シンプルなパイプラインを作成する (CodeCommit リポジトリ)。
- このチュートリアルに示すように、Amazon EC2 Linux インスタンスを起動し、Docker をインストールしてイメージを作成します。使用するイメージがすでにある場合、この前提条件は省略できます。

ステップ 1: イメージを作成して Amazon ECR リポジトリにプッシュする

このセクションでは、Docker を使用してイメージを作成し、 を使用して Amazon ECR リポジトリ AWS CLI を作成し、そのイメージをリポジトリにプッシュします。

> **Note**
> 使用するイメージがすでにある場合、このスキップは省略できます。

イメージを作成するには

1. Docker がインストールされている Linux インスタンスにサインインします。

   nginx のイメージをプルダウンから選択します。このコマンドは nginx:latest イメージを返します。

   ```
   docker pull nginx
   ```

2. docker images を実行します。リストにイメージが表示されます.。

   ```
   docker images
   ```

Amazon ECR リポジトリを作成してイメージをプッシュするには

1. イメージを保存する Amazon ECR リポジトリを作成します。出力の `repositoryUri` を書き留めます。

```
aws ecr create-repository --repository-name nginx
```

出力:

```
{
    "repository": {
        "registryId": "aws_account_id",
        "repositoryName": "nginx",
        "repositoryArn": "arn:aws:ecr:us-east-1:aws_account_id:repository/nginx",
        "createdAt": 1505337806.0,
        "repositoryUri": "aws_account_id.dkr.ecr.us-east-1.amazonaws.com/nginx"
    }
}
```

2. 前のステップの `repositoryUri` でイメージをタグ付けします。

```
docker tag nginx:latest aws_account_id.dkr.ecr.us-east-1.amazonaws.com/nginx:latest
```

3. この例に示すように、us-west-2 リージョンと 111122223333 アカウント ID を指定して aws ecr get-login-password コマンドを実行します。

```
aws ecr get-login-password --region us-west-2 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-west-2.amazonaws.com/nginx
```

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

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/nginx:latest
```

ステップ 2: タスク定義ソースファイルと AppSpec ソースファイルを作成して、CodeCommit リポジトリにプッシュする

このセクションでは、Amazon ECS でタスク定義 JSON ファイルを作成します。次に、CodeDeploy 用の AppSpec ファイルを作成し、Git クライアントを使用してファイルを CodeCommit リポジトリにプッシュします。

イメージのタスク定義を作成するには

1. 次の内容で、taskdef.json という名前のファイルを作成します。image に、イメージの名前 (例: nginx) を入力します。この値は、パイプラインの実行時に更新されます。

> **Note**
>
> タスク定義に指定されている実行ロールに AmazonECSTaskExecutionRolePolicy が含まれていることを確認します。詳細については、Amazon ECS デベロッパーガイドの Amazon ECS タスク実行 IAM ロール を参照してください。

```
{
    "executionRoleArn": "arn:aws:iam::account_ID:role/ecsTaskExecutionRole",
    "containerDefinitions": [
        {
            "name": "sample-website",
            "image": "nginx",
            "essential": true,
            "portMappings": [
                {
                    "hostPort": 80,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ]
        }
    ],
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "networkMode": "awsvpc",
    "cpu": "256",
    "memory": "512",
    "family": "ecs-demo"
}
```

2. タスク定義を taskdef.json ファイルに登録します。

```
aws ecs register-task-definition --cli-input-json file://taskdef.json
```

3. タスク定義が登録されたら、イメージ名を削除して、イメージフィールド名に <IMAGE1_NAME> プレースホルダーが含まれるようにファイルを編集します。

```
{
    "executionRoleArn": "arn:aws:iam::account_ID:role/ecsTaskExecutionRole",
    "containerDefinitions": [
        {
            "name": "sample-website",
            "image": "<IMAGE1_NAME>",
            "essential": true,
            "portMappings": [
                {
                    "hostPort": 80,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ]
        }
    ],
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "networkMode": "awsvpc",
    "cpu": "256",
    "memory": "512",
    "family": "ecs-demo"
}
```

AppSpec ファイルを作成するには

- AppSpec ファイルは、CodeDeploy のデプロイに使用されます。オプションのフィールドが含まれるファイルでは、この形式を使用します。

```
version: 0.0
Resources:
  - TargetService:
      Type: AWS::ECS::Service
      Properties:
        TaskDefinition: "task-definition-ARN"
        LoadBalancerInfo:
          ContainerName: "container-name"
```

```
          ContainerPort: container-port-number
# Optional properties
        PlatformVersion: "LATEST"
        NetworkConfiguration:
            AwsvpcConfiguration:
              Subnets: ["subnet-name-1", "subnet-name-2"]
              SecurityGroups: ["security-group"]
              AssignPublicIp: "ENABLED"
Hooks:
- BeforeInstall: "BeforeInstallHookFunctionName"
- AfterInstall: "AfterInstallHookFunctionName"
- AfterAllowTestTraffic: "AfterAllowTestTrafficHookFunctionName"
- BeforeAllowTraffic: "BeforeAllowTrafficHookFunctionName"
- AfterAllowTraffic: "AfterAllowTrafficHookFunctionName"
```

例を含む AppSpec ファイルの詳細については、CodeDeploy AppSpec ファイルのリファレンスを参照してください。

次の内容で、appspec.yaml という名前のファイルを作成します。TaskDefinition の <TASK_DEFINITION> プレースホルダーテキストは変更しないでください。この値は、パイプラインの実行時に更新されます。

```
version: 0.0
Resources:
  - TargetService:
      Type: AWS::ECS::Service
      Properties:
        TaskDefinition: <TASK_DEFINITION>
        LoadBalancerInfo:
          ContainerName: "sample-website"
          ContainerPort: 80
```

ファイルを CodeCommit リポジトリにプッシュするには

1. ファイルを CodeCommit リポジトリにプッシュまたはアップロードします。このファイルは、CodePipeline でのデプロイアクションのために パイプラインの作成 のウィザードによって作成されたソースアーティファクトです。ファイルは、ローカルディレクトリに次のように表示されます。

```
/tmp
```

```
|my-demo-repo
  |-- appspec.yaml
  |-- taskdef.json
```

2. ファイルをアップロードする方法を選択します。

   a. ローカルコンピュータのクローンされたリポジトリから git コマンドを使用するには

      i. ディレクトリをローカルリポジトリに変更する:

      ```
      (For Linux, macOS, or Unix) cd /tmp/my-demo-repo
      (For Windows) cd c:\temp\my-demo-repo
      ```

      ii. 以下のコマンドを実行して、すべてのファイルを一度にステージングします。

      ```
      git add -A
      ```

      iii. 以下のコマンドを実行して、コミットメッセージによりファイルをコミットします。

      ```
      git commit -m "Added task definition files"
      ```

      iv. 以下のコマンドを実行して、ローカルリポジトリから CodeCommit リポジトリにファイルをプッシュします。

      ```
      git push
      ```

   b. CodeCommit コンソールを使用してファイルをアップロードするには:

      i. CodeCommit コンソールを開き、リポジトリ リストから自分のリポジトリを選択します。

      ii. [Add file]、[Upload file] の順に選択します。

      iii. [Choose file] を選択し、ファイルを参照します。ユーザー名とメールアドレスを入力して、変更をコミットします。[Commit changes] (変更のコミット) を選択します。

      iv. アップロードするファイルごとにこのステップを繰り返します。

ステップ 3: Application Load Balancer とターゲットグループを作成する

このセクションでは、Amazon EC2 Application Load Balancer を作成します。後で Amazon ECS サービスを作成するときに、ロードバランサーで作成したサブネット名とターゲットグループ値を使用します。Application Load Balancer または Network Load Balancer を作成できます。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 を確認して、ロードバランサーを作成してから、ロードバランサーの 2 つのターゲットグループを作成します。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 とパブリックサブネットを確認するには

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/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。デフォルトの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 VPC です。選択するデフォルトのサブネットが含まれています。
3. [サブネット] を選択します。デフォルトのサブネット 列に はい と表示されている 2 つのサブネットを選択します。

   > **Note**
   >
   > サブネット ID を書き留めます。これらは、このチュートリアルで後ほど必要になります。

4. サブネットを選択後、[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、ルートテーブルにゲートウェイ行が含まれていることを確認します。

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. ロードバランサー用に 2 つのリスナーポートを設定するには:

    a. [Load Balancer Protocol (ロードバランサーのプロトコル)] で、[HTTP] を選択します。[Load Balancer Port (ロードバランサーポート)] に [**80**] を入力します。

    b. [リスナーの追加] を選択します。

    c. 2 番目のリスナーの [Load Balancer Protocol] で、[HTTP] を選択します。[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 を選択します。次に、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] を選択し、セキュリティグループ ID を書き留めます。

13. [Next: Configure Routing] を選択します。

14. [Target group] で、[New target group] 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「**8080**」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**
>
> 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。最初のターゲットグループの ARN を書き留める必要があります。この ARN は、次のステップの `create-service` JSON ファイルで使用されます。

2 番目のターゲットグループを含めるようにロードバランサーを更新するには

1. Amazon EC2 コンソールを開きます。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。
2. ロードバランサーを選択後、[Listeners] を選択します。ポート 8080 のリスナーを選択後、[編集] を選択します。
3. [Forward to] の横の鉛筆アイコンを選択します。2 番目のターゲットグループを選択してから、チェックマークを選択します。[更新] を選択して、更新を保存します。

ステップ 4: Amazon ECS クラスターとサービスを作成する

このセクションでは、CodeDeploy がデプロイ中に (EC2インスタンスではなくAmazon ECS クラスターに) トラフィックをルーティングするAmazon ECS クラスターとサービスを作成します。Amazon ECS サービスを作成するには、ロードバランサーで作成したサブネット名、セキュリティグループ、ターゲットグループの値を使用してサービスを作成する必要があります。

> **Note**
>
> これらのステップを使用して Amazon ECS クラスターを作成する場合、Fargate コンテナをプロビジョニング AWS する Networking Only クラスターテンプレートを使用します。AWS Fargate は、コンテナインスタンスインフラストラクチャを管理するテクノロジーです。Amazon ECS クラスター用の Amazon EC2 インスタンスを手動で選択または作成する必要はありません。

Amazon ECS クラスターを作成するには

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 [Clusters] (クラスター) を選択してください。

3. [クラスターを作成] を選択します。
4. AWS Fargate を使用する ネットワークのみ クラスターテンプレートを選択後、次のステップを選択します。
5. [Configure cluster (クラスターの設定)] ページで、クラスター名を入力します。リソースに任意のタグを追加することができます。[作成] を選択します。

Amazon ECS サービスを作成する

AWS CLI を使用して、Amazon ECS でサービスを作成します。

1. JSON ファイルを作成し、`create-service.json` と名付けます。次の内容を JSON ファイルに貼り付けます。

   `taskDefinition`フィールドでは、Amazon ECS にタスク定義を登録するときに、ファミリーを指定します。これは、リビジョン番号で指定された、複数バージョンのタスク定義の名前に似ています。この例では、ファイル内のファミリーとリビジョン番号に「`ecs-demo:1`」を使用します。ステップ 3: Application Load Balancer とターゲットグループを作成する でロードバランサーを使用して作成したサブネット名、セキュリティグループ、ターゲットグループの値を使用します。

   > **Note**
   >
   > ターゲットグループ ARN をこのファイルに含める必要があります。Amazon EC2 コンソールを開き、ナビゲーションペインの ロードバランシング で ターゲットグループ を選択します。最初のターゲットグループを選択します。[説明] タブから ARN をコピーします。

```
{
    "taskDefinition": "family:revision-number",
    "cluster": "my-cluster",
    "loadBalancers": [
        {
            "targetGroupArn": "target-group-arn",
            "containerName": "sample-website",
            "containerPort": 80
        }
    ],
    "desiredCount": 1,
```

```
    "launchType": "FARGATE",
    "schedulingStrategy": "REPLICA",
    "deploymentController": {
        "type": "CODE_DEPLOY"
    },
    "networkConfiguration": {
        "awsvpcConfiguration": {
            "subnets": [
                "subnet-1",
                "subnet-2"
            ],
            "securityGroups": [
                "security-group"
            ],
            "assignPublicIp": "ENABLED"
        }
    }
}
```

2. JSON ファイルを指定して、create-service コマンドを実行します。

> **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。

この例では、`my-service` という名前のサービスが作成されます。

> **Note**
>
> このコマンド例では、my-service という名前のサービスを作成します。この名前のサービスがすでにある場合、このコマンドはエラーを返します。

```
aws ecs create-service --service-name my-service --cli-input-json file://create-service.json
```

出力はサービスの説明フィールドが返ります。

3. describe-services コマンドを実行して、サービスが作成されたことを確認します。

```
aws ecs describe-services --cluster cluster-name --services service-name
```

ステップ 5: CodeDeploy アプリケーションとデプロイグループ (ECS コンピューティングプラットフォーム) を作成する

Amazon ECS コンピューティングプラットフォーム用の CodeDeploy アプリケーションとデプロイグループを作成すると、アプリケーションはデプロイ中に使用され、正しいデプロイグループ、ターゲットグループ、リスナー、およびトラフィックの再ルーティング動作を参照します。

CodeDeploy でアプリケーションを作成する。

1. CodeDeploy コンソールを開き、アプリケーションの作成 を選択します。
2. [アプリケーション名] に、使用する名前を入力します。
3. [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。
4. [アプリケーションを作成] を選択します。

次に、CodeDeploy デプロイ グループを作成します。

1. アプリケーションページの [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. [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。
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。新しいサービスロールを作成するには、次の手順を実行します。

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

    b. コンソールダッシュボードで [ロール] を選択します。

    c. [ロールの作成] を選択してください。

    d. [信頼されたエンティティのタイプを選択] で、[AWS のサービス] を選択します。ユースケースの選択 で、CodeDeploy を選択します。[ユースケースの選択] で、[CodeDeploy - ECS] を選択します。[Next: Permissions] (次のステップ: 許可) を選択します。AWSCodeDeployRoleForECS マネージドポリシーはロールにアタッチ済みです。

    e. [次の手順: タグ]、[次の手順: 確認] の順に選択します。

    f. ロールの名前 (例: **CodeDeployECSRole**) を入力し、[ロールの作成] を選択します。

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。[Test listener port] で、テストリスナーのポートとプロトコルを選択します。
7. [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。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。
8. すぐにトラフィックを再ルーティング を選択して、デプロイが成功してから更新された Amazon ECS タスクにトラフィックを再ルーティングするまでの時間を決定します。
9. デプロイグループの作成 を選択します。

ステップ 6: パイプラインを作成する

このセクションでは、以下のアクションを使用してパイプラインを作成します。

- ソースアーティファクトがタスク定義と AppSpec ファイルである CodeCommit アクション。
- ソースアーティファクトがイメージファイルである Amazon ECR ソースアクションを持つソースステージ。
- デプロイが CodeDeploy アプリケーションとデプロイグループで実行される、Amazon ECS デプロイアクションを使用したデプロイステージ。

ウィザードで 2 ステージパイプラインを作成するには

1. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。
3. [ステップ 1: 作成オプションを選択する] ページの [作成オプション] で、[カスタムパイプラインを構築する] オプションを選択します。[次へ] を選択します。
4. [ステップ 2: パイプラインの設定を選択する] で、[パイプライン名] に「**MyImagePipeline**」と入力します。

5. CodePipeline は、特徴と料金が異なる V1 タイプと V2 タイプのパイプラインを提供しています。V2 タイプは、コンソールで選択できる唯一のタイプです。詳細については、「パイプラインタイプ」を参照してください。CodePipeline の料金については、料金を参照してください。
6. サービスロール で、新しいサービスロール を選択して、CodePipeline が IAM でサービスロールを作成できるようにします。
7. [詳細設定] をデフォルト設定のままにし、[次へ] を選択します。
8. [ステップ 3: ソースステージの追加] の [ソースプロバイダー] で、[AWS CodeCommit] を選択します。リポジトリ名 で、ステップ 1: CodeCommit リポジトリを作成する で作成した CodeCommit リポジトリの名前を選択します。[Branch name] で、最新のコード更新を含むブランチの名前を選択します。

   [次へ] を選択します。
9. [ステップ 4: ビルドステージの追加] で、[ビルドステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。[次へ] を選択します。
10. [ステップ 5: テストステージの追加] で、[テストステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

    [次へ] を選択します。
11. ステップ 6: デプロイステージの追加:

    a. [Deploy provider] で、[Amazon ECS (Blue/Green)] を選択します。[アプリケーション名] で、`codedeployapp` などの、アプリケーションの名前を入力またはリストから選択します。[デプロイグループ] に、`codedeploydeplgroup` などの、デプロイグループの名前を入力またはリストから選択します。

    > **Note**
    >
    > 「デプロイ」は、[ステップ 4: デプロイ] ステップで作成されるステージにデフォルトで付けられる名前であり、「ソース」は、パイプラインの最初のステージに付けられる名前です。

    b. [Amazon ECS タスク定義] で、[SourceArtifact] を選択します。フィールドに [**`taskdef.json`**] と入力します。

    c. AWS CodeDeploy AppSpec ファイル で、SourceArtifact を選択します。フィールドに [**`appspec.yaml`**] と入力します。

> **Note**
>
> この時点では、[Dynamically update task definition image] に情報は入力しないでください。

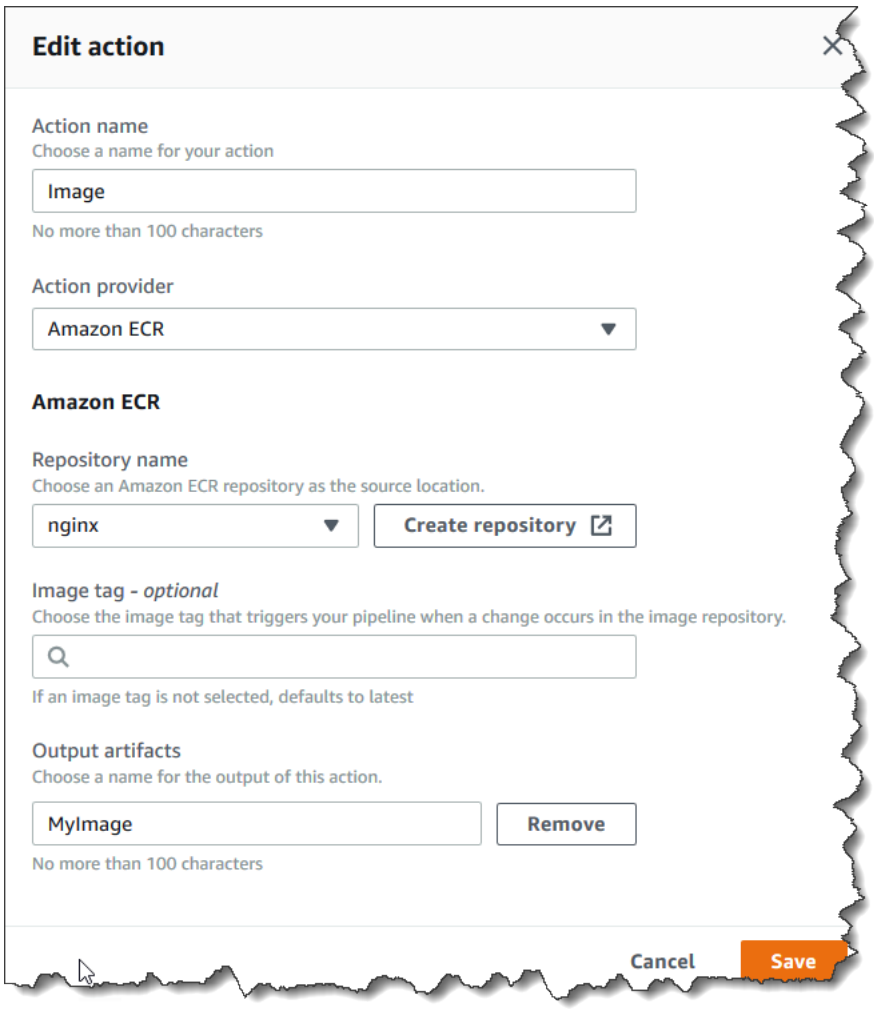
d. [次へ] を選択します。

12. ステップ 7: 情報を確認してから、パイプラインの作成を選択します。

Amazon ECR ソースアクションをパイプラインに追加するには

パイプラインを表示し、Amazon ECR ソースアクションをパイプラインに追加します。

1. パイプラインを選択します。左上の [Edit] (編集) を選択します。
2. ソースステージで、[ステージを編集] を選択します。
3. CodeCommitソースアクションの横にある +アクションの追加 を選択して、並列アクションを追加します。
4. [アクション名] に、名前を入力します (例えば、**`Image`**)。
5. [アクションプロバイダ] で [Amazon ECR] を選択します。



6. リポジトリ名 で、Amazon ECR リポジトリの名前を選択します。
7. [Image tag] で、イメージの名前とバージョンを指定します (最新でない場合)。
8. 出力アーティファクトで、次のステージで使用するイメージ名およびリポジトリ URI 情報が含まれている出力アーティファクトのデフォルト(例: `MyImage`)を選択します。
9. アクション画面で、[Save] を選択します。ステージ画面で、[Done] を選択します。パイプラインで、[Save] を選択します。メッセージは、Amazon ECR ソースアクション用に作成される Amazon CloudWatch Events ルールを示しています。

ソースアーティファクトをデプロイアクションに関連付けるには

1. デプロイステージで 編集 を選択後、アイコンを選択して、Amazon ECS (Blue/Green) アクションを編集します。
2. ペインの下部までスクロールします。[入力アーティファクト] で [Add] を選択します。新しい Amazon ECR リポジトリ (例: `MyImage` など) からソースアーティファクトを追加します。

3. [タスク定義] で [SourceArtifact] を選択し、「**taskdef.json**」と入力されていることを確認します。
4. AWS CodeDeploy AppSpec ファイル で、SourceArtifact を選択し、**appspec.yaml**と入力されていることを確認します。
5. [Dynamically update task definition image] の [Input Artifact with Image URI] で、[MyImage] を選択し、`taskdef.json` ファイルで使用されているプレースホルダー「 **IMAGE1_NAME**」を入力します。[保存] を選択します。
6. AWS CodePipeline ペインで、パイプラインの変更の保存を選択し、変更の保存を選択します。更新されたパイプラインを表示します。

   このサンプルパイプラインが作成されると、コンソールエントリのアクション設定が以下のようにパイプライン構造に表示されます。

```
"configuration": {
  "AppSpecTemplateArtifact": "SourceArtifact",
  "AppSpecTemplatePath": "appspec.yaml",
  "TaskDefinitionTemplateArtifact": "SourceArtifact",
  "TaskDefinitionTemplatePath": "taskdef.json",
  "ApplicationName": "codedeployapp",
  "DeploymentGroupName": "codedeploydeplgroup",
  "Image1ArtifactName": "MyImage",
  "Image1ContainerName": "IMAGE1_NAME"
},
```

7. 変更を送信してパイプラインのビルドを開始するには、[変更をリリース]、[リリース] の順に選択します。
8. デプロイアクションを選択して CodeDeploy で表示し、トラフィックシフトの進捗状況を確認します。

> **Note**
>
> オプションの待機時間を示すデプロイステップが表示される場合があります。デフォルトで、CodeDeploy はデプロイが成功してから 1 時間後に元のタスク設定を終了します。この時間を使用してタスクをロールバックまたは終了することはできますが、それ以外の場合、タスクセットが終了した時点でデプロイは完了します。

ステップ 7: パイプラインに変更を加えてデプロイを確認する

イメージを変更して、その変更を Amazon ECR リポジトリにプッシュします。これにより、パイプラインの実行がトリガーされます。イメージソースの変更がデプロイされていることを確認します。

チュートリアル: Amazon Alexa Skill をデプロイするパイプラインを作成する

このチュートリアルでは、デプロイステージでデプロイプロバイダとして Alexa Skills Kit を使用して Alexa スキルを継続的にデリバリーするパイプラインを設定します。ソースリポジトリのソースファイルに変更を加えると、完成したパイプラインはスキルの変更を検出します。次に、パイプラインは Alexa Skills Kit を使用して、その変更を Alexa スキル開発ステージにデプロイします。

> **Important**
>
> パイプライン作成の一環として、CodePipeline は、ユーザーが指定した S3 アーティファクトバケットをアーティファクトとして使用します (これは S3 ソースアクションで使用するバケットとは異なります)。S3 アーティファクトバケットがパイプラインのアカウントとは異なるアカウントにある場合は、S3 アーティファクトバケットが によって所有 AWS アカウント されており、安全で信頼できることを確認してください。

> **Note**
>
> この特徴は、アジアパシフィック (香港) またはヨーロッパ (ミラノ) リージョンでは使用できません。当該地域で使用可能な他のデプロイアクションを使用する場合、デプロイアクションの統合 を参照してください。

カスタムスキルを Lambda 関数として作成するには、「カスタムスキルを AWS Lambda 関数としてホストする」を参照してください。Lambda ソースファイルと CodeBuild プロジェクトを使用して、スキルに合わせて変更を Lambda にデプロイするパイプラインを作成することもできます。

前提条件

以下のものを用意しておく必要があります。

- CodeCommit リポジトリ。で作成した AWS CodeCommit リポジトリを使用できますチュートリアル: シンプルなパイプラインを作成する (CodeCommit リポジトリ)。
- Amazon 開発者アカウント。これは Alexa スキルを所有するアカウントです。Alexa Skills Kit でアカウントを無料で作成できます。
- Alexa スキル。「カスタムスキルサンプルコードを取得する」チュートリアルを使用してサンプルスキルを作成できます。
- ASK CLI をインストールし、`ask init` 認証情報で AWS を使用して設定します。「ASK CLI のインストールと初期化」を参照してください。

ステップ 1: Alexa デベロッパーサービス LWA セキュリティプロファイルを作成する

このセクションでは、Login with Amazon (LWA) で使用するセキュリティプロファイルを作成します。プロファイルがすでにある場合、このステップは省略できます。

- 「generate-lwa-tokens」の手順を使用して、セキュリティプロファイルを作成します。
- プロファイルを作成したら、[クライアント ID] と [クライアントシークレット] の値をメモしておきます。
- それらの手順に従って [Allowed Return URLs (許可されたリターン URL)] に入力します。これらの URL を ASK CLI コマンドで使用して、更新トークンリクエストをリダイレクトできます。

ステップ 2 : Alexa スキルのソースファイルを作成して CodeCommit リポジトリにプッシュします。

このセクションでは、Alexa スキルのソースファイルを作成し、パイプラインによってソースステージに使用されるリポジトリにプッシュします。Amazon 開発者コンソールで作成したスキル用に、以下のものを作成してプッシュします。

- `skill.json` ファイル。
- `interactionModel/custom` フォルダ。

> **Note**
>
> このディレクトリ構造は、「スキルパッケージ形式」で説明されているように、Alexa Skills Kit スキルパッケージ形式の要件に準拠しています。ディレクトリ構造で正しいスキ

ルパッケージ形式が使用されていない場合、変更は Alexa Skills Kit コンソールに正常にデプロイされません。

スキルのソースファイルを作成するには

1. Alexa Skills Kit 開発者コンソールからスキル ID を取得します。以下のコマンドを使用します。

   ```
   ask api list-skills
   ```

   名前に基づいてスキルを見つけ、`skillId` フィールドで、関連付けられた ID をコピーします。

2. スキルの詳細を含む `skill.json` ファイルを生成します。以下のコマンドを使用します。

   ```
   ask api get-skill -s skill-ID > skill.json
   ```

3. (オプション) `interactionModel/custom` フォルダを作成します。

   以下のコマンドを使用して、フォルダ内にインタラクションモデルファイルを生成します。locale について、このチュートリアルではファイル名の口ケールとして en-US を使用します。

   ```
   ask api get-model --skill-id skill-ID --locale locale >
       ./interactionModel/custom/locale.json
   ```

ファイルを CodeCommit リポジトリにプッシュするには

1. ファイルを CodeCommit リポジトリにプッシュまたはアップロードします。これらのファイルは、 AWS CodePipelineでのデプロイアクションのために パイプライン作成 ウィザードによって作成されたソースアーティファクトです。ファイルは、ローカルディレクトリに次のように表示されます。

   ```
   skill.json
   /interactionModel
     /custom
       |en-US.json
   ```

2. ファイルをアップロードする方法を選択します。

a. ローカルコンピュータで複製されたリポジトリから Git コマンドラインを使用するには:

i. 以下のコマンドを実行して、すべてのファイルを一度にステージングします。

```
git add -A
```

ii. 以下のコマンドを実行して、コミットメッセージによりファイルをコミットします。

```
git commit -m "Added Alexa skill files"
```

iii. 以下のコマンドを実行して、ローカルリポジトリから CodeCommit リポジトリにファイルをプッシュします。

```
git push
```

b. CodeCommit コンソールを使用してファイルをアップロードするには:

i. CodeCommit コンソールを開き、リポジトリ リストから自分のリポジトリを選択します。

ii. [Add file]、[Upload file] の順に選択します。

iii. [Choose file] を選択し、ファイルを参照します。ユーザー名とメールアドレスを入力して、変更をコミットします。[Commit changes] (変更のコミット) を選択します。

iv. アップロードするファイルごとにこのステップを繰り返します。

ステップ 3: ASK CLI コマンドを使用して更新トークンを作成する

CodePipeline は、Amazon デベロッパーアカウントのクライアント ID とシークレットに基づく更新トークンを使用して、お客様の代わりに実行するアクションを認可します。このセクションでは、ASK CLI を使用してトークンを作成します。[パイプラインを作成する] ウィザードでこれらの認証情報を使用します。

Amazon 開発者アカウントの認証情報を使用して更新トークンを作成するには

1. 以下のコマンドを使用します。

```
ask util generate-lwa-tokens
```

2. プロンプトが表示されたら、以下の例に示すようにクライアント ID とシークレットを入力します。

```
? Please type in the client ID:
amzn1.application-client.example112233445566
? Please type in the client secret:
example112233445566
```

3. サインインのブラウザページが表示されます。Amazon アカウント認証情報を使用してサインインします。
4. コマンドライン画面に戻ります。アクセストークンと更新トークンが出力に生成されます。出力に返された更新トークンをコピーします。

ステップ 4: パイプラインを作成する

このセクションでは、以下のアクションを使用してパイプラインを作成します。

- ソースアーティファクトがスキルをサポートする Alexa スキルファイルである CodeCommit アクションを含むソースステージ。
- Alexa Skills Kit のデプロイアクションを含むデプロイステージ。

ウィザードを使用してパイプラインを作成するには

1. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. プロジェクトとそのリソースを作成する AWS リージョンを選択します。Alexa スキルランタイムは、以下のリージョンでのみ利用できます。
   - アジアパシフィック (東京)
   - 欧州 (アイルランド)
   - 米国東部 (バージニア北部)
   - 米国西部 (オレゴン)
3. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。
4. [ステップ 1: 作成オプションを選択する] ページの [作成オプション] で、[カスタムパイプラインを構築する] オプションを選択します。[次へ] を選択します。

5. [ステップ 2: パイプラインの設定を選択する] で、[パイプライン名] に「**MyAlexaPipeline**」と入力します。
6. CodePipeline は、特徴と料金が異なる V1 タイプと V2 タイプのパイプラインを提供しています。V2 タイプは、コンソールで選択できる唯一のタイプです。詳細については、「パイプラインタイプ」を参照してください。CodePipeline の料金については、料金を参照してください。
7. サービスロール で、新しいサービスロール を選択して、CodePipeline が IAM でサービスロールを作成できるようにします。
8. [詳細設定] をデフォルト設定のままにし、[次へ] を選択します。
9. [ステップ 3: ソースステージの追加] の [ソースプロバイダー] で、[AWS CodeCommit] を選択します。リポジトリ名 で、ステップ 1: CodeCommit リポジトリを作成する で作成した CodeCommit リポジトリの名前を選択します。[Branch name] で、最新のコード更新を含むブランチの名前を選択します。

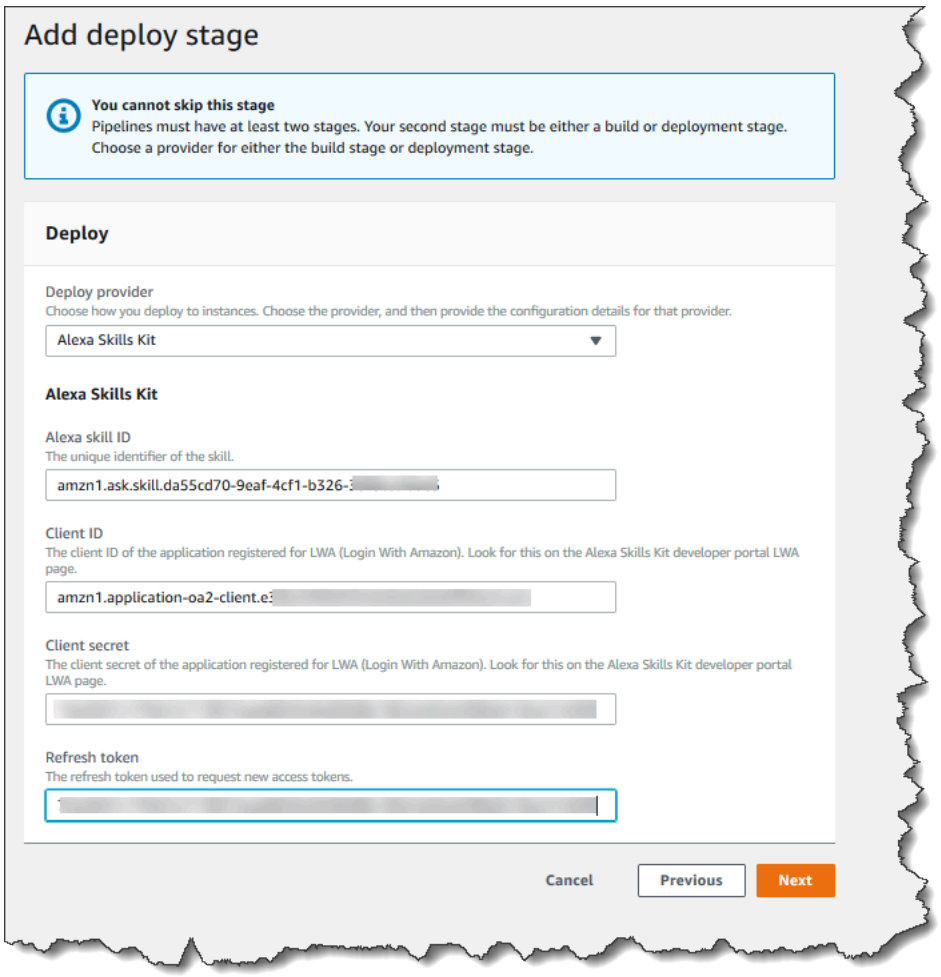
   リポジトリ名とブランチを選択した後、このパイプラインのために作成される Amazon CloudWatch Events ルールを示すメッセージが表示されます。

   [次へ] を選択します。
10. [ステップ 4: ビルドステージの追加] で、[ビルドステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

    [次へ] を選択します。
11. [ステップ 5: テストステージの追加] で、[テストステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

    [次へ] を選択します。
12. ステップ 6: デプロイステージの追加:

    a. [デプロイプロバイダ] で、[Alexa Skills Kit] を選択します。
    b. [Alexa skill ID (Alexa スキル ID)] に、Alexa Skills Kit 開発者コンソールでスキルに割り当てられているスキル ID を入力します。
    c. [クライアント ID] に、登録したアプリケーションの ID を入力します。
    d. [Client secret (クライアントシークレット)] に、登録時に選択したシークレットを入力します。
    e. [Refresh token (更新トークン)] に、ステップ 3 で生成したトークンを入力します。



f. [次へ] を選択します。

13. ステップ 7: 情報を確認してから、パイプラインの作成を選択します。

ステップ 5: 任意のソースファイルに変更を加えてデプロイを確認する

スキルに変更を加え、その変更をリポジトリにプッシュします。これにより、パイプラインの実行がトリガーされます。スキルが Alexa Skills Kit 開発者コンソールで更新されていることを確認します。

チュートリアル: Amazon S3 をデプロイプロバイダとして使用するパイプラインを作成する

このチュートリアルでは、デプロイステージでデプロイアクションプロバイダーとして Amazon S3 を使用して、ファイルを継続的に配信するパイプラインを設定します。ソースリポジトリ内のソー

スファイルに変更を加えると、完成したパイプラインはその変更を検出します。パイプラインは Amazon S3 を使用してそれらのファイルをバケットにデプロイします。ソースの場所で Web サイトのファイルを変更または追加するたびに、デプロイで最新のファイルを使用して Web サイトが作成されます。

> **Important**
>
> パイプライン作成の一環として、CodePipeline は、ユーザーが指定した S3 アーティファクトバケットをアーティファクトとして使用します (これは S3 ソースアクションで使用するバケットとは異なります)。S3 アーティファクトバケットがパイプラインのアカウントとは異なるアカウントにある場合は、S3 アーティファクトバケットが によって所有 AWS アカウント されており、安全で信頼できることを確認してください。

> **Note**
>
> ソースリポジトリからファイルを削除しても、S3 デプロイアクションでは、削除されたファイルに対応する S3 オブジェクトは削除されません。

このチュートリアルには 2 つのオプションがあります。

- 静的ウェブサイトを S3 パブリックバケットにデプロイするパイプラインを作成する。この例では、 AWS CodeCommit ソースアクションと Amazon S3 デプロイアクションを使用してパイプラインを作成します。「オプション 1: 静的ウェブサイトファイルを Amazon S3 にデプロイする」を参照してください。
- サンプル TypeScript コードを JavaScript にコンパイルし、CodeBuild 出力アーティファクトをアーカイブ用の S3 バケットにデプロイするパイプラインを作成します。この例では、Amazon S3 ソースアクション、CodeBuild 構築アクション、Amazon S3 デプロイアクションを使用するパイプラインを作成します。「オプション 2 : 構築されたアーカイブファイルを S3 ソースバケットから Amazon S3 にデプロイする」を参照してください。

> **Important**
>
> この手順でパイプラインに追加するアクションの多くには、パイプラインを作成する前に作成する必要がある AWS リソースが含まれます。ソースアクションの AWS リソースは常に、パイプラインを作成するのと同じ AWS リージョンで作成する必要があります。例え

ば、米国東部 (オハイオ) リージョンにパイプラインを作成している場合、CodeCommit リポジトリは米国東部 (オハイオ) リージョンにある必要があります。
パイプラインの作成時にクロスリージョンアクションを追加できます。クロスリージョンアクションの AWS リソースは、アクションを実行する予定のリージョンと同じ AWS リージョンに存在する必要があります。詳細については、「CodePipeline でクロスリージョンアクションを追加する」を参照してください。

オプション 1: 静的ウェブサイトファイルを Amazon S3 にデプロイする

この例では、サンプルの静的ウェブサイトテンプレートファイルをダウンロードし、 AWS CodeCommit リポジトリにファイルをアップロードしてバケットを作成し、ホスティング用に設定します。次に、 AWS CodePipeline コンソールを使用してパイプラインを作成し、Amazon S3 デプロイ設定を指定します。

前提条件

以下のものを用意しておく必要があります。

- CodeCommit リポジトリ。で作成した AWS CodeCommit リポジトリを使用できますチュートリアル: シンプルなパイプラインを作成する (CodeCommit リポジトリ)。
- 静的ウェブサイトのソースファイル。このリンクを使用して サンプル静的ウェブサイトをダウンロードします。sample-website.zip をダウンロードすると、以下のファイルが生成されます。
  - `index.html` ファイル
  - `main.css` ファイル
  - `graphic.jpg` ファイル
- ウェブサイトホスティング用に設定された S3 バケット。「静的ウェブサイトを Amazon S3 でホスティングする」を参照してください。必ずパイプラインと同じリージョンにバケットを作成します。

> **Note**
>
> ウェブサイトのホスティングには、バケットへのパブリック読み取りアクセスを許可するアクセス設定が必要です。ウェブサイトのホスティングを除き、S3 バケットへのパブリックアクセスをブロックするデフォルトのアクセス設定を維持してください。

ステップ 1: ソースファイルを CodeCommit リポジトリにプッシュする

このセクションでは、パイプラインによってソースステージに使用されるリポジトリに、ソースファイルをプッシュします。

ファイルを CodeCommit リポジトリにプッシュするには

1. ダウンロードしたサンプルファイルを解凍します。ZIP ファイルをリポジトリにアップロードしないでください。
2. ファイルを CodeCommit リポジトリにプッシュまたはアップロードします。このファイルは、CodePipeline でのデプロイアクションのために パイプラインの作成 ウィザードで作成したソースアーティファクトです。ファイルは、ローカルディレクトリに次のように表示されます。

```
index.html
main.css
graphic.jpg
```

3. Git または CodeCommit コンソールを使用してファイルをアップロードできます。

   a. ローカルコンピュータで複製されたリポジトリから Git コマンドラインを使用するには:

      i. 以下のコマンドを実行して、すべてのファイルを一度にステージングします。

```
git add -A
```

      ii. 以下のコマンドを実行して、コミットメッセージによりファイルをコミットします。

```
git commit -m "Added static website files"
```

      iii. 以下のコマンドを実行して、ローカルリポジトリから CodeCommit リポジトリにファイルをプッシュします。

```
git push
```

   b. CodeCommit コンソールを使用してファイルをアップロードするには:

      i. CodeCommit コンソールを開き、リポジトリ リストから自分のリポジトリを選択します。

      ii. [Add file]、[Upload file] の順に選択します。

iii. [ファイルの選択] を選択し、ファイルを参照します。ユーザー名とメールアドレスを入力して、変更をコミットします。[Commit changes] (変更のコミット) を選択します。

iv. アップロードするファイルごとにこのステップを繰り返します。

ステップ 2: パイプラインを作成する

このセクションでは、次のアクションを使用してパイプラインを作成します。

- ソースアーティファクトが Web サイトのファイルである CodeCommit アクションを使用したソースステージ。
- Amazon S3 デプロイメントアクションを使用したデプロイメントステージ。

ウィザードを使用してパイプラインを作成するには

1. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。
3. [ステップ 1: 作成オプションを選択する] ページの [作成オプション] で、[カスタムパイプラインを構築する] オプションを選択します。[次へ] を選択します。
4. [ステップ 2: パイプラインの設定を選択する] で、[パイプライン名] に「**MyS3DeployPipeline**」と入力します。
5. [パイプラインタイプ] で、[V2] を選択します。詳細については、「パイプラインのタイプ」を参照してください。[Next] (次へ) を選択します。
6. サービスロール で、新しいサービスロール を選択して、CodePipeline が IAM でサービスロールを作成できるようにします。
7. [詳細設定] をデフォルト設定のままにし、[次へ] を選択します。
8. [ステップ 3: ソースステージの追加] の [ソースプロバイダー] で、[AWS CodeCommit] を選択します。リポジトリ名 で、ステップ 1: CodeCommit リポジトリを作成する で作成した CodeCommit リポジトリの名前を選択します。[Branch name] で、最新のコード更新を含むブランチの名前を選択します。独自のブランチを作成する場合を除き、ここで使用できるのは `main` のみです。

   リポジトリ名とブランチを選択した後、このパイプライン用に作成される Amazon CloudWatch Events ルールが表示されます。

[次へ] を選択します。

9. [ステップ 4: ビルドステージの追加] で、[ビルドステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

   [次へ] を選択します。

10. [ステップ 5: テストステージの追加] で、[テストステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

    [次へ] を選択します。

11. ステップ 6: デプロイステージの追加:

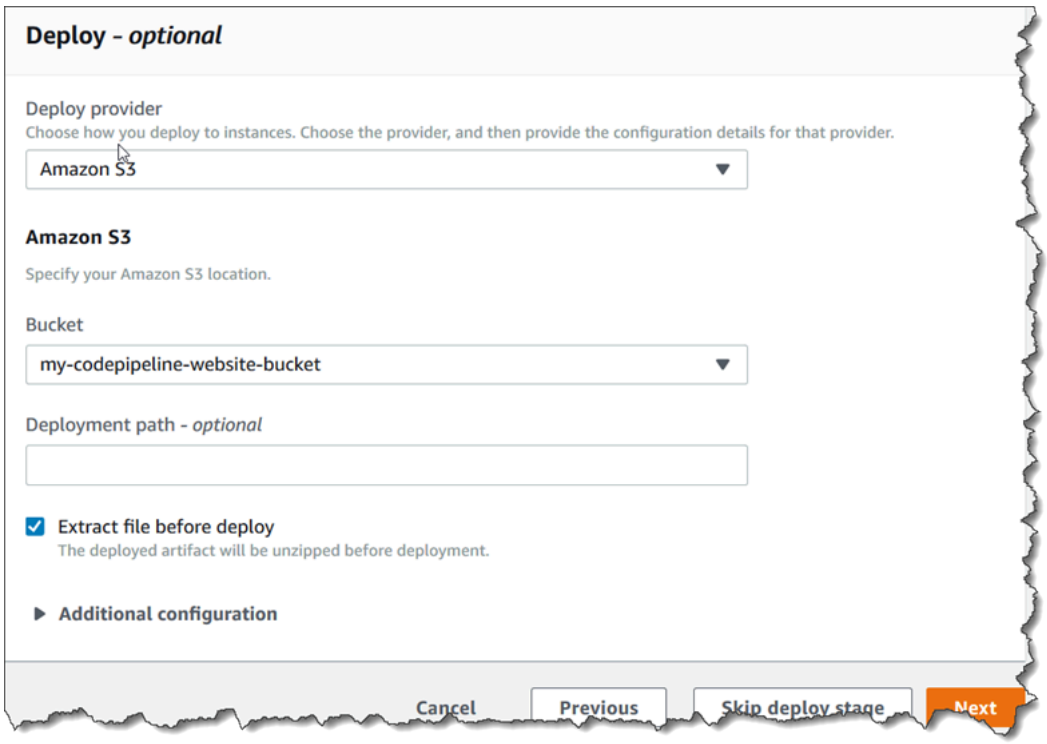
    a. [デプロイプロバイダ] で、[Amazon S3] を選択します。

    b. [バケット] にパブリックバケットの名前を入力します。

    c. [Extract file before deploy (デプロイ前にファイルを展開)] を選択します。

    > **Note**
    >
    > [デプロイ前にファイルを抽出] を選択しないと、デプロイに失敗します。これは、パイプラインの AWS CodeCommit アクションがソースアーティファクトを圧縮し、ファイルが ZIP ファイルであるためです。

    [Extract file before deploy (デプロイ前にファイルを展開)] を選択すると、[Deployment path (デプロイパス)] が表示されます。使用するパスの名前を入力します。これにより、ファイルが展開されるフォルダ構造が Amazon S3 に抽出されます。このチュートリアルでは、このフィールドを空欄にします。



d. (オプション) [既定 ACL] で、既定 ACL と呼ばれる、あらかじめ定義された一連の許可を、アップロードされたアーティファクトに適用できます。

e. (オプション) [キャッシュコントロール] で、キャッシュパラメータを入力します。これを設定して、リクエスト/レスポンスのキャッシュ動作を制御できます。有効な値については、HTTP オペレーションの `Cache-Control` ヘッダーフィールドを参照してください。

f. [次へ] を選択します。

12. ステップ 7: 情報を確認してから、パイプラインの作成を選択します。

13. パイプラインが正常に実行されたら、Amazon S3 コンソールを開き、ファイルが公開バケットに表示されていることを確認します。

```
index.html
main.css
graphic.jpg
```

14. エンドポイントにアクセスしてウェブサイトをテストします。エンドポイントは `http://`*`bucket-name`*`.s3-website-`*`region`*`.amazonaws.com/` という形式に従います。

    エンドポイントの例: `http://my-bucket.s3-website-us-west-2.amazonaws.com/`

    サンプルのウェブページが表示されます。

ステップ 3: 任意のソースファイルに変更を加えてデプロイを確認する

ソースファイルに変更を加え、その変更をリポジトリにプッシュします。これにより、パイプラインの実行がトリガーされます。ウェブサイトが更新されていることを確認します。

オプション 2：構築されたアーカイブファイルを S3 ソースバケットから Amazon S3 にデプロイする

このオプションでは、ビルドステージのビルドコマンドが TypeScript コードを JavaScript コードとしてコンパイルし、出力を別のタイムスタンプ付きフォルダの下の S3 ターゲットバケットにデプロイします。まず、TypeScript コードと buildspec.yml ファイルを作成します。ソースファイルを ZIP ファイルに結合した後、ソース ZIP ファイルを S3 ソースバケットにアップロードし、CodeBuild ステージを使用して構築されたアプリケーション ZIP ファイルを S3 ターゲットバケットにデプロイします。コンパイルされたコードはアーカイブとしてターゲットバケットに保存されます。

前提条件

以下のものを用意しておく必要があります。

- S3 ソースバケット。「チュートリアル: シンプルなパイプラインを作成する (S3 バケット)」で作成したバケットを使用できます。
- S3 ターゲットバケット。「静的ウェブサイトを Amazon S3 でホスティングする」を参照してください。バケットは、作成するパイプライン AWS リージョン と同じ に作成してください。

> **Note**
>
> この例では、ファイルをプライベートバケットにデプロイする方法を示します。ウェブサイトのホスティング用にターゲットバケットを有効にしたり、バケットを公開するポリシーをアタッチしたりしないでください。

ステップ 1: ソースファイルを作成して S3 ソースバケットにアップロードする

このセクションでは、ソースファイルを作成し、パイプラインによってソースステージに使用されるバケットにプッシュします。このセクションでは、以下のソースファイルを作成する手順について説明します。

- CodeBuild 構築プロジェクトに使用される `buildspec.yml` ファイル。
- `index.ts` ファイル。

buildspec.yml ファイルを作成するには

- 次の内容で、buildspec.yml という名前のファイルを作成します。これらのビルドコマンドは TypeScript をインストールし、TypeScript コンパイラを使用して index.ts のコードを JavaScript コードに書き換えます。

```
version: 0.2

phases:
  install:
    commands:
      - npm install -g typescript
  build:
    commands:
      - tsc index.ts
artifacts:
  files:
    - index.js
```

index.ts ファイルを作成するには

- 次の内容で、index.ts という名前のファイルを作成します。

```
interface Greeting {
    message: string;
}

class HelloGreeting implements Greeting {
    message = "Hello!";
}

function greet(greeting: Greeting) {
    console.log(greeting.message);
}

let greeting = new HelloGreeting();

greet(greeting);
```

ファイルを S3 ソースバケットにアップロードするには

1. ファイルは、ローカルディレクトリに次のように表示されます。

   ```
   buildspec.yml
   index.ts
   ```

   ファイルを圧縮して、ファイルに `source.zip` という名前を付けます。
2. Amazon S3 コンソールで、ソースバケット用に アップロード を選択します。[ファイルを追加] を選択し、作成した ZIP ファイルを参照します。
3. アップロード を選択します。このファイルは、CodePipeline でのデプロイアクションのために パイプラインの作成 ウィザードで作成したソースアーティファクトです。ファイルはバケットで以下のようになっています。

   ```
   source.zip
   ```

ステップ 2: パイプラインを作成する

このセクションでは、次のアクションを使用してパイプラインを作成します。

- ソースアーティファクトがダウンロード可能なアプリケーションのファイルである Amazon S3 アクションを含むソースステージ。
- Amazon S3 デプロイメントアクションを使用したデプロイメントステージ。

ウィザードを使用してパイプラインを作成するには

1. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。
3. [ステップ 1: 作成オプションを選択する] ページの [作成オプション] で、[カスタムパイプラインを構築する] オプションを選択します。[次へ] を選択します。
4. [ステップ 2: パイプラインの設定を選択する] で、[パイプライン名] に「**MyS3DeployPipeline**」と入力します。
5. サービスロール で、新しいサービスロール を選択して、CodePipeline が IAM でサービスロールを作成できるようにします。

6. [詳細設定] をデフォルト設定のままにし、[次へ] を選択します。
7. [ステップ 3: ソースステージの追加] の [ソースプロバイダー] で、[Amazon S3] を選択します。[バケット] で、ソースバケットの名前を選択します。[S3 object key (S3 オブジェクトキー)] に、ソース ZIP ファイルの名前を入力します。必ず、ファイル拡張子.zip を含めてください。

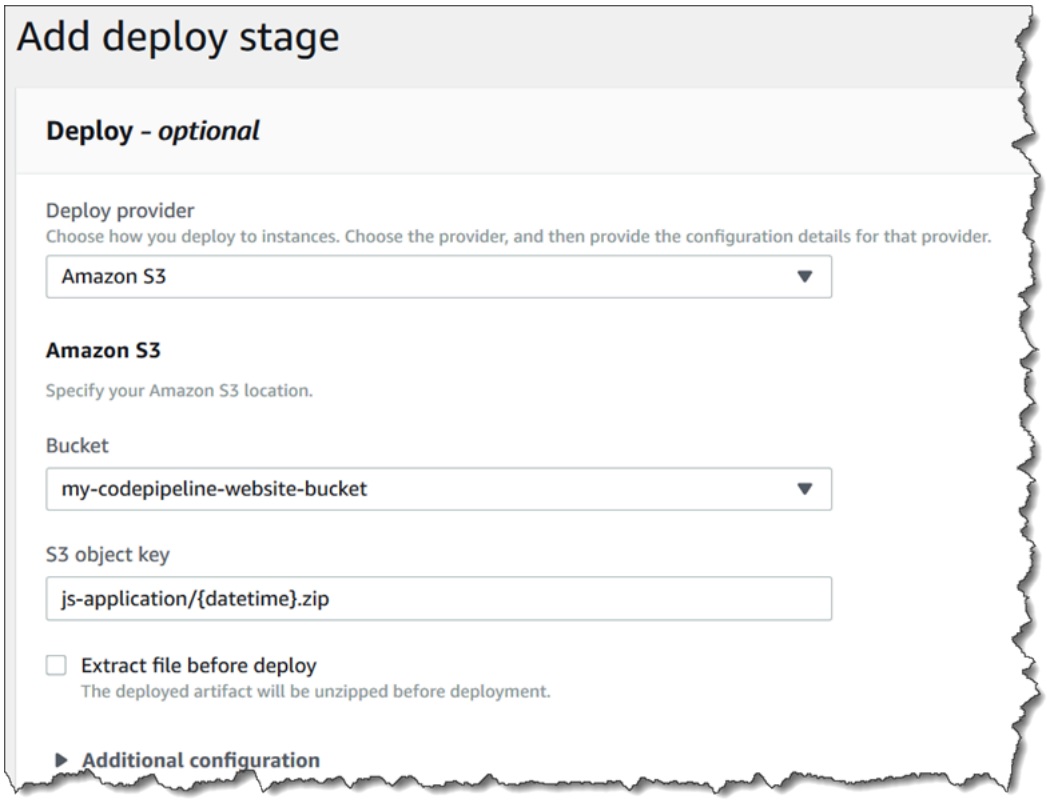
   [次へ] を選択します。
8. [ステップ 4: ビルドステージの追加] で、次の操作を行います。

   a. [ビルドプロバイダー] で、[CodeBuild] を選択します。
   b. Create build project (ビルドプロジェクトの作成)を選択します。[プロジェクトの作成] ページで:
   c. [プロジェクト名] に、このビルドプロジェクトの名前を入力します。
   d. [環境] で、[マネージド型イメージ] を選択します。[Operating system] で、[Ubuntu] を選択します。
   e. [ランタイム] で、[Standard (標準)] を選択します。[ランタイムバージョン] で、[aws/codebuild/standard:1.0] を選択します。
   f. [イメージのバージョン] で、[Always use the latest image for this runtime version (このランタイムバージョンには常に最新のイメージを使用)] を選択します。
   g. サービスのロール で、CodeBuild サービスロールを選択または作成します。
   h. [ビルド仕様] で、[Use a buildspec file (ビルド仕様ファイルの使用)] を選択します。
   i. [Continue to CodePipeline] (CodePipeline に進む) を選択します。プロジェクトが正常に作成された場合はメッセージが表示されます。
   j. [次へ] を選択します。
9. [ステップ 5: デプロイステージの追加] で、次の操作を行います。

   a. [デプロイプロバイダ] で、[Amazon S3] を選択します。
   b. [バケット] に、S3 ターゲットバケットの名前を入力します。
   c. [Extract file before deploy (デプロイ前にファイルを展開)] がオフになっていることを確認します。

      [Extract file before deploy (デプロイ前にファイルを展開)] がオフになっていると、[S3 object key (S3 オブジェクトキー)] が表示されています。使用するパスの名前を入力します: `js-application/{datetime}.zip`。

これにより、ファイルが展開される `js-application` フォルダが Amazon S3 に作成されます。このフォルダでは、パイプラインの実行時に `{datetime}` 変数によって各出力ファイルにタイムスタンプが付けられます。



d. (オプション) [既定 ACL] で、既定 ACL と呼ばれる、あらかじめ定義された一連の許可を、アップロードされたアーティファクトに適用できます。

e. (オプション) [キャッシュコントロール] で、キャッシュパラメータを入力します。これを設定して、リクエスト/レスポンスのキャッシュ動作を制御できます。有効な値については、HTTP オペレーションの `Cache-Control` ヘッダーフィールドを参照してください。

f. [次へ] を選択します。

10. [ステップ 6: 確認] で情報を確認し、[パイプラインの作成] を選択します。

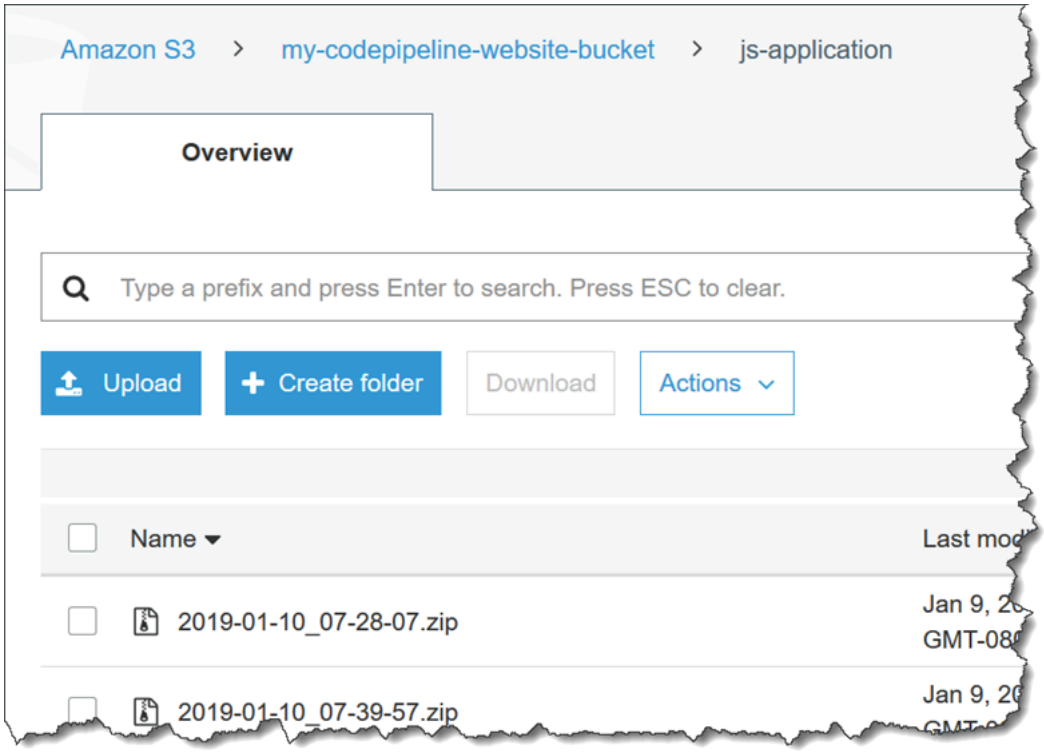
11. パイプラインが正常に実行されたら、Amazon S3 コンソールでバケットを表示します。デプロイした ZIP ファイルがターゲットバケットの `js-application` フォルダの下に表示されていることを確認します。ZIP ファイルに含まれる JavaScript ファイルは `index.js` です。`index.js` ファイルには、以下の出力が含まれています。

```
var HelloGreeting = /** @class */ (function () {
    function HelloGreeting() {
        this.message = "Hello!";
```

```
    }
    return HelloGreeting;
}());
function greet(greeting) {
    console.log(greeting.message);
}
var greeting = new HelloGreeting();
greet(greeting);
```

ステップ 3: 任意のソースファイルに変更を加えてデプロイを確認する

ソースファイルに変更を加え、それらのファイルをソースバケットにアップロードします。これにより、パイプラインの実行がトリガーされます。ターゲットのバケットを表示し、デプロイされた出力ファイルが以下のように `js-application` フォルダにあることを確認します。



チュートリアル: サーバーレスアプリケーションを に発行するパイプラインを作成する AWS Serverless Application Repository

を使用して AWS CodePipeline 、 AWS SAM サーバーレスアプリケーションを に継続的に配信できます AWS Serverless Application Repository。

> **Important**
>
> パイプライン作成の一環として、CodePipeline は、ユーザーが指定した S3 アーティファクトバケットをアーティファクトとして使用します (これは S3 ソースアクションで使用するバケットとは異なります)。S3 アーティファクトバケットがパイプラインのアカウントとは異なるアカウントにある場合は、S3 アーティファクトバケットが によって所有 AWS アカウント されており、安全で信頼できることを確認してください。

このチュートリアルでは、GitHub でホストされているサーバーレスアプリケーションを構築して AWS Serverless Application Repository に自動的に公開するようにパイプラインを作成して設定する方法を示します。このパイプラインでは、ソースプロバイダとして GitHub を使用し、ビルドプロバイダとして CodeBuild を使用します。サーバーレスアプリケーションを に発行するには AWS Serverless Application Repository、アプリケーションを (から AWS Serverless Application Repository) デプロイし、そのアプリケーションによって作成された Lambda 関数をパイプラインの呼び出しアクションプロバイダーとして関連付けます。その後、コードを記述することなく AWS Serverless Application Repository、アプリケーションの更新を に継続的に配信できます。

> **Important**
>
> この手順でパイプラインに追加するアクションの多くには、パイプラインを作成する前に作成する必要がある AWS リソースが含まれます。ソースアクションの AWS リソースは常に、パイプラインを作成するのと同じ AWS リージョンで作成する必要があります。例えば、米国東部 (オハイオ) リージョンにパイプラインを作成している場合、CodeCommit リポジトリは米国東部 (オハイオ) リージョンにある必要があります。
>
> パイプラインの作成時にクロスリージョンアクションを追加できます。クロスリージョンアクションの AWS リソースは、アクションを実行する予定のリージョンと同じ AWS リージョンに存在する必要があります。詳細については、「CodePipeline でクロスリージョンアクションを追加する」を参照してください。

[開始する前に]

このチュートリアルでは、以下のことを前提としています。

- AWS Serverless Application Model (AWS SAM) と AWS Serverless Application Repository をよく理解していること。

- CLI AWS Serverless Application Repository を使用して に発行したサーバーレスアプリケーションが GitHub AWS SAM でホストされている。サンプルアプリケーションを に公開するには AWS Serverless Application Repository、「 AWS Serverless Application Repository デベロッパーガイド」の「クイックスタート: アプリケーションの公開」を参照してください。独自のアプリケーションを に発行するには AWS Serverless Application Repository、「 AWS Serverless Application Model デベロッパーガイド」の「 CLI AWS SAM を使用したアプリケーションの発行」を参照してください。

ステップ 1: buildspec.yml ファイルを作成する

`buildspec.yml` ファイルを以下のとおりに作成し、これをサーバーレスアプリケーションの GitHub リポジトリに追加します。*`template.yml`* をアプリケーションの AWS SAM テンプレートに置き換え、 *`bucketname`* をパッケージ化されたアプリケーションが保存されている S3 バケットに置き換えます。

```
version: 0.2
phases:
  install:
    runtime-versions:
        python: 3.8
  build:
    commands:
      - sam package --template-file template.yml --s3-bucket bucketname --output-
template-file packaged-template.yml
artifacts:
  files:
    - packaged-template.yml
```

ステップ 2: パイプラインを作成して設定する

サーバーレスアプリケーションを公開 AWS リージョン する にパイプラインを作成するには、次の手順に従います。

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/codepipeline/ で CodePipeline コンソールを開きます。
2. 必要に応じて、サーバーレスアプリケーションを公開 AWS リージョン する に切り替えます。
3. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。

4. [ステップ 1: 作成オプションを選択する] ページの [作成オプション] で、[カスタムパイプラインを構築する] オプションを選択します。[次へ] を選択します。
5. [パイプラインの作成] を選択します。[ステップ 2: パイプラインの設定を選択する] ページで、[パイプライン名] にパイプラインの名前を入力します。
6. [パイプラインタイプ] で、[V2] を選択します。詳細については、「パイプラインのタイプ」を参照してください。[Next] (次へ) を選択します。
7. サービスロール で、新しいサービスロール を選択して、CodePipeline が IAM でサービスロールを作成できるようにします。
8. [詳細設定] をデフォルト設定のままにし、[次へ] を選択します。
9. [ステップ 3: ソースステージの追加] ページの [ソースプロバイダー] で、[GitHub] を選択します。
10. 接続 で、既存の接続を選択するか、新規の接続を作成します。GitHub ソースアクション用の接続を作成または管理するには、GitHub コネクション を参照してください。
11. [リポジトリ] で、GitHub ソースリポジトリを選択します。
12. [ブランチ] で、GitHub ブランチを選択します。
13. ソースアクションの残りはデフォルトのままにしておきます。[次へ] を選択します。
14. [ステップ 4: ビルドステージの追加] ページで、ビルドステージを追加します。

    a. [ビルドプロバイダ] で、[AWS CodeBuild] を選択します。[リージョン] で、パイプラインリージョンを使用します。

    b. [プロジェクトを作成] を選択します。

    c. [プロジェクト名] に、このビルドプロジェクトの名前を入力します。

    d. [環境イメージ] で、[Managed image (マネージド型イメージ)] を選択します。[Operating system] で、[Ubuntu] を選択します。

    e. [ランタイム] と [ランタイムバージョン] で、サーバーレスアプリケーションに必要なランタイムとバージョンを選択します。

    f. [サービスロール] で、[New service role (新しいサービスロール)] を選択します。

    g. [ビルド仕様] で、[Use a buildspec file (ビルド仕様ファイルの使用)] を選択します。

    h. [Continue to CodePipeline] (CodePipeline に進む) を選択します。これにより CodePipeline コンソールが開き、リポジトリ内の `buildspec.yml` の作成に使用する CodeBuild プロジェクトが作成されます。ビルドプロジェクトでは、サービスロールを使用して AWS のサービス のアクセス許可を管理します。このステップには数分かかる場合があります。

    i. [次へ] を選択します。

15. [ステップ 5: テストステージの追加] で、[テストステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

    [次へ] を選択します。

16. [ステップ 6: デプロイステージの追加] ページで、[デプロイステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。[次へ] を選択します。

17. ステップ 7: 確認で、パイプラインの作成を選択します。ステージを示す図が表示されます。

18. パッケージ化されたアプリケーションの保存先の S3 バケットにアクセスする許可を CodeBuild サービスロールに付与します。

    a. 新しいパイプラインの [ビルド] ステージで、[CodeBuild] を選択します。

    b. [Build details (ビルドの詳細)] タブを選択します。

    c. [環境] で、CodeBuild サービスロールを選択して IAM コンソールを開きます。

    d. `CodeBuildBasePolicy` の選択を展開し、[Edit policy (ポリシーの編集)] を選択します。

    e. JSON を選択します。

    f. 新しいポリシーステートメントを以下の内容で追加します。このステートメントにより、CodeBuild はパッケージ化されたアプリケーションの保存先である S3 バケットにオブジェクトを配置できます。*bucketname* は、実際の S3 バケット名に置き換えます。

    ```
        {
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::bucketname/*"
            ],
            "Action": [
                "s3:PutObject"
            ]
        }
    ```

    g. [ポリシーの確認] を選択します。

    h. [Save changes] (変更の保存) をクリックします。

ステップ 3: 発行アプリケーションをデプロイする

以下のステップに従って、 AWS Serverless Application Repositoryの公開を実行する Lambda 関数を含むアプリケーションをデプロイします。このアプリケーションは aws-serverless-codepipeline-serverlessrepo-publish です。

> **Note**
>
> パイプライン AWS リージョン と同じ にアプリケーションをデプロイする必要があります。

1. アプリケーションのページに移動し、[デプロイ] を選択します。
2. [I acknowledge that this app creates custom IAM roles (このアプリケーションがカスタム IAM ロールを作成することを承認します)] を選択します。
3. [デプロイ] をクリックします。
4. CloudFormation スタックの表示を選択して CloudFormation コンソールを開きます。
5. [リソース] セクションを展開します。ServerlessRepoPublish (AWS::Lambda::Function 型) が表示されます。このリソースの物理 ID を書き留めます。次のステップで使用します。この物理 ID は、CodePipeline で新しい公開アクションを作成するときに使用します。

ステップ 4: 発行アクションを作成する

以下のステップに従って、パイプラインの発行アクションを作成します。

1. CodePipeline コンソール (https://console.aws.amazon.com/codepipeline/) を開きます。
2. 左のナビゲーションセクションで、編集するパイプラインを選択します。
3. [編集] を選択します。
4. 現在のパイプラインの最後のステージが終わったら、[+ Add stage (+ ステージの追加)] を選択します。[Stage name (ステージ名)] に名前 (**`Publish`** など) を入力し、[Add stage (ステージの追加)] を選択します。
5. 新しいステージで、[+ Add action group (+ アクショングループの追加)] を選択します。
6. アクション名を入力します。[アクションプロバイダ] の [呼び出し] で、[AWS Lambda] を選択します。
7. [入力アーティファクト] で、[BuildArtifact] を選択します。
8. [関数名] で、前のステップで書き留めた Lambda 関数の物理 ID を選択します。
9. アクションの [保存] を選択します。
10. ステージの [完了] を選択します。
11. 右上の [保存] を選択します。
12. パイプラインを検証するには、GitHub でアプリケーションに変更を加えます。たとえば、 AWS SAM テンプレートファイルの `Metadata`セクションでアプリケーションの説明を変更します。

変更をコミットして GitHub ブランチにプッシュします。これにより、パイプラインの実行がトリガーされます。パイプラインが完了したら、アプリケーションが更新されて変更が反映されていることを AWS Serverless Application Repository で確認します。

チュートリアル: CodePipeline を使用した Lambda 関数のデプロイ

このチュートリアルは、Lambda で設定した関数にコードをデプロイするデプロイアクションを CodePipeline で作成する方法について説明します。ここでは、サンプル Lambda 関数を作成します。この関数では、エイリアスとバージョンを作成し、圧縮された Lambda 関数をソースの場所に追加して、パイプラインで Lambda アクションを実行します。

> **Note**
>
> コンソールでのパイプライン作成の一環として、CodePipeline は S3 アーティファクトバケットをアーティファクトとして使用します (これは S3 ソースアクションで使用するバケットとは異なります)。S3 アーティファクトバケットがパイプラインのアカウントとは異なるアカウントにある場合は、S3 アーティファクトバケットが によって所有 AWS アカウントされており、安全で信頼できることを確認してください。

> **Note**
>
> Lambda デプロイアクションは V2 タイプのパイプラインでのみ使用できます。

前提条件

このチュートリアルで CD パイプラインを作成する前に、いつくかのリソースを用意する必要があります。使用を開始するために必要なものは以下のとおりです。

> **Note**
>
> これらのリソースはすべて、同じ AWS リージョン内に作成する必要があります。

- GitHub などのソースコントロールリポジトリ、または Lambda 関数用に作成した `.zip` ファイルを保存するソース S3 バケット (このチュートリアルでは S3 を使用します)。

- このアクションのアクセス許可で更新された既存の CodePipeline サービスロールを使用する必要があります。サービスロールを更新するには、「Lambda デプロイアクションのサービスロールポリシーのアクセス許可」を参照してください。

これらの前提条件を満たした後、チュートリアルに進んで CD パイプラインを作成できます。

ステップ 1: サンプル Lambda 関数を作成する

このステップでは、デプロイする Lambda 関数を作成します。

Lambda 関数を作成するには

1. Lambda コンソールにアクセスし、次のチュートリアルの手順に従ってサンプル Lambda 関数を作成します。
2. 上部のナビゲーションで [作成]を選択し、ページの上部で [ゼロから開始] を選択します。
3. [名前] に「**MyLambdaFunction**」と入力します。
4. 新しいバージョンを発行します。これは、エイリアスが指すボットの更新されたバージョンになります。
   a. 関数を選択します。
   b. [アクション] ドロップダウンメニューを選択します。
   c. [新しいバージョンを発行] を選択します。
   d. (オプション) [説明] に説明を追加します。
   e. [公開] を選択します。
5. `aliasV1` などの関数のエイリアスを作成します。
6. エイリアスが、先ほど作成したバージョン (1 など) を指していることを確認します。

> **Note**
>
> [$LATEST] を選択した場合、Lambda は複数のバージョンを指すエイリアスに対して $LATEST をサポートしていないため、トラフィックシフト機能は使用できません。

ステップ 2: 関数ファイルをリポジトリにアップロードする

関数をダウンロードして、zip ファイルとして保存します。次の手順を使用して、圧縮ファイルを S3 バケットにアップロードします。

ソースリポジトリに **.zip** ファイルを追加するには

1. S3 バケットを開きます。
2. アップロード を選択します。
3. `sample_lambda_source.zip` ファイルを含む zip ファイルをソースバケットにアップロードします。

   パスをメモします。

   ```
   object key
   ```

ステップ 3: パイプラインを作成する

CodePipeline ウィザードを使用してパイプラインステージを作成し、ソースリポジトリを接続します。

パイプラインを作成するには

1. CodePipeline コンソール (https://console.aws.amazon.com/codepipeline/) を開きます。
2. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。
3. [ステップ 1: 作成オプションを選択する] ページの [作成オプション] で、[カスタムパイプラインを構築する] オプションを選択します。[次へ] を選択します。
4. [ステップ 2: パイプラインの設定を選択する] で、[パイプライン名] に「**MyPipeline**」と入力します。
5. CodePipeline は、特徴と料金が異なる V1 タイプと V2 タイプのパイプラインを提供しています。V2 タイプは、コンソールで選択できる唯一のタイプです。詳細については、「パイプラインタイプ」を参照してください。CodePipeline の料金については、料金を参照してください。
6. [サービスロール] で、[既存サービスロールの使用] を選択し、このアクションに必要なアクセス許可で更新された CodePipeline サービスロールを選択します。このアクション用に

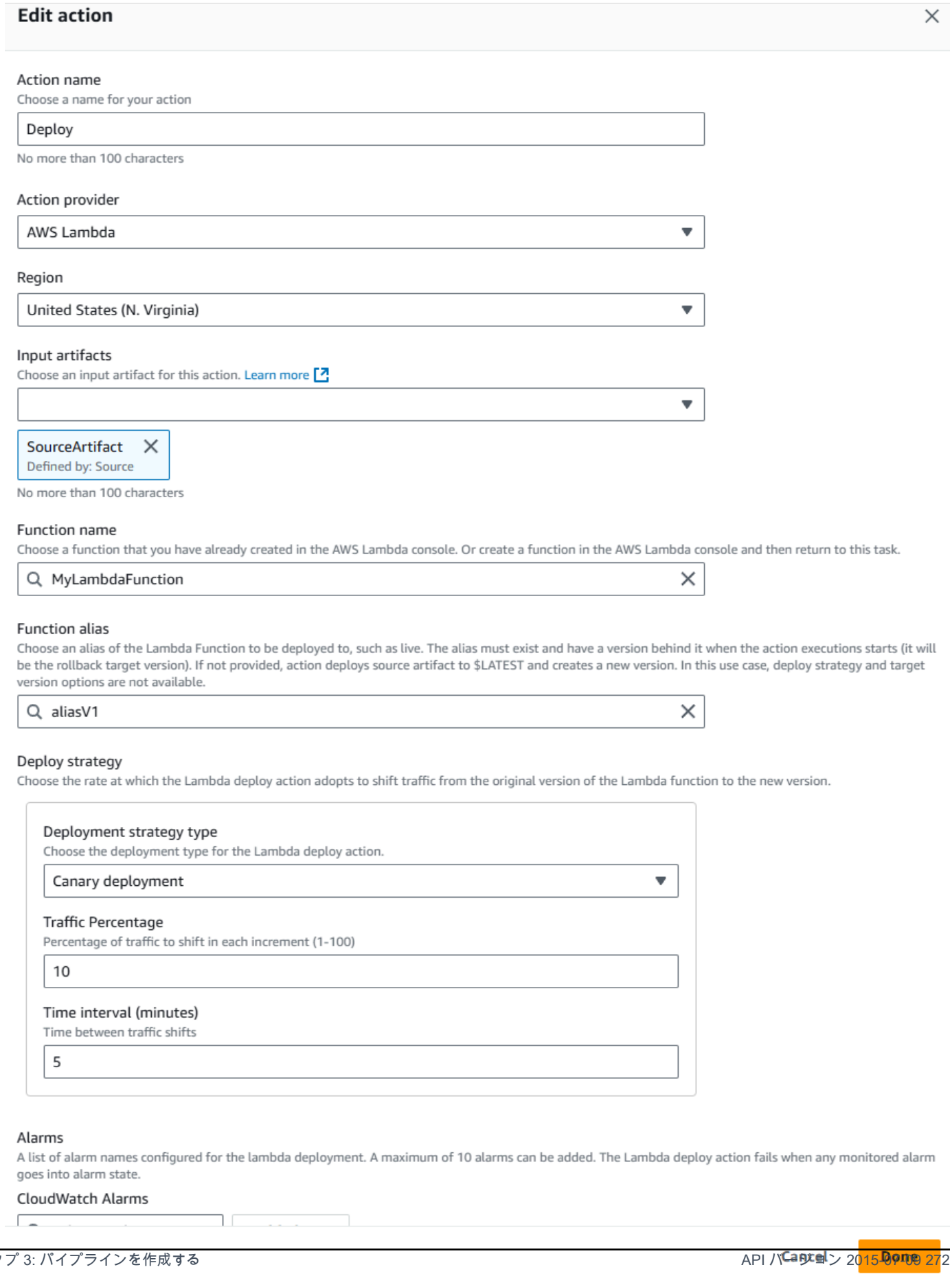
CodePipeline サービスロールを設定するには、「Lambda デプロイアクションのサービスロールポリシーのアクセス許可」を参照してください。

7. [詳細設定] をデフォルト設定のままにし、[次へ] を選択します。
8. [ステップ 3: ソースステージの追加] ページで、ソースステージを追加します。

   a. [ソースプロバイダ] で [Amazon S3] を選択します。

   b. [オブジェクトキー] で、`sample_lambda_source.zip` などのファイル拡張子を含む .zip ファイルの名前を追加します。

   [次へ] を選択します。
9. [ステップ 4: ビルドステージの追加] ページで、[スキップ] を選択します。
10. [ステップ 5: テストステージの追加] ページで、[スキップ] を選択します。
11. [ステップ 6: デプロイステージの追加] ページで、[Lambda] を選択します。

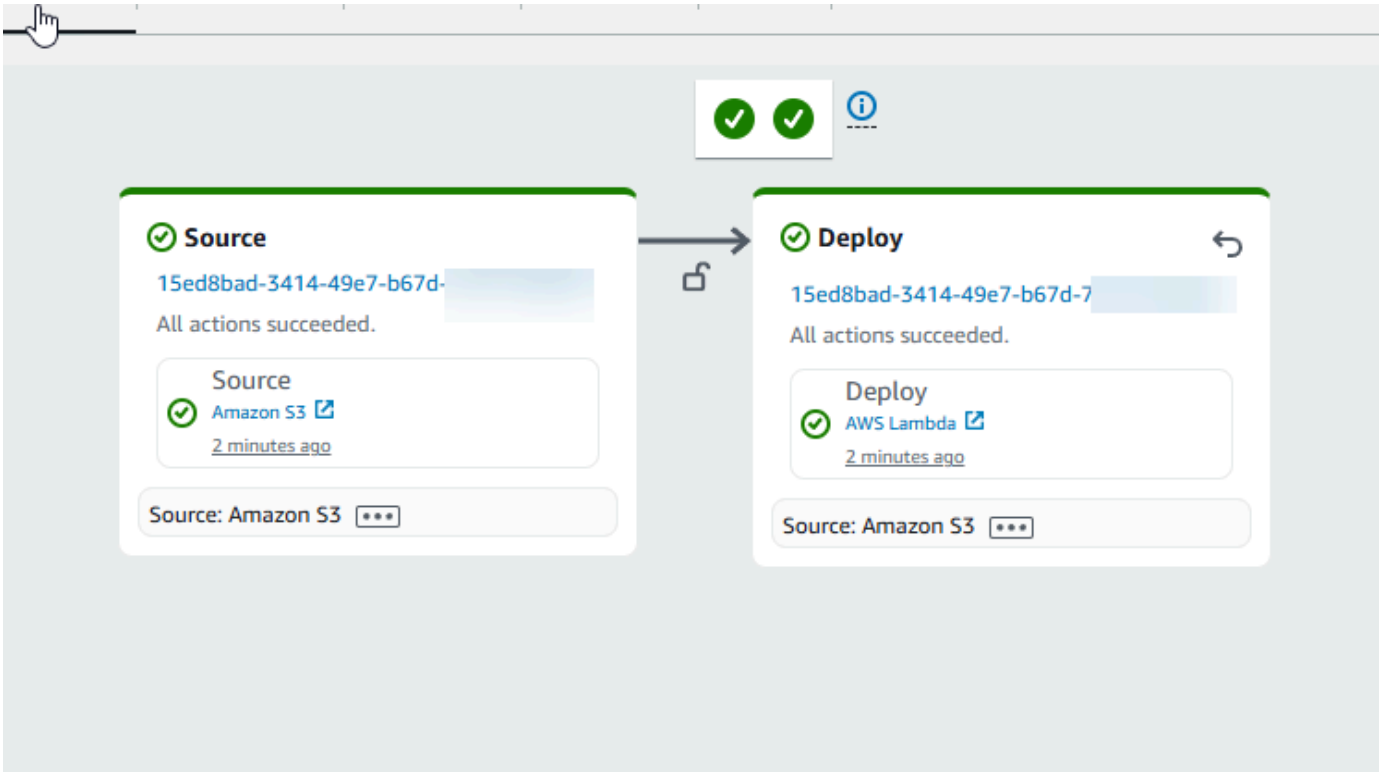

Edit action
Action name
Choose a name for your action
Deploy
No more than 100 characters
Action provider
AWS Lambda
Region
United States (N. Virginia)
Input artifacts
Choose an input artifact for this action. Learn more
SourceArtifact
Defined by: Source
No more than 100 characters
Function name
Choose a function that you have already created in the AWS Lambda console. Or create a function in the AWS Lambda console and then return to this task.
MyLambdaFunction
Function alias
Choose an alias of the Lambda Function to be deployed to, such as live. The alias must exist and have a version behind it when the action executions starts (it will be the rollback target version). If not provided, action deploys source artifact to $LATEST and creates a new version. In this use case, deploy strategy and target version options are not available.
aliasV1
Deploy strategy
Choose the rate at which the Lambda deploy action adopts to shift traffic from the original version of the Lambda function to the new version.
Deployment strategy type
Choose the deployment type for the Lambda deploy action.
Canary deployment
Traffic Percentage
Percentage of traffic to shift in each increment (1-100)
10
Time interval (minutes)
Time between traffic shifts
5
Alarms
A list of alarm names configured for the lambda deployment. A maximum of 10 alarms can be added. The Lambda deploy action fails when any monitored alarm goes into alarm state.
CloudWatch Alarms
Cancel
Done

a. 関数名とエイリアスを追加します。

b. デプロイ戦略を選択します。

c. [次へ] を選択します。

12. ステップ 7: 確認ページで、パイプライン設定を確認し、パイプラインの作成を選択してパイプラインを作成します。



ステップ 4: パイプラインのテスト

パイプラインには、end-to-endのネイティブ AWS 継続的デプロイを実行するためのすべてが必要です。次は、コードの変更をソースリポジトリにプッシュすることで機能をテストします。

パイプラインをテストするには

1. 設定済みソースリポジトリにコード変更を行い、変更をコミットしてプッシュします。
2. CodePipeline コンソール (https://console.aws.amazon.com/codepipeline/) を開きます。
3. リストからパイプラインを選択します。
4. ステージを通してパイプラインの進行状況を監視します。これでパイプラインが完了し、アクションが Lambda 関数にデプロイされます。

詳細情報

Lambda デプロイアクションでは、2 つのデプロイ方法を使用できます。1 つ目の方法は、ソースアクションからの入力アーティファクトなしでトラフィックを単独で移行することです。もう 1 つの方法は、ソースアクションの入力アーティファクトを使用して関数コードを更新し、更新されたコードに基づいて新しいバージョンを発行することです。2 つ目の方法では、エイリアスが指定されている場合、CodePipeline はトラフィックの移行も行います。この Lambda デプロイアクションのチュートリアルでは、ソースアーティファクトを使用して関数を更新する方法を示します。

アクションの詳細については、「AWS Lambda デプロイアクションリファレンス」のアクションリファレンスページを参照してください。

チュートリアル: Lambda 呼び出しアクションで変数を使用する

Lambda 呼び出しアクションは、入力のパートとして別のアクションの変数を使用し、出力とともに新しい変数を返すことができます。CodePipeline のアクションの変数については、「変数リファレンス」を参照してください。

> **Important**
>
> パイプライン作成の一環として、CodePipeline は、ユーザーが指定した S3 アーティファクトバケットをアーティファクトとして使用します (これは S3 ソースアクションで使用するバケットとは異なります)。S3 アーティファクトバケットがパイプラインのアカウントとは異なるアカウントにある場合は、S3 アーティファクトバケットが によって所有 AWS アカウント されており、安全で信頼できることを確認してください。

このチュートリアルを終了すると、以下の項目が使用可能になります。

- Lambda は次のアクションを呼び出します。
  - CodeCommit ソースアクションからの `CommitId` 変数を使用する
  - 3 つの新しい変数として `dateTime`、`testRunId`、`region` を出力する
- Lambda 呼び出しアクションからの新しい変数を使用してテスト URL とテスト実行 ID を提供する手動承認アクション
- 新しいアクションを反映して更新されたパイプライン

トピック

- 前提条件
- ステップ 1: Lambda 関数を作成する
- ステップ 2: Lambda 呼び出しアクションと手動承認アクションをパイプラインに追加する

前提条件

始めるには以下のものが必要です。

- チュートリアル: シンプルなパイプラインを作成する (CodeCommit リポジトリ) の CodeCommit ソースを使用してパイプラインを作成または使用できます。
- 既存のパイプラインを編集して CodeCommit ソースアクションに名前空間を含めます。名前空間 `SourceVariables` をアクションに割り当てます。

ステップ 1: Lambda 関数を作成する

次のステップを使用して Lambda 関数と Lambda 実行ロールを作成します。Lambda 関数を作成した後、パイプラインに Lambda アクションを追加します。

Lambda 関数と実行ロールを作成するには

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/lambda/ で AWS Lambda コンソールを開きます。
2. [関数の作成] を選択してください。[一から作成] が選択された状態のままにしておきます。
3. [関数名] に、関数の名前 (**`myInvokeFunction`** など) を入力します。[ランタイム] は、デフォルトのオプションを選択したままにします。
4. [実行ロールの選択または作成] を選択します。[基本的な Lambda アクセス権限で新しいロールを作成] を選択します。
5. [関数の作成] を選択してください。
6. 別のアクションからの変数を使用するには、Lambda 呼び出しアクションの設定で `UserParameters` にその変数を渡す必要があります。アクションはチュートリアルの後半でパイプラインで設定しますが、変数が渡されると仮定してコードを追加します。

   新しい変数を生成するには、入力の `outputVariables` というプロパティを `putJobSuccessResult` に設定します。`putJobFailureResult` の一部として変数を生成することはできない点に注意してください。

```
 const putJobSuccess = async (message) => {
        const params = {
            jobId: jobId,
            outputVariables: {
                testRunId: Math.floor(Math.random() * 1000).toString(),
                dateTime: Date(Date.now()).toString(),
                region: lambdaRegion
            }
        };
```

新しい関数の [コード] タブで、次のサンプルコードを index.mjs の下に貼り付けます。

```
import { CodePipeline } from '@aws-sdk/client-codepipeline';

export const handler = async (event, context) => {
    const codepipeline = new CodePipeline({});

    // Retrieve the Job ID from the Lambda action
    const jobId = event["CodePipeline.job"].id;

    // Retrieve UserParameters
    const params =
 event["CodePipeline.job"].data.actionConfiguration.configuration.UserParameters;

    // The region from where the lambda function is being executed
    const lambdaRegion = process.env.AWS_REGION;

    // Notify CodePipeline of a successful job
    const putJobSuccess = async (message) => {
        const params = {
            jobId: jobId,
            outputVariables: {
                testRunId: Math.floor(Math.random() * 1000).toString(),
                dateTime: Date(Date.now()).toString(),
                region: lambdaRegion
            }
        };

        try {
            await codepipeline.putJobSuccessResult(params);
            return message;
        } catch (err) {
```

```
            throw err;
        }
    };

    // Notify CodePipeline of a failed job
    const putJobFailure = async (message) => {
        const params = {
            jobId: jobId,
            failureDetails: {
                message: JSON.stringify(message),
                type: 'JobFailed',
                externalExecutionId: context.invokeid
            }
        };

        try {
            await codepipeline.putJobFailureResult(params);
            throw message;
        } catch (err) {
            throw err;
        }
    };

    try {
        console.log("Testing commit - " + params);

        // Your tests here

        // Succeed the job
        return await putJobSuccess("Tests passed.");
    } catch (ex) {
        // If any of the assertions failed then fail the job
        return await putJobFailure(ex);
    }
};
```

7. 自動保存することを関数に許可します。
8. 画面上部の [関数 ARN] にある Amazon リソースネーム (ARN) をコピーします。
9. 最後のステップとして、https://console.aws.amazon.com/iam/ で AWS Identity and Access Management (IAM) コンソールを開きます。Lambda 実行ロールを変更して、次のポリシーを追加します。AWSCodePipelineCustomActionAccess。Lambda 実行ロールを作成したり、ロール

ポリシーを変更したりする手順については、「ステップ 2 : Lambda 関数を作成する」を参照してください。

ステップ 2: Lambda 呼び出しアクションと手動承認アクションをパイプラインに追加する

このステップでは、パイプラインに Lambda 呼び出しアクションを追加します。Test という名前のステージの一部としてアクションを追加します。アクションタイプは、呼び出しアクションです。次に、呼び出しアクションの後に、手動承認アクションを追加します。

パイプラインに Lambda アクションと手動承認アクションを追加するには

1. CodePipeline コンソール (https://console.aws.amazon.com/codepipeline/) を開きます。

   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。アクションを追加するパイプラインを選択します。
2. パイプラインに Lambda テストアクションを追加します。

   a. パイプラインを編集するには、[編集] を選択します。既存のパイプラインで、ソースアクションの後にステージを追加します。ステージの名前 (**Test** など) を入力します。

   b. 新しいステージで、[アクショングループを追加する] を選択してアクションを追加します。「アクション名」に、呼び出しアクションの名前 (**Test_Commit** など) を入力します。

   c. [アクションプロバイダー] で、[AWS Lambda] を選択します。

   d. [入力アーティファクト] で、ソースアクションの出力アーティファクトの名前 (`SourceArtifact` など) を選択します。

   e. [関数名] で、作成した Lambda 関数の ARN を追加します。

   f. [変数名前空間] に名前空間名 (**TestVariables** など) を追加します。

   g. [出力アーティファクト] で、出力アーティファクト名 (**LambdaArtifact** など) を追加します。

   h. [Done] (完了) をクリックします。
3. パイプラインに手動の承認アクションを追加します。

   a. パイプラインが編集モードのままで、呼び出しアクションの後にステージを追加します。ステージの名前 (**Approval** など) を入力します。

   b. 新しいステージで、アクションを追加するアイコンを選択します。[アクション名] に、承認アクションの名前 (**Change_Approval** など) を入力します。

c. [アクションプロバイダ] で、[手動承認] を選択します。

d. [レビューする URL] で、region 変数と CommitId 変数の変数構文を追加して URL を作成します。出力変数を提供するアクションに割り当てられた名前空間を使用してください。

この例では、CodeCommit アクションの変数構文を持つ URL にはデフォルトの名前空間 SourceVariables があります。Lambda リージョン出力変数には、TestVariables 名前空間があります。URL は次のようになります。

```
https://#{TestVariables.region}.console.aws.amazon.com/codesuite/codecommit/
repositories/MyDemoRepo/commit/#{SourceVariables.CommitId}
```

[コメント] で、testRunId 変数の変数構文を追加して、承認メッセージテキストを作成します。この例では、Lambda testRunId 出力変数の変数構文を持つ URL には TestVariables 名前空間があります。以下のメッセージを入力します。

```
Make sure to review the code before approving this action. Test Run ID:
 #{TestVariables.testRunId}
```

4. [完了] を選択してアクションの編集画面を閉じ、[完了] を選択してステージの編集画面を閉じます。パイプラインを保存するには、[完了] を選択します。完成したパイプラインには、ソース、テスト、承認、デプロイの各ステージがある構造が含まれています。

[変更のリリース] を選択して、パイプライン構造で最新の変更を実行します。

5. パイプラインが手動承認ステージに達したら、[確認] を選択します。解決された変数は、コミット ID の URL として表示されます。承認者は、コミットを表示する URL を選択できます。

6. パイプラインが正常に実行されたら、アクションの実行履歴ページで変数の値を表示することもできます。

チュートリアル: パイプラインで AWS Step Functions 呼び出しアクションを使用する

AWS Step Functions を使用して、ステートマシンを作成および設定できます。このチュートリアルでは、パイプラインからステートマシンの実行を有効化するパイプラインに呼び出しアクションを追加する方法を説明します。

> **Important**
>
> パイプライン作成の一環として、CodePipeline は、ユーザーが指定した S3 アーティファクトバケットをアーティファクトとして使用します (これは S3 ソースアクションで使用するバケットとは異なります)。S3 アーティファクトバケットがパイプラインのアカウントとは異なるアカウントにある場合は、S3 アーティファクトバケットが によって所有 AWS アカウント されており、安全で信頼できることを確認してください。

このチュートリアルでは、以下のタスクを行います。

- で標準ステートマシンを作成します AWS Step Functions。
- ステートマシンの入力 JSON ディレクトリを直接入力します。ステートマシンの入力ファイルを Amazon Simple Storage Service (Amazon S3) バケットにアップロードすることもできます。
- ステートマシンのアクションを追加して、パイプラインを更新します。

トピック

- 前提条件: シンプルなパイプラインを作成または選択する
- ステップ 1: サンプルステートマシンを作成する
- ステップ 2: パイプラインにステップ関数呼び出しアクションを追加する

前提条件: シンプルなパイプラインを作成または選択する

このチュートリアルでは、既存のパイプラインに呼び出しアクションを追加します。チュートリアル: シンプルなパイプラインを作成する (S3 バケット) または チュートリアル: シンプルなパイプラインを作成する (CodeCommit リポジトリ) で作成したパイプラインを使用できます。

ソースアクションと少なくとも 2 ステージ構造を持つ既存のパイプラインを使用しますが、この例ではソースアーティファクトを使用しません。

> **Note**
>
> このアクションを実行するために必要な追加のアクセス許可を使用して、パイプラインで使用されるサービスロールを更新する必要がある場合があります。これを行うには、 AWS Identity and Access Management (IAM) コンソールを開き、ロールを見つけてから、ロール

のポリシーにアクセス許可を追加します。詳細については、「CodePipeline サービスロールにアクセス許可を追加する」を参照してください。

ステップ 1: サンプルステートマシンを作成する

ステップ関数コンソールで、`HelloWorld` サンプルテンプレートを使用してステートマシンを作成します。手順については、AWS Step Functions デベロッパーガイド の ステートマシンの作成 を参照してください。

ステップ 2: パイプラインにステップ関数呼び出しアクションを追加する

次のように、ステップ関数呼び出しアクションをパイプラインに追加します。

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

   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。
2. [名前] で、編集するパイプラインの名前を選択します。これにより、パイプラインの詳細ビューが開いて、パイプラインの各ステージの各アクションの状態などがわかります。
3. パイプライン詳細ページで、[編集] を選択します。
4. シンプルなパイプラインの 2 番目のステージで、[Edit stage (ステージの編集)] を選択します。[削除] を選択します。これで、不要になった 2 番目のステージが削除されました。
5. 図の最下部で [+ Add stage] (+ ステージの追加) を選択します。
6. [ステージ名] にステージ名 (**`Invoke`** など) を入力し、[Add stage (ステージの追加)] を選択します。
7. [+ Add action group (+ アクションの追加)] を選択します。
8. [アクション名] に名前 (**`Invoke`** など) を入力します。
9. [アクションプロバイダー] で、[AWS ステップ関数] を選択します。[リージョン] をデフォルトでパイプラインのリージョンにすることを許可します。
10. [入力アーティファクト] で [`SourceArtifact`] を選択します。
11. [State machine ARN (ステートマシン ARN)] で、前に作成したステートマシンの Amazon リソースネーム (ARN) を選択します。
12. (オプション) [Execution name prefix (実行名のプレフィックス)] に、ステートマシンの実行 ID に追加するプレフィックスを入力します。
13. [Input type (入力タイプ)] で [Literal (リテラル)] を選択します。

14. [Input (入力)] に、HelloWorld サンプルステートマシンが想定する入力 JSON を入力します。

> **Note**
>
> ステートマシンの実行への入力は、CodePipeline でアクションの入力アーティファクトを記述するために使用される条件とは異なります。

この例では、次の JSON を入力します。

```
{"IsHelloWorldExample": true}
```

15. [Done] (完了) をクリックします。
16. 編集中のステージで、[完了] を選択します。 AWS CodePipeline のペインで [保存] を選択し、警告メッセージで [保存] を選択します。
17. 変更を送信してパイプラインの実行を開始するには、[変更のリリース]、[リリース] の順に選択します。
18. 完了したパイプライン内の呼び出しアクションで、[AWS ステップ関数] を選択します。 AWS Step Functions コンソールで、ステートマシンの実行 ID を表示します。ID には、ステートマシン名 HelloWorld と、ステートマシンの実行 ID とプレフィックス my-prefix が表示されます。

```
arn:aws:states:us-west-2:account-ID:execution:HelloWorld:my-
prefix-0d9a0900-3609-4ebc-925e-83d9618fcca1
```

チュートリアル: AWS AppConfig をデプロイプロバイダーとして使用するパイプラインを作成する

このチュートリアルでは、デプロイステージのデプロイアクションプロバイダーとして AWS AppConfig を使用して設定ファイルを継続的に配信するパイプラインを設定します。

> **Important**
>
> パイプライン作成の一環として、CodePipeline は、ユーザーが指定した S3 アーティファクトバケットをアーティファクトとして使用します (これは S3 ソースアクションで使用するバケットとは異なります)。S3 アーティファクトバケットがパイプラインのアカウントとは異

なるアカウントにある場合は、S3 アーティファクトバケットが によって所有 AWS アカウント されており、安全で信頼できることを確認してください。

トピック

- 前提条件
- ステップ 1: AWS AppConfig リソースを作成する
- ステップ 2 : ファイルを S3 ソースバケットにアップロードします。
- ステップ 3: パイプラインを作成する
- ステップ 4 : 任意のソースファイルに変更を加えてデプロイを確認します。

前提条件

開始する前に、次を完了しておく必要があります。

- この例では、パイプラインに S3 ソースを使用します。バージョニングが有効になっている Amazon S3 バケットを作成するか、使用します。「ステップ 1: アプリケーションの S3 バケットを作成する」の手順に従って、S3 バケットを作成します。

ステップ 1: AWS AppConfig リソースを作成する

このセクションでは、次のリソースを作成します。

- AWS AppConfig のアプリケーションは、顧客に機能を提供するコードの論理単位です。
- AWS AppConfig の環境は、ベータ環境や本番環境のアプリケーションなど、AppConfig ターゲットの論理デプロイグループです。
- 設定プロファイル は、アプリケーションの動作に影響する設定のコレクションです。設定プロファイルにより、 AWS AppConfig は保存された場所にある設定にアクセスできます。
- （ オプション ）AWS AppConfig の デプロイメント戦略 は、デプロイメント中の任意の時点で、クライアントの何パーセントが新しいデプロイされた設定を受け取る必要があるかなど、設定デプロイメントの動作を定義します。

アプリケーション、環境、設定プロファイル、デプロイ戦略を作成します。

1. AWS マネジメントコンソールにサインインします。

2. AWS AppConfig のリソースを作成するには、次のトピックのステップを使用します

   - アプリケーションを作成します。
   - 環境を作成します。
   - AWS CodePipeline 設定プロファイルを作成します。
   - (オプション) 定義済みのデプロイ戦略を選択するか、独自のデプロイメント戦略を作成します。

ステップ 2：ファイルを S3 ソースバケットにアップロードします。

このセクションでは、設定ファイルを作成します。次に、パイプラインがソースステージに使用するバケットにソースファイルを zip してプッシュします。

設定ファイルを作成します。

1. 各リージョンの設定ごとに configuration.json ファイルを作成します。次の内容を含めます。:

```
Hello World!
```

2. 次のステップを使用して、設定ファイルを zip してアップロードします。

ソースファイルを zip してアップロードします。

1. ファイルで .zip ファイルを作成し、.zip ファイル configuration-files.zip に名前を付けます。たとえば、.zip ファイルは次の構造を使用できます。:

```
.
### appconfig-configurations
    ### MyConfigurations
        ### us-east-1
        #   ### configuration.json
        ### us-west-2
            ### configuration.json
```

2. バケットの Amazon S3 コンソールで、アップロード を選択し、指示に従って、zip ファイルをアップロードします。

ステップ 3: パイプラインを作成する

このセクションでは、以下のアクションを使用してパイプラインを作成します。

- ソースアーティファクトが設定のファイルである Amazon S3 アクションを含むソースステージ。
- AppConfig デプロイメントアクションを使用したデプロイメントステージ。

ウィザードを使用してパイプラインを作成するには

1. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。
3. [ステップ 1: 作成オプションを選択する] ページの [作成オプション] で、[カスタムパイプラインを構築する] オプションを選択します。[次へ] を選択します。
4. [ステップ 2: パイプラインの設定を選択する] で、[パイプライン名] に「**MyAppConfigPipeline**」と入力します。
5. CodePipeline は、特徴と料金が異なる V1 タイプと V2 タイプのパイプラインを提供しています。V2 タイプは、コンソールで選択できる唯一のタイプです。詳細については、「パイプラインタイプ」を参照してください。CodePipeline の料金については、料金を参照してください。
6. サービスロール で、新しいサービスロール を選択して、CodePipeline が IAM でサービスロールを作成できるようにします。
7. [詳細設定] をデフォルト設定のままにし、[次へ] を選択します。
8. [ステップ 3: ソースステージの追加] の [ソースプロバイダー] で、[Amazon S3] を選択します。バケット で、S3 ソースバケットの名前を選択します。

   S3 オブジェクトキー に、ZIP ファイル名に `configuration-files.zip` を入力します。

   [次へ] を選択します。
9. [ステップ 4: ビルドステージの追加] で、[ビルドステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

   [次へ] を選択します。
10. [ステップ 5: テストステージの追加] で、[テストステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

[次へ] を選択します。

11. ステップ 6: デプロイステージの追加:

    a. [デプロイプロバイダー] で、[AWS AppConfig] を選択します。

    b. Application で、 AWS AppConfig で作成したアプリケーションの名前を選択します。フィールドにはアプリケーションの ID が表示されます。

    c. Environment で、 AWS AppConfig で作成した環境の名前を選択します。フィールドには環境の ID が表示されます。

    d. 設定プロファイルで、 AWS AppConfig で作成した設定プロファイルの名前を選択します。このフィールドには、設定プロファイルの ID が表示されます。

    e. デプロイ戦略 で、デプロイ戦略の名前を選択します。これは、AppConfig で作成したデプロイ戦略、または AppConfig の事前定義されたデプロイ戦略から選択したデプロイ戦略のいずれかです。このフィールドには、デプロイ戦略の ID が表示されます。

    f. アーティファクト設定パスを入力 に、ファイルパスを入力します。入力アーティファクト設定パスが S3 バケット.zip ファイルのディレクトリ構造と一致していることを確認します。この例では、次のファイルパス : `appconfig-configurations/MyConfigurations/us-west-2/configuration.json` を入力します。

    g. [次へ] を選択します。

12. ステップ 7: 情報を確認してから、パイプラインの作成を選択します。

ステップ 4 : 任意のソースファイルに変更を加えてデプロイを確認します。

ソースファイルに変更を加え、変更をバケットにアップロードします。これにより、パイプラインの実行がトリガーされます。バージョンを表示して、設定が使用可能であることを確認します。

チュートリアル: CodeCommit パイプラインソースで完全なクローンを使用する

CodePipeline で GitHub ソースアクションの完全なクローンオプションを選択できます。このオプションを使用して、パイプラインビルドアクションで Git メタデータの CodeBuild コマンドを実行します。

> **Note**
>
> ここで説明する完全クローンオプションは、CodePipeline でリポジトリメタデータをクローンするかどうかを指定するもので、CodeBuild コマンドでのみ使用できます。CodeBuild プロジェクトで使用する GitHub ユーザーアクセストークンを使用するには、以下の手順に従って AWS Connector for GitHub アプリをインストールし、アプリのインストールフィールドを空のままにします。 AWS CodeConnections は接続にユーザーアクセストークンを使用します。

> **Important**
>
> パイプライン作成の一環として、CodePipeline は、ユーザーが指定した S3 アーティファクトバケットをアーティファクトとして使用します (これは S3 ソースアクションで使用するバケットとは異なります)。S3 アーティファクトバケットがパイプラインのアカウントとは異なるアカウントにある場合は、S3 アーティファクトバケットが によって所有 AWS アカウント されており、安全で信頼できることを確認してください。

このチュートリアルでは、GitHub リポジトリに接続し、ソースデータに完全なクローンオプションを使用し、リポジトリをクローンし、リポジトリの Git コマンドを実行する CodeBuild ビルドを実行するパイプラインを作成します。

> **Note**
>
> この機能は、アジアパシフィック (香港) 、アフリカ (ケープタウン) 、中東 (バーレーン) 、欧州 (チューリッヒ) 、または AWS GovCloud (米国西部) の各リージョンでは使用できません。利用可能なその他のアクションについては、「CodePipeline との製品とサービスの統合」を参照してください。欧州 (ミラノ) リージョンでのこのアクションに関する考慮事項については、「CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアクションの場合)」の注意を参照してください。

トピック

- 前提条件

- ステップ 1: README ファイルを作成する
- ステップ 2: パイプラインを作成してプロジェクトをビルドする
- ステップ 3: 接続を使用するように CodeBuild サービスロールポリシーを更新する
- ステップ 4: ビルド出力でリポジトリコマンドを表示する

前提条件

開始する前に、以下を実行する必要があります。

- GitHub アカウントで GitHub リポジトリを作成します。
- GitHub の認証情報を準備してください。を使用して接続 AWS マネジメントコンソール を設定すると、GitHub 認証情報を使用してサインインするように求められます。

ステップ 1: README ファイルを作成する

GitHub リポジトリを作成したら、次のステップを使用して README ファイルを追加します。

1. GitHub リポジトリにログインし、リポジトリを選択します。
2. 新規のファイルを作成するには、ファイルの追加 > ファイルの作成 を選択します。ファイルに「`README.md`」という名前を付け、次のテキストを追加します。

   ```
   This is a GitHub repository!
   ```

3. [Commit changes] (変更のコミット) を選択します。

   `README.md` ファイルがリポジトリのルートレベルにあることを確認してください。

ステップ 2: パイプラインを作成してプロジェクトをビルドする

このセクションでは、次のアクションを使用してパイプラインを作成します。

- Bitbucket リポジトリとアクションへの接続を持つソースステージ。
- ビルドアクションを含む AWS CodeBuild ビルドステージ。

ウィザードを使用してパイプラインを作成するには

1. CodePipeline コンソール (http://console.aws.amazon.com/codesuite/codepipeline/home) にサインインします。
2. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。
3. [ステップ 1: 作成オプションを選択する] ページの [作成オプション] で、[カスタムパイプラインを構築する] オプションを選択します。[次へ] を選択します。
4. [ステップ 2: パイプラインの設定を選択する] で、[パイプライン名] に「**MyGitHubPipeline**」と入力します。
5. このチュートリアルの目的では、[パイプラインタイプ] で、[V1] を選択します。[V2] を選択することもできますが、パイプラインタイプは特性と価格が異なることに注意してください。詳細については、「パイプラインのタイプ」を参照してください。
6. [サービスロール] で、[New service role (新しいサービスロール)] を選択します。

   > **Note**
   >
   > 既存の CodePipeline サービスロールを代わりに使用する場合は、サービスロールポリシーに対する `codestar-connections:UseConnection` IAM アクセス許可を追加したことを確認してください。CodePipeline サービスロールの手順については、CodePipeline サービスロールにアクセス許可を追加する」を参照してください。

7. [詳細設定] では、デフォルト値のままにします。アーティファクトストアで、[Default location] (デフォルトの場所)を選択し、パイプライン用に選択したリージョン内のパイプラインのデフォルトのアーティファクトストア (デフォルトとして指定された Amazon S3 アーティファクトバケットなど) を使用します。

   > **Note**
   >
   > これはソースコードのソースバケットではありません。パイプラインのアーティファクトストアです。パイプラインごとに S3 バケットなどの個別のアーティファクトストアが必要です。

   [次へ] を選択します。
8. [ステップ 3: ソースステージの追加] ページで、ソースステージを追加します。

a. [ソースプロバイダー] で、[GitHub (GitHub アプリ経由)] を選択します。

b. 接続 で、既存の接続を選択するか、新規の接続を作成します。GitHub ソースアクション用の接続を作成または管理するには、「GitHub コネクション」を参照してください。

特定のプロバイダーへのすべての接続に対してアプリを 1 つインストールします。 AWS Connector for GitHub アプリを既にインストールしている場合は、それを選択してこのステップをスキップします。

> **Note**
>
> ユーザーアクセストークンを作成する場合は、 Connector for GitHub アプリが既にインストール AWS されていることを確認し、アプリのインストールフィールドを空のままにします。 AWS CodeConnections は接続にユーザーアクセストークンを使用します。詳細については、「CodeBuild でソースプロバイダーにアクセスする」を参照してください。

c. リポジトリ名 で、GitHub リポジトリの名前を選択します。

d. BranchName に、使用するリポジトリブランチを入力します。

e. [ソースコードの変更時にパイプラインを開始する] オプションが選択されていることを確認します。

f. [出力アーティファクト形式] で [完全なクローン] を選択し、ソースリポジトリの Git クローンオプションを有効にします。Git クローンオプションを使用できるのは、CodeBuild によって提供されるアクションだけです。このチュートリアルでは、このオプションを使用するための CodeBuild プロジェクトサービスロールの許可を更新するために ステップ 3: 接続を使用するように CodeBuild サービスロールポリシーを更新する を使用します。

[次へ] を選択します。

9. [ステップ 4: ビルドステージの追加] で、ビルドステージを追加します。

a. [ビルドプロバイダ] で、[AWS CodeBuild] を選択します。[リージョン] をデフォルトでパイプラインのリージョンにすることを許可します。

b. [プロジェクトを作成] を選択します。

c. [プロジェクト名] に、このビルドプロジェクトの名前を入力します。

d. [環境イメージ] で、[Managed image (マネージド型イメージ)] を選択します。[Operating system] で、[Ubuntu] を選択します。

e. [ランタイム] で、[Standard (標準)] を選択します。[イメージ] で、[aws/codebuild/standard:5.0] を選択します。

f. [サービスロール] で、[New service role (新しいサービスロール)] を選択します。

> **Note**
>
> CodeBuild サービスロールの名前を書き留めます。このチュートリアルの最後のステップでは、ロール名が必要になります。

g. [Buildspec] の Build specifications (ビルド仕様) で、[Insert build commands] (ビルドコマンドの挿入) を選択します。エディタに切り替え を選択し、ビルドコマンド に以下を貼り付けます。

> **Note**
>
> ビルド仕様の env セクションで、この例に示すように、git コマンドの認証情報ヘルパーが有効になっていることを確認します。

```
version: 0.2

env:
  git-credential-helper: yes
phases:
  install:
    #If you use the Ubuntu standard image 2.0 or later, you must specify
 runtime-versions.
    #If you specify runtime-versions and use an image other than Ubuntu
 standard image 2.0, the build fails.
    runtime-versions:
      nodejs: 12
      # name: version
    #commands:
      # - command
      # - command
  pre_build:
    commands:
      - ls -lt
      - cat README.md
  build:
```

```
    commands:
      - git log | head -100
      - git status
      - ls
      - git archive --format=zip HEAD > application.zip
  #post_build:
    #commands:
      # - command
      # - command
artifacts:
  files:
    - application.zip
    # - location
  #name: $(date +%Y-%m-%d)
  #discard-paths: yes
  #base-directory: location
#cache:
  #paths:
    # - paths
```

h. [Continue to CodePipeline] (CodePipeline に進む) を選択します。CodePipeline コンソールに戻り、ビルドコマンドを使用して設定する CodeBuild プロジェクトが作成されます。ビルドプロジェクトでは、サービスロールを使用して AWS のサービス アクセス許可を管理します。このステップには数分かかる場合があります。

i. [次へ] を選択します。

10. [ステップ 5: テストステージの追加] で、[テストステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

    [次へ] を選択します。

11. [ステップ 6: デプロイステージの追加] ページで、[デプロイステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。[次へ] を選択します。

12. ステップ 7: 確認で、パイプラインの作成を選択します。

ステップ 3: 接続を使用するように CodeBuild サービスロールポリシーを更新する

CodeBuild サービスロールが接続使用許可の更新をする必要があるため、最初のパイプラインの実行は失敗します。`codestar-connections:UseConnection` IAM 許可をサービスロールポリシーに追加します。IAM コンソールでポリシーを更新する手順については、Bitbucket、GitHub、GitHub

Enterprise Server、または GitLab.com に接続するための CodeBuild GitClone アクセス許可を追加します。 を参照してください。

ステップ 4: ビルド出力でリポジトリコマンドを表示する

1. サービスロールが正常に更新されたら、失敗した CodeBuild ステージで 再試行 を選択します。
2. パイプラインが正常に実行されたら、成功したビルドステージで [詳細を表示] を選択します。

   詳細ページで、[ログ] タブを選択します。CodeBuild ビルド出力を表示します。このコマンドは、入力された変数の値を出力します。

   コマンドは、`README.md` ファイルの内容を出力し、ディレクトリ内のファイルを一覧表示し、リポジトリのクローンを作成し、ログを表示し、リポジトリを ZIP ファイルとしてアーカイブします。

チュートリアル: CodeCommit パイプラインソースでフルクローンを使用する

CodePipeline で CodeCommit ソースアクションの完全なクローンオプションを選択できます。このオプションを使用して、CodeBuild がパイプライン構築アクションで Git メタデータにアクセスできるようにします。

このチュートリアルでは、CodeCommit リポジトリにアクセスし、ソースデータの完全なクローンオプションを使用し、リポジトリのクローンを作成してリポジトリの Git コマンドを実行する CodeBuild 構築を実行するパイプラインを作成します。

> **Note**
>
> CodeBuild アクションは、Git クローンオプションで使用できる Git メタデータの使用をサポートする唯一のダウンストリームアクションです。また、パイプラインにクロスアカウントアクションを含めることはできますが、フルクローンオプションを成功させるには、CodeCommit アクションと CodeBuild アクションを同じアカウントに含める必要があります。

> **Important**
>
> パイプライン作成の一環として、CodePipeline は、ユーザーが指定した S3 アーティファクトバケットをアーティファクトとして使用します (これは S3 ソースアクションで使用するバケットとは異なります)。S3 アーティファクトバケットがパイプラインのアカウントとは異なるアカウントにある場合は、S3 アーティファクトバケットが によって所有 AWS アカウント されており、安全で信頼できることを確認してください。

トピック

- 前提条件
- ステップ 1 : README ファイルを作成する
- ステップ 2: パイプラインを作成してプロジェクトをビルドする
- ステップ 3: CodeBuild サービスロールポリシーを更新してリポジトリをクローンする
- ステップ 4: 構築出力でリポジトリコマンドを表示する

前提条件

開始する前に、パイプラインと同じ AWS アカウントとリージョンに CodeCommit リポジトリを作成する必要があります。

ステップ 1 : README ファイルを作成する

これらのステップを使用して、README ファイルをソースリポジトリに追加します。README ファイルは、CodeBuild ダウンストリームアクションが読み取るためのサンプルソースファイルを提供します。

README ファイルを追加するには

1. リポジトリにログインし、リポジトリを選択します。
2. 新規のファイルを作成するには、ファイルの追加 > ファイルの作成 を選択します。ファイルに「`README.md`」という名前を付け、次のテキストを追加します。

   ```
   This is a CodeCommit repository!
   ```

3. [Commit changes] (変更のコミット) を選択します。

README.md ファイルがリポジトリのルートレベルにあることを確認してください。

ステップ 2: パイプラインを作成してプロジェクトをビルドする

このセクションでは、次のアクションを使用してパイプラインを作成します。

- CodeCommit ソースアクションを持つソースステージ。
- ビルドアクションを含む AWS CodeBuild ビルドステージ。

ウィザードを使用してパイプラインを作成するには

1. CodePipeline コンソール (http://console.aws.amazon.com/codesuite/codepipeline/home) にサインインします。
2. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。
3. [ステップ 1: 作成オプションを選択する] ページの [作成オプション] で、[カスタムパイプラインを構築する] オプションを選択します。[次へ] を選択します。
4. [ステップ 2: パイプラインの設定を選択する] で、[パイプライン名] に「**MyCodeCommitPipeline**」と入力します。
5. CodePipeline は、特徴と料金が異なる V1 タイプと V2 タイプのパイプラインを提供しています。V2 タイプは、コンソールで選択できる唯一のタイプです。詳細については、「パイプラインタイプ」を参照してください。CodePipeline の料金については、料金を参照してください。
6. [Service role (サービスロール)] で、次のいずれかの操作を行います。
   - [Existing service role (既存のサービスロール)] を選択します。
   - 既存の CodePipeline サービスロールを選択します。このロールには、サービスロールポリシーに対する codecommit:GetRepository IAM 許可が必要です。CodePipeline サービスロールにアクセス許可を追加する」を参照してください。
7. [詳細設定] では、デフォルト値のままにします。[次へ] を選択します。
8. [ステップ 3: ソースステージの追加] ページで、次の操作を行います。
   a. ソースプロバイダ で、CodeCommit を選択します。
   b. リポジトリ名 で、 リポジトリの名前を選択します。
   c. ブランチ名 で、ブランチ名を選択します。

d. [ソースコードの変更時にパイプラインを開始する] オプションが選択されていることを確認します。

e. [出力アーティファクト形式] で [完全なクローン] を選択し、ソースリポジトリの Git クローンオプションを有効にします。Git クローンオプションを使用できるのは、CodeBuild によって提供されるアクションだけです。

[次へ] を選択します。

9. [ステップ 4: ビルドステージの追加] で、以下を行います。

a. [ビルドプロバイダ] で、[AWS CodeBuild] を選択します。[リージョン] をデフォルトでパイプラインのリージョンにすることを許可します。

b. [プロジェクトを作成] を選択します。

c. [プロジェクト名] に、このビルドプロジェクトの名前を入力します。

d. [環境イメージ] で、[Managed image (マネージド型イメージ)] を選択します。[Operating system] で、[Ubuntu] を選択します。

e. [ランタイム] で、[Standard (標準)] を選択します。[イメージ] で、[aws/codebuild/standard:5.0] を選択します。

f. [サービスロール] で、[New service role (新しいサービスロール)] を選択します。

> **Note**
>
> CodeBuild サービスロールの名前を書き留めます。このチュートリアルの最後のステップでは、ロール名が必要になります。

g. [Buildspec] の Build specifications (ビルド仕様) で、[Insert build commands] (ビルドコマンドの挿入) を選択します。エディタに切り替え を選択し、構築コマンド の下に次のコードを貼り付けます。

```
version: 0.2

env:
  git-credential-helper: yes
phases:
  install:
    #If you use the Ubuntu standard image 2.0 or later, you must specify
 runtime-versions.
```

```
    #If you specify runtime-versions and use an image other than Ubuntu
 standard image 2.0, the build fails.
    runtime-versions:
      nodejs: 12
      # name: version
    #commands:
      # - command
      # - command
  pre_build:
    commands:
      - ls -lt
      - cat README.md
  build:
    commands:
      - git log | head -100
      - git status
      - ls
      - git describe --all
  #post_build:
    #commands:
      # - command
      # - command
#artifacts:
  #files:
    # - location
  #name: $(date +%Y-%m-%d)
  #discard-paths: yes
  #base-directory: location
#cache:
  #paths:
    # - paths
```

h. [Continue to CodePipeline] (CodePipeline に進む) を選択します。これにより、CodePipeline コンソールに戻り、構築コマンドを使用して構成する CodeBuild プロジェクトが作成されます。ビルドプロジェクトでは、サービスロールを使用して AWS のサービス のアクセス許可を管理します。このステップには数分かかる場合があります。

i. [次へ] を選択します。

10. [ステップ 5: テストステージの追加] で、[テストステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

    [次へ] を選択します。

11. [ステップ 6: デプロイステージの追加] ページで、[デプロイステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。[次へ] を選択します。
12. ステップ 7: 確認で、パイプラインの作成を選択します。

ステップ 3: CodeBuild サービスロールポリシーを更新してリポジトリをクローンする

リポジトリからプルを許可した CodeBuild サービスロールを更新する必要があるため、最初のパイプラインの実行は失敗します。

`codecommit:GitPull` IAM 許可をサービスロールポリシーに追加します。IAM コンソールでポリシーを更新する手順については、CodeBuild GitClone のアクセス権限を CodeCommit ソースアクションに追加します。 を参照してください。

ステップ 4: 構築出力でリポジトリコマンドを表示する

ビルド出力 を表示するには

1. サービスロールが正常に更新されたら、失敗した CodeBuild ステージで 再試行 を選択します。
2. パイプラインが正常に実行されたら、成功したビルドステージで [詳細を表示] を選択します。

   詳細ページで、[ログ] タブを選択します。CodeBuild ビルド出力を表示します。このコマンドは、入力された変数の値を出力します。

   コマンドは、ファイルの内容を出力し、ディレクトリ内の `README.md` ファイルを一覧表示し、リポジトリのクローンを作成し、ログを表示して、`git describe --all` を実行します。

チュートリアル: AWS CloudFormation StackSets デプロイアクションを使用してパイプラインを作成する

このチュートリアルでは、 AWS CodePipeline コンソールを使用して、スタックセットとスタックインスタンスを作成するためのデプロイアクションを含むパイプラインを作成します。パイプラインが実行されると、テンプレートはスタックセットを作成し、スタックセットをデプロイするインスタンスを作成および更新します。

> **Important**
>
> パイプライン作成の一環として、CodePipeline は、ユーザーが指定した S3 アーティファクトバケットをアーティファクトとして使用します (これは S3 ソースアクションで使用するバケットとは異なります)。S3 アーティファクトバケットがパイプラインのアカウントとは異なるアカウントにある場合は、S3 アーティファクトバケットが によって所有 AWS アカウント されており、安全で信頼できることを確認してください。

スタックセットのアクセス許可を管理するには、セルフマネージド IAM ロールとマネージド IAM ロール AWSの 2 つの方法があります。このチュートリアルでは、セルフマネージド型の許可の例を示します。

CodePipeline で StackSets を最も効果的に使用するには、 CloudFormation StackSets の背後にある概念とその仕組みを明確に理解しておく必要があります。AWS CloudFormation ユーザーガイドの「StackSets の概念」を参照してください。

トピック

- 前提条件
- ステップ 1: サンプル AWS CloudFormation テンプレートとパラメータファイルをアップロードする
- ステップ 2: パイプラインを作成する
- ステップ 3: 初期デプロイを表示する
- ステップ 4: CloudFormationsStackInstances アクションを追加する
- ステップ 5: デプロイのスタックセットリソースを表示する
- ステップ 6: スタックセットを更新する

前提条件

スタックセットオペレーションでは、管理者アカウントとターゲットアカウントの 2 つの異なるアカウントを使用します。管理者アカウントでは、スタックセットを作成します。ターゲットアカウントでは、スタックセットに属する個別のスタックを作成します。

管理者アカウントで管理者ロールを作成するには

- 「スタックセットオペレーションの基本アクセス許可の設定」の手順に従います。ロールは **AWSCloudFormationStackSetAdministrationRole** という名前にする必要があります。

ターゲットアカウントにサービスロールを作成するには

- 管理者アカウントを信頼するターゲットアカウントにサービスロールを作成します。「スタックセットオペレーションの基本アクセス許可の設定」の手順に従います。ロールは **AWSCloudFormationStackSetExecutionRole** という名前にする必要があります。

ステップ 1: サンプル AWS CloudFormation テンプレートとパラメータファイルをアップロードする

スタックセットテンプレートとパラメータファイルのソースバケットを作成します。サンプル AWS CloudFormation テンプレートファイルをダウンロードし、パラメータファイルを設定し、ファイルを圧縮してから S3 ソースバケットにアップロードします。

> **Note**
>
> ソースファイルが唯一のテンプレートであっても、S3 ソースバケットにアップロードする前に、必ずソースファイルを圧縮してください。

S3 ソースバケットを作成するには

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/s3/ で Amazon S3 コンソールを開きます。
2. [バケットを作成] を選択します。
3. [バケット名] にバケットの名前を入力します。

   [リージョン] で、パイプラインを作成するリージョンを選択します。[バケットを作成] を選択します。
4. バケットが作成されると、成功バナーが表示されます。[バケットの詳細に移動] を選択します。
5. [プロパティ] タブで、[バージョニング] を選択します。[バージョニングの有効化] を選択し、[保存] を選択します。

AWS CloudFormation テンプレートファイルを作成するには

1. スタックセットの CloudTrail 設定を生成するために、サンプルテンプレートファイル https://s3.amazonaws.com/cloudformation-stackset-sample-templates-us-east-1/EnableAWSCloudtrail.yml をダウンロードします。
2. `template.yml` という名前でファイルを保存します。

parameters.txt ファイルを作成するには

1. デプロイのパラメータでファイルを作成します。パラメータは、実行時にスタック内で更新する値です。次のサンプルファイルは、スタックセットのテンプレートパラメータを更新して、ログ記録検証とグローバルイベントを有効にします。

```
[
  {
    "ParameterKey": "EnableLogFileValidation",
    "ParameterValue": "true"
  },
  {
    "ParameterKey": "IncludeGlobalEvents",
    "ParameterValue": "true"
  }
]
```

2. `parameters.txt` という名前でファイルを保存します。

accounts.txt ファイルを作成するには

1. 次のサンプルファイルに示されているように、インスタンスを作成するアカウントでファイルを作成します。

```
[
    "111111222222","333333444444"
]
```

2. `accounts.txt` という名前でファイルを保存します。

ソースファイルを作成してアップロードするには

1. ファイルを単一の zip ファイルに結合します。ファイルは zip ファイルで以下のようになっています。

```
template.yml
parameters.txt
accounts.txt
```

2. zip ファイルを S3 バケットにアップロードします。このファイルは、CodePipeline でのデプロイアクションのために [パイプラインを作成する] ウィザードによって作成されたソースアーティファクトです。

ステップ 2: パイプラインを作成する

このセクションでは、次のアクションを使用してパイプラインを作成します。

- ソースアーティファクトがテンプレートファイルやサポートソースファイルである S3 ソースアクションのあるソースステージ。
- CloudFormation スタックセットを作成するスタックセットデプロイアクションを含むデプロイステージ。
- ターゲットアカウント内に CloudFormation スタックとインスタンスを作成するスタックインスタンスのデプロイアクションを含むデプロイステージ。

CloudFormationStackSet アクションを使用してパイプラインを作成するには

1. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。
3. [ステップ 1: 作成オプションを選択する] ページの [作成オプション] で、[カスタムパイプラインを構築する] オプションを選択します。[次へ] を選択します。
4. [ステップ 2: パイプラインの設定を選択する] で、[パイプライン名] に「**MyStackSetsPipeline**」と入力します。
5. このチュートリアルの目的では、[パイプラインタイプ] で、[V1] を選択します。[V2] を選択することもできますが、パイプラインタイプは特性と価格が異なることに注意してください。詳細については、「パイプラインのタイプ」を参照してください。

6. [サービスロール] で、[新しいサービスロール] を選択し、CodePipeline に IAM でのサービスロールの作成を許可します。
7. [アーティファクトストア] では、デフォルト値はそのままにしておきます。

> **Note**
>
> これはソースコードのソースバケットではありません。パイプラインのアーティファクトストアです。パイプラインごとに S3 バケットなどの個別のアーティファクトストアが必要です。パイプラインを作成または編集するときは、パイプラインリージョンにアーティファクトバケットと、アクションを実行している AWS リージョンごとに 1 つのアーティファクトバケットが必要です。
> 詳細については、「入力および出力アーティファクト」および「CodePipeline パイプライン構造リファレンス」を参照してください。

[次へ] を選択します。

8. [ステップ 3: ソースステージの追加] ページの [ソースプロバイダー] で、[Amazon S3] を選択します。
9. [バケット] に、このチュートリアル用に作成した S3 ソースバケット (`BucketName` など) を入力します。[S3 オブジェクトキー] に、zip ファイルのファイルパスとファイル名 (`MyFiles.zip` など) を入力します。
10. [次へ] を選択します。
11. [ステップ 4: ビルドステージの追加] で、[ビルドステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

    [次へ] を選択します。
12. [ステップ 5: テストステージの追加] で、[テストステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

    [次へ] を選択します。
13. ステップ 6: デプロイステージの追加:

    a. [デプロイプロバイダー] で、[AWS CloudFormation スタックセット] を選択します。
    b. [スタックセット名] に、スタックセットの名前を入力します。これは、テンプレートが作成するスタックセットの名前です。

> **Note**
>
> スタックセット名を記録します。この名前は、パイプラインに 2 番目の StackSets デプロイアクションを追加するときに使用します。

c. [テンプレートパス] に、テンプレートファイルをアップロードしたアーティファクト名とファイルパスを入力します。例えば、デフォルトのソースアーティファクト名 SourceArtifact を使用して次のように入力します。

```
SourceArtifact::template.yml
```

d. [デプロイターゲット] に、アカウントファイルをアップロードしたアーティファクト名とファイルパスを入力します。例えば、デフォルトのソースアーティファクト名 SourceArtifact を使用して次のように入力します。

```
SourceArtifact::accounts.txt
```

e. デプロイターゲット AWS リージョンで、 など、最初のスタックインスタンスをデプロイするリージョンを 1 つ入力しますus-east-1。

f. [デプロイオプション] を拡張します。[パラメータ] に、パラメータファイルをアップロードしたアーティファクト名とファイルパスを入力します。例えば、デフォルトのソースアーティファクト名 SourceArtifact を使用して次のように入力します。

```
SourceArtifact::parameters.txt
```

パラメータをファイルパスではなく、リテラル入力として入力するには、次のように入力します。

```
ParameterKey=EnableLogFileValidation,ParameterValue=true
ParameterKey=IncludeGlobalEvents,ParameterValue=true
```

g. [Capabilities] (機能) で、[CAPABILITY_IAM] と [CAPABILITY_NAMED_IAM] を選択します。

h. [アクセス許可モデル] で、[SELF_MANAGED] を選択します。

i. [障害耐性の割合] に「20」と入力します。

j. [最大同時割合] に「25」と入力します。

k. [次へ] を選択します。

l. [ステップ 7: レビュー] で、[パイプラインを作成する] を選択します。パイプラインが表示されます。

m. パイプラインの実行を許可します。

ステップ 3: 初期デプロイを表示する

初期デプロイのリソースとステータスを表示します。デプロイでスタックセットが正常に作成されたことを確認したら、2 番目のアクションを [デプロイ] ステージに追加します。

リソースを表示するには

1. CodePipeline コンソール (https://console.aws.amazon.com/codepipeline/) を開きます。
2. [パイプライン] で、パイプラインを選択してから、[表示] を選択します。この図は、パイプラインのソースとデプロイのステージを示しています。
3. パイプラインの CloudFormationStackSet CloudFormation アクションで アクションを選択します。スタックセットのテンプレート、リソース、イベントが CloudFormation コンソールに表示されます。
4. 左のナビゲーションメニューから [StackSets] を選択します。リストで、新しいスタックセットを選択します。
5. [スタックインスタンス] タブを選択します。us-east-1 リージョンでは、提供したアカウントごとに 1 つのスタックインスタンスが作成されていることを確認します。各スタックインスタンスのステータスが CURRENT になっていることを確認します。

ステップ 4: CloudFormationsStackInstances アクションを追加する

パイプラインに次のアクションを作成し、 CloudFormation StackSets が残りのスタックインスタンスを作成できるようにします。

パイプラインで次のアクションを作成するには

1. CodePipeline コンソール (https://console.aws.amazon.com/codepipeline/) を開きます。

   [パイプライン] で、パイプラインを選択してから、[表示] を選択します。この図は、パイプラインのソースとデプロイのステージを示しています。
2. パイプラインの編集を選択します。パイプラインは [編集] モードで表示されます。
3. [デプロイ] ステージで、[編集] を選択します。

4. [AWS CloudFormation スタックセット] デプロイアクションで、[アクショングループの追加] を選択します。
5. [アクションの編集] ページで、アクションの詳細を追加します。

   a. [アクション名] に、アクションの名前を入力します。

   b. [アクションプロバイダー] で、[AWS CloudFormation スタックインスタンス] を選択します。

   c. [入力アーティファクト] で、[ソースアーティファクト] を選択します。

   d. [スタックセット名] に、スタックセットの名前を入力します。これは、最初のアクションで指定したスタックセットの名前です。

   e. [デプロイターゲット] に、アカウントファイルをアップロードしたアーティファクト名とファイルパスを入力します。例えば、デフォルトのソースアーティファクト名 `SourceArtifact` を使用して次のように入力します。

   ```
   SourceArtifact::accounts.txt
   ```

   f. デプロイターゲット AWS リージョンで、 `us-east-2`や など、残りのスタックインスタンスをデプロイするリージョンを`eu-central-1`次のように入力します。

   ```
   us-east2, eu-central-1
   ```

   g. [障害耐性の割合] に「`20`」と入力します。

   h. [最大同時割合] に「`25`」と入力します。

   i. [保存] を選択します。

   j. 変更を手動でリリースします。更新されたパイプラインがデプロイステージに 2 つのアクションと共に表示されます。

ステップ 5: デプロイのスタックセットリソースを表示する

スタックセットのデプロイのリソースとステータスを表示します。

リソースを表示するには

1. CodePipeline コンソール (https://console.aws.amazon.com/codepipeline/) を開きます。
2. [パイプライン] で、パイプラインを選択してから、[表示] を選択します。この図は、パイプラインのソースとデプロイのステージを示しています。

3. パイプライン内の CloudFormation アクションの**AWS CloudFormation Stack Instances**アクションを選択します。スタックセットのテンプレート、リソース、イベントが CloudFormation コンソールに表示されます。
4. 左のナビゲーションメニューから [StackSets] を選択します。リストで、スタックセットを選択します。
5. [スタックインスタンス] タブを選択します。提供した各アカウントの残りのスタックインスタンスが、すべて想定したリージョンで作成または更新されていることを確認します。各スタックインスタンスのステータスが CURRENT になっていることを確認します。

ステップ 6: スタックセットを更新する

スタックセットを更新し、インスタンスに更新をデプロイします。この例では、更新用に指定するデプロイターゲットも変更します。更新のパートではないインスタンスは、古いステータスに移行します。

1. CodePipeline コンソール (https://console.aws.amazon.com/codepipeline/) を開きます。
2. [パイプライン] で、パイプラインを選択してから、[編集] を選択します。[デプロイ] ステージで、[編集] を選択します。
3. パイプラインで、[AWS CloudFormation スタックセット] アクションを選択して編集します。[説明] で、既存の説明をスタックセットの新しい説明に書き直します。
4. パイプラインで、[AWS CloudFormation スタックインスタンス] アクションを選択して編集します。デプロイターゲット AWS リージョンで、アクションの作成時に入力されたus-east-2値を削除します。
5. 変更を保存します。[変更のリリース] を選択して、パイプラインを実行します。
6. CloudFormationでアクションを開きます。[StackSet の情報] タブを選択します。[StackSet の説明] で、新しい説明が表示されていることを確認します。
7. [スタックインスタンス] タブを選択します。[ステータス] で、us-east-2 のスタックインスタンスのステータスが OUTDATED であることを確認します。

チュートリアル: パイプラインの変数チェックルールを入力条件として作成する

このチュートリアルでは、ソースステージで GitHub をソースアクションプロバイダーとして使用して、ファイルを継続的に配信するパイプラインを設定します。ソースリポジトリ内のソースファイル

に変更を加えると、完成したパイプラインはその変更を検出します。パイプラインは出力変数を実行して、ビルドステージへの入力条件で指定したソースリポジトリ名およびブランチ名と照合します。

> **Important**
>
> パイプライン作成の一環として、CodePipeline は、ユーザーが指定した S3 アーティファクトバケットをアーティファクトとして使用します (これは S3 ソースアクションで使用するバケットとは異なります)。S3 アーティファクトバケットがパイプラインのアカウントとは異なるアカウントにある場合は、S3 アーティファクトバケットが によって所有 AWS アカウント されており、安全で信頼できることを確認してください。

> **Important**
>
> この手順でパイプラインに追加するアクションの多くには、パイプラインを作成する前に作成する必要がある AWS リソースが含まれます。ソースアクションの AWS リソースは常に、パイプラインを作成するのと同じ AWS リージョンで作成する必要があります。例えば、米国東部 (オハイオ) リージョンにパイプラインを作成している場合、CodeCommit リポジトリは米国東部 (オハイオ) リージョンにある必要があります。
> パイプラインの作成時にクロスリージョンアクションを追加できます。クロスリージョンアクションの AWS リソースは、アクションを実行する予定のリージョンと同じ AWS リージョンに存在する必要があります。詳細については、「CodePipeline でクロスリージョンアクションを追加する」を参照してください。

この例では、GitHub (Version2) ソースアクションと CodeBuild ビルドアクションを含むサンプルのパイプラインを使用して、ビルドステージの入力条件で変数をチェックします。

前提条件

開始する前に、以下を実行する必要があります。

- GitHub アカウントで GitHub リポジトリを作成します。
- GitHub の認証情報を準備してください。を使用して接続 AWS マネジメントコンソール を設定すると、GitHub 認証情報を使用してサインインするように求められます。
- パイプラインのソースアクションとして GitHub (GitHub アプリ経由) を設定するためのリポジトリへの接続。GitHub リポジトリへの接続を作成するには、「GitHub コネクション」を参照してください。

ステップ 1: サンプルのソースファイルを作成して GitHub リポジトリに追加する

このセクションでは、パイプラインでソースステージとして使用するサンプルのソースファイルを作成してリポジトリに追加します。この例では、以下を作成して追加します。

- README.md ファイル。

GitHub リポジトリを作成したら、次の手順を使用して README ファイルを追加します。

1. GitHub リポジトリにログインし、リポジトリを選択します。
2. 新しいファイルを作成するには、[ファイルの追加]、[新しいファイルを作成] の順に選択します。ファイルに「README.md」という名前を付け、次のテキストを追加します。

   ```
   This is a GitHub repository!
   ```

3. [Commit changes] (変更のコミット) を選択します。このチュートリアルでは、次の例のように先頭文字が大文字の単語「Update」を含むコミットメッセージを追加します。

   ```
   Update to source files
   ```

   > **Note**
   >
   > 文字列のルールチェックでは、大文字と小文字が区別されます。

   README.md ファイルがリポジトリのルートレベルにあることを確認してください。

ステップ 2: パイプラインを作成する

このセクションでは、次のアクションを使用してパイプラインを作成します。

- Bitbucket リポジトリとアクションへの接続を持つソースステージ。
- CodeBuild ビルドステージ。ステージには、変数チェックルールに対するエントリ時の条件が設定されています。

ウィザードを使用してパイプラインを作成するには

1. CodePipeline コンソール (http://console.aws.amazon.com/codesuite/codepipeline/home) にサインインします。
2. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。
3. [ステップ 1: 作成オプションを選択する] ページの [作成オプション] で、[カスタムパイプラインを構築する] オプションを選択します。[次へ] を選択します。
4. [ステップ 2: パイプラインの設定を選択する] で、[パイプライン名] に「**MyVarCheckPipeline**」と入力します。
5. CodePipeline は、特徴と料金が異なる V1 タイプと V2 タイプのパイプラインを提供しています。V2 タイプは、コンソールで選択できる唯一のタイプです。詳細については、「パイプラインタイプ」を参照してください。CodePipeline の料金については、料金を参照してください。
6. [サービスロール] で、[New service role (新しいサービスロール)] を選択します。

   > **Note**
   >
   > 既存の CodePipeline サービスロールを代わりに使用する場合は、サービスロールポリシーに対する `codeconnections:UseConnection` IAM アクセス許可を追加したことを確認してください。CodePipeline サービスロールの手順については、CodePipeline サービスロールにアクセス許可を追加する」を参照してください。

7. [詳細設定] では、デフォルト値のままにします。

   [次へ] を選択します。
8. [ステップ 3: ソースステージの追加] ページで、ソースステージを追加します。

   a. [ソースプロバイダー] で、[GitHub (GitHub アプリ経由)] を選択します。

   b. 接続 で、既存の接続を選択するか、新規の接続を作成します。GitHub ソースアクション用の接続を作成または管理する方法については、GitHub コネクション を参照してください。

   c. リポジトリ名 で、GitHub リポジトリの名前を選択します。

   d. BranchName に、使用するリポジトリブランチを入力します。

   e. [トリガーなし] オプションが選択されていないことを確認します。

   [次へ] を選択します。

9. [ステップ 4: ビルドステージの追加] で、ビルドステージを追加します。

   a. [ビルドプロバイダ] で、[AWS CodeBuild] を選択します。[リージョン] をデフォルトでパイプラインのリージョンにすることを許可します。

   b. [プロジェクトを作成] を選択します。

   c. [プロジェクト名] に、このビルドプロジェクトの名前を入力します。

   d. [環境イメージ] で、[Managed image (マネージド型イメージ)] を選択します。[Operating system] で、[Ubuntu] を選択します。

   e. [ランタイム] で、[Standard (標準)] を選択します。[イメージ] で、[aws/codebuild/standard:5.0] を選択します。

   f. [サービスロール] で、[New service role (新しいサービスロール)] を選択します。

   > **Note**
   >
   > CodeBuild サービスロールの名前を書き留めます。このチュートリアルの最後のステップでは、ロール名が必要になります。

   g. [Buildspec] の Build specifications (ビルド仕様) で、[Insert build commands] (ビルドコマンドの挿入) を選択します。エディタに切り替え を選択し、ビルドコマンド に以下を貼り付けます。

```
version: 0.2
#env:
  #variables:
     # key: "value"
     # key: "value"
  #parameter-store:
     # key: "value"
     # key: "value"
  #git-credential-helper: yes
phases:
  install:
    #If you use the Ubuntu standard image 2.0 or later, you must specify
 runtime-versions.
    #If you specify runtime-versions and use an image other than Ubuntu
 standard image 2.0, the build fails.
    runtime-versions:
      nodejs: 12
    #commands:
```

```
      # - command
      # - command
  #pre_build:
    #commands:
      # - command
      # - command
  build:
    commands:
      -
  #post_build:
    #commands:
      # - command
      # - command
artifacts:
  files:
     - '*'
    # - location
  name: $(date +%Y-%m-%d)
  #discard-paths: yes
  #base-directory: location
#cache:
  #paths:
    # - paths
```

h. [Continue to CodePipeline] (CodePipeline に進む) を選択します。CodePipeline コンソールに戻り、ビルドコマンドを使用して設定する CodeBuild プロジェクトが作成されます。ビルドプロジェクトでは、サービスロールを使用して AWS のサービス アクセス許可を管理します。このステップには数分かかる場合があります。

i. [次へ] を選択します。

10. [ステップ 5: テストステージの追加] で、[テストステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

    [次へ] を選択します。

11. [ステップ 6: デプロイステージの追加] ページで、[デプロイステージをスキップ] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。[次へ] を選択します。

12. ステップ 7: 確認で、パイプラインの作成を選択します。

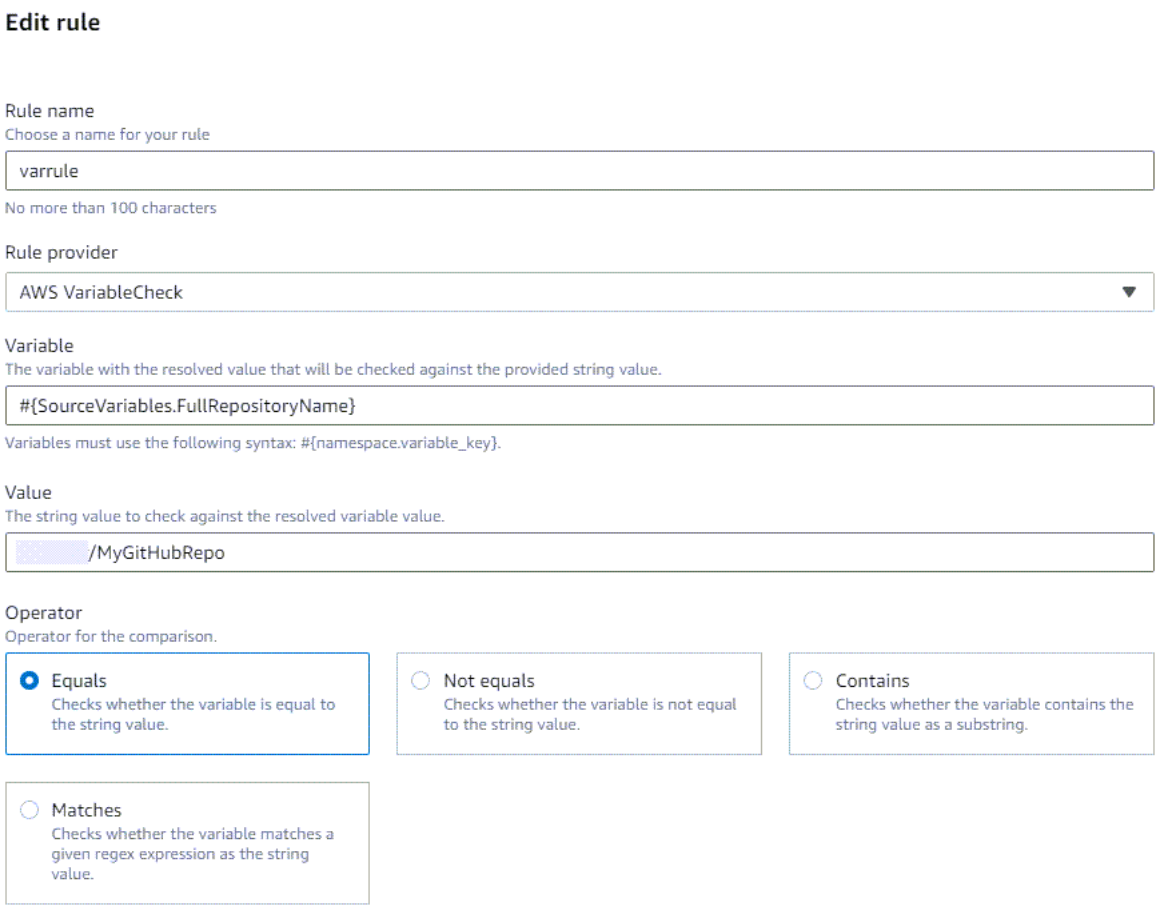
ステップ 2: ビルドステージを編集して条件とルールを追加する

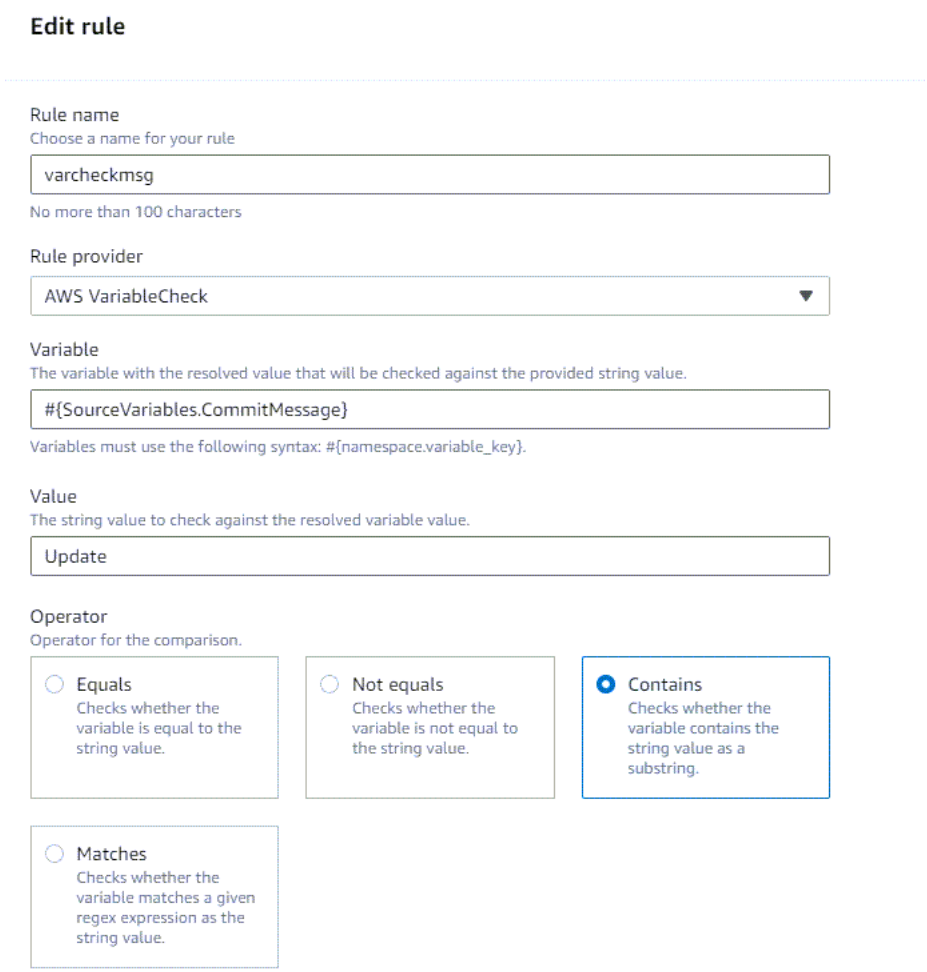
このステップでは、ステージを編集して、可変チェックルールのエントリ時の条件を追加します。

1. パイプラインを選択し、[編集] を選択します。ビルドステージでエントリルールを追加することを選択します。

   [ルールプロバイダー] で、[VariableCheck] を選択します。

2. [変数] に、チェックする変数を入力します。[値] に文字列値を入力して、解決された変数と照合します。次の画面の例では、[等しい] チェック用のルールを作成し、[含む] チェック用の別のルールを作成します。





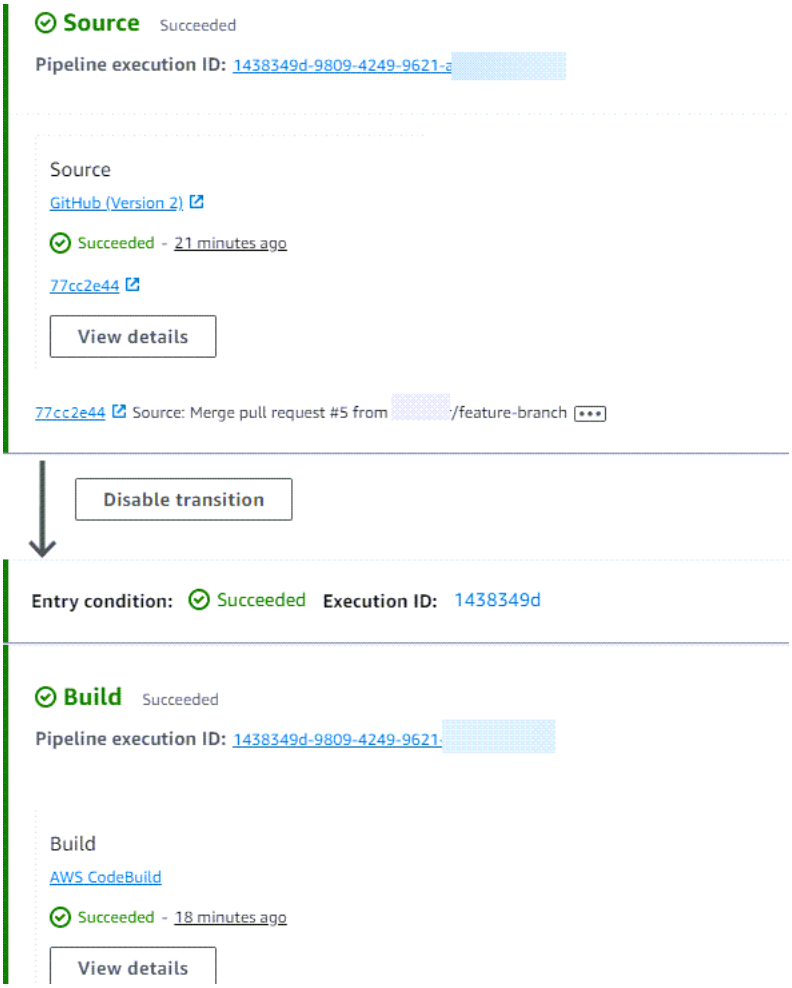
3. [保存] を選択します。

   [Done] (完了) をクリックします。

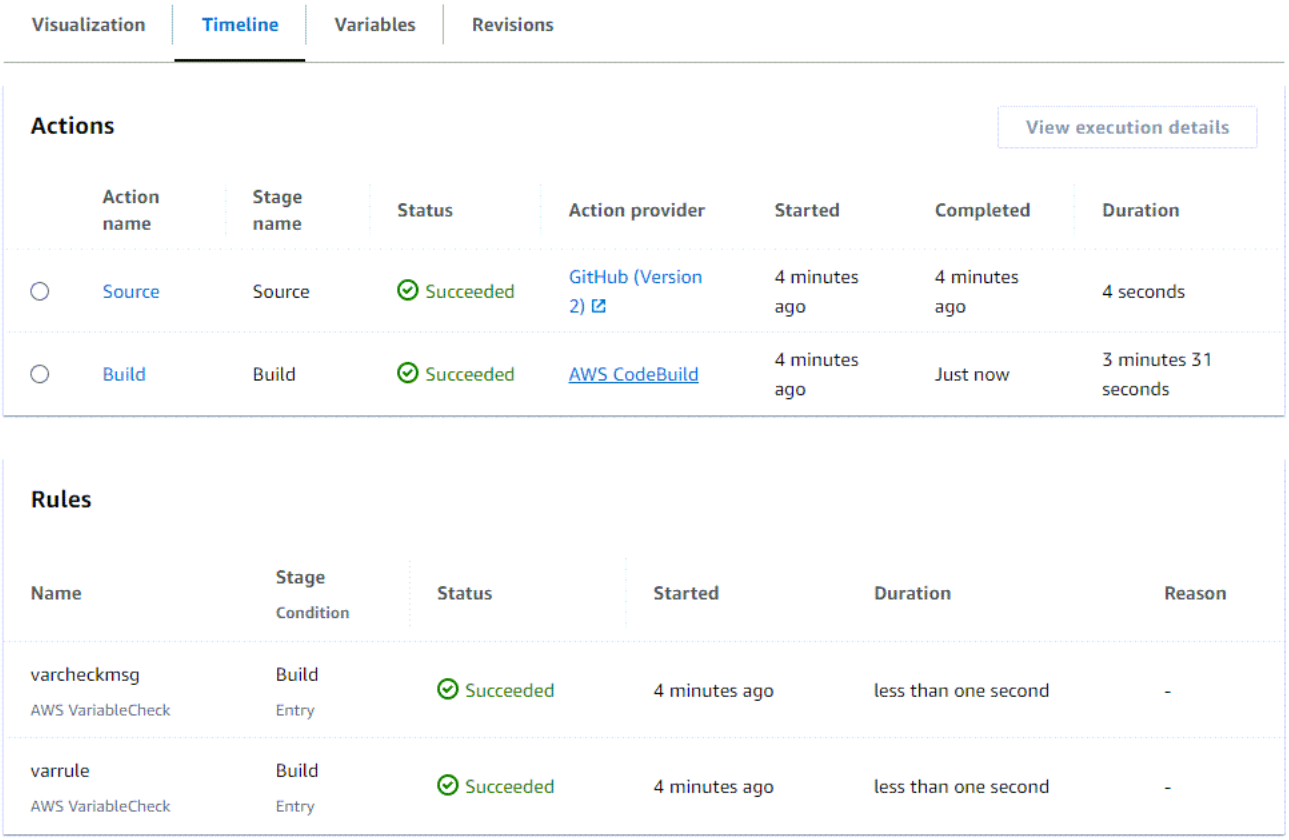
ステップ 3: パイプラインを実行し、解決された変数を表示する

このステップでは、変数チェックルールの解決された値と結果を表示します。

1. 次の例に示すように、ルールチェックが成功した後の解決された実行を表示します。



2. [タイムライン] タブで変数情報を表示します。


Visualization
Timeline
Variables
Revisions
Actions
View execution details
Action name
Stage name
Status
Action provider
Started
Completed
Duration
Source
Source
Succeeded
GitHub (Version 2)
4 minutes ago
4 minutes ago
4 seconds
Build
Build
Succeeded
AWS CodeBuild
4 minutes ago
Just now
3 minutes 31 seconds
Rules
Name
Stage
Condition
Status
Started
Duration
Reason
varcheckmsg
AWS VariableCheck
Build
Entry
Succeeded
4 minutes ago
less than one second
-
varrule
AWS VariableCheck
Build
Entry
Succeeded
4 minutes ago
less than one second
-

CodePipeline のユースケース

以下のセクションでは、CodePipeline のユースケースについて説明します。

トピック

- CodePipeline のユースケース

CodePipeline のユースケース

他の と統合するパイプラインを作成できます AWS のサービス。これらは Amazon S3 や GitHub のようなサードパーティー製品の AWS のサービスです。このセクションは CodePipeline を使用して別の製品統合を使いコードリリースを自動化する場合の例を示しています。アクションタイプ別に整理した CodePipeline との統合の一覧は、CodePipeline パイプライン構造リファレンス を参照してください。

トピック

- Amazon S3 で CodePipeline を使用する AWS CodeCommit、および Amazon S3 AWS CodeDeploy
- サードパーティーアクションプロバイダー (GitHub や Jenkins) で CodePipeline を使用する
- CodePipeline を使用して、CodeBuild でコードをコンパイル、ビルド、テストする
- CodePipeline で Amazon ECS を使用してクラウドにコンテナベースのアプリケーションを継続的に配信する
- Elastic Beanstalk で CodePipeline を使用してクラウドにウェブアプリケーションを継続的にデリバリーする
- で CodePipeline を使用して Lambda ベースのアプリケーションとサーバーレスアプリケーションの AWS Lambda 継続的な配信を行う
- CloudFormation テンプレートで CodePipeline を使用してクラウドに継続的に配信する

Amazon S3 で CodePipeline を使用する AWS CodeCommit、および Amazon S3 AWS CodeDeploy

パイプラインを作成すると、CodePipeline はパイプラインの各ステージでアクションプロバイダーとして機能する AWS 製品やサービスと統合されます。ウィザードでステージを選択する場合は、

ソースステージそしてビルドまたはデプロイステージを少なくても 1 つ選ぶ必要があります。ウィザードは変更することができないデフォルト名を使用してステージを作成します。こうしたステージの名前は、ウィザードで 3 つの完全なステージをセットアップした際に作成されたものです。

- 「ソース」というデフォルト名を使用したソースアクションステージ
- 「ビルド」というデフォルト名を使用したビルドアクションステージ
- 「ステージング」というデフォルト名を使用したデプロイアクションステージ

このガイドのチュートリアルを使用してパイプラインを作成しステージを指定できます。

- チュートリアル: シンプルなパイプラインを作成する (S3 バケット) のステップは、ウィザードを使用して Amazon S3 リポジトリがソースプロバイダーとなる「ソース」と「ステージング」という 2 つのデフォルトステージを含むパイプラインの作成をサポートします。このチュートリアルでは、 を使用して Amazon S3 バケットから Amazon Linux を実行している Amazon EC2 インスタンスにサンプルアプリケーションを AWS CodeDeploy デプロイするパイプラインを作成します。
- のステップは、ウィザードを使用して、 AWS CodeCommit リポジトリをソースプロバイダーとして使用する「ソース」ステージでパイプラインを作成するチュートリアル: シンプルなパイプラインを作成する (CodeCommit リポジトリ)のに役立ちます。このチュートリアルでは、 AWS CodeDeploy を使用してサンプルアプリケーションを AWS CodeCommit リポジトリから Amazon Linux を実行している Amazon EC2 インスタンスにデプロイするパイプラインを作成します。

サードパーティーアクションプロバイダー (GitHub や Jenkins) で CodePipeline を使用する

GitHub や Jenkins といったサードパーティー製品と統合するパイプラインを作成できます。チュートリアル: 4 ステージのパイプラインを作成する のステップは、次の操作を実行するパイプラインの作成方法を示しています。

- GitHub リポジトリからソースコードを取得、
- Jenkins を使用してソースコードの構築とテストを実行、
- AWS CodeDeploy を使用して、Amazon Linux または Microsoft Windows Server を実行している Amazon EC2 インスタンスに、構築およびテスト済みのソースコードをデプロイします。

CodePipeline を使用して、CodeBuild でコードをコンパイル、ビルド、テストする

CodeBuild はクラウドにあるマネージド型のビルドサービスで、サーバーやシステムを必要とせずにコードを構築したりテストを実行できるようにします。CodePipeline と CodeBuild を使用すると、ソースコードに変更があるたびにパイプラインを介してリビジョンの実行を自動化し、ソフトウェアのビルドを継続的にデリバリーすることができます。詳しくは、「CodePipeline と CodeBuild を使って、コードのテストとビルドを実行する]を参照してください。

CodePipeline で Amazon ECS を使用してクラウドにコンテナベースのアプリケーションを継続的に配信する

Amazon ECS はコンテナ管理サービスで、クラウド内の Amazon ECS インスタンスにコンテナベースのアプリケーションをデプロイできるようにします。Amazon ECS と CodePipeline を使用して、ソースイメージのリポジトリに変更があるたびにパイプラインを介してコンテナベースのアプリケーションのデプロイを継続的に実行できるようにするため、リビジョンの実行を自動化できます。詳細については、「チュートリアル: CodePipeline を使用した継続的なデプロイ」 を参照してください。

Elastic Beanstalk で CodePipeline を使用してクラウドにウェブアプリケーションを継続的にデリバリーする

Elastic Beanstalk はウェブサーバーでウェブアプリケーションとサービスをデプロイできるようにするコンピューティングサービスです。CodePipeline と Elastic Beanstalk を使用してアプリケーション環境でウェブアプリケーションを継続的にデプロイします。 AWS CodeStar を使用して、Elastic Beanstalk デプロイアクションでパイプラインを作成することもできます。

で CodePipeline を使用して Lambda ベースのアプリケーションとサーバーレスアプリケーションの AWS Lambda 継続的な配信を行う

「サーバーレスアプリケーションのデプロイ」で説明されているように、CodePipeline AWS Lambda で を使用して AWS Lambda 関数を呼び出すことができます。 AWS Lambda および を使用して AWS CodeStar 、サーバーレスアプリケーションをデプロイするためのパイプラインを作成することもできます。

CloudFormation テンプレートで CodePipeline を使用してクラウドに継続的に配信する

CodePipeline CloudFormation で を使用して、継続的な配信と自動化を行うことができます。詳細については、CodePipeline を使用した継続的デリバリー」を参照してください。 CloudFormation は、で作成されたパイプラインのテンプレートの作成にも使用されます AWS CodeStar。

Amazon Virtual Private Cloud で CodePipeline を使用

AWS CodePipeline は、 を搭載した Amazon Virtual Private Cloud (Amazon VPC) エンドポイントをサポートするようになりましたAWS PrivateLink。つまり、VPC 内のプライベートエンドポイントを介して CodePipeline に直接接続し、VPC と AWS ネットワーク内にすべてのトラフィックを保持できます。

Amazon VPC AWS のサービス 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 です。VPC では、次のようなネットワーク設定を管理することができます。

- 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

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 AWS のサービス を使用する間のプライベート通信を容易にする AWS テクノロジーを搭載しています。VPC を CodePipeline に接続するには、CodePipeline のインターフェイス VPC エンドポイントを定義します。このタイプのエンドポイントにより、VPC を AWS のサービスに接続できるようになります。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 CodePipeline への接続を提供します。VPC を設定する方法の詳細については、VPC ユーザーガイド参照してください。

利用可能な状況

CodePipeline は現在、以下の VPC エンドポイントをサポートしています AWS リージョン。

- 米国東部(オハイオ)
- 米国東部 (バージニア北部)
- 米国西部 (北カリフォルニア)
- 米国西部 (オレゴン)
- カナダ (中部)
- 欧州 (フランクフルト)

- 欧州 (アイルランド)
- 欧州 (ロンドン)
- ヨーロッパ (ミラノ)*
- 欧州 (パリ)
- 欧州 (ストックホルム)
- アジアパシフィック (香港)*
- アジアパシフィック (ムンバイ)
- アジアパシフィック (東京)
- アジアパシフィック (ソウル)
- アジアパシフィック (シンガポール)
- アジアパシフィック (シドニー)
- 南米 (サンパウロ)
- AWS GovCloud (米国西部)

* 使用する前に、このリージョンを有効にする必要があります。

CodePipeline 用の VPC エンドポイントポリシーを作成する

Amazon VPC コンソールを使用して、com.amazonaws.***region***. codepipeline VPC エンドポイントを作成します。コンソールでは、*region* は、米国東部 (オハイオ) リージョンなど、CodePipeline で AWS リージョン サポートされている `us-east-2`のリージョン識別子です。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。

エンドポイントには、 AWSにサインインしたときに指定したリージョンが事前に設定されています。別のリージョンにサインインすると、VPC エンドポイントは新しいリージョンに更新されます。

> **Note**
>
> VPC サポートを提供し、CodeCommit などの CodePipeline と統合 AWS のサービス する他の は、その統合に Amazon VPC エンドポイントを使用することをサポートしていない場合があります。例えば、CodePipeline と CodeCommit の間のトラフィックを VPC サブネット範囲に制限することはできません。

VPC 設定のトラブルシューティング

VPC の問題をトラブルシューティングする場合、インターネット接続エラーメッセージに表示される情報を、問題の特定、診断、対処のために使用します。

1. インターネットゲートウェイが VPC にアタッチされていることを確認します。
2. パブリックサブネットのルートテーブルがインターネットゲートウェイを参照していることを確認します。
3. ネットワーク ACL がトラフィックのフローを許可していることを確認します。
4. セキュリティグループがトラフィックのフローを許可していることを確認します。
5. プライベートサブネットのルートテーブルが仮想バーチャルゲートウェイを参照していることを確認します。
6. CodePipeline のサービスロールに、適切なアクセス許可があることを確認します。例えば、CodePipeline の操作に必要な Amazon EC2 アクセス許可が Amazon VPC にない場合は、「Unexpected EC2 error: UnauthorizedOperation.」というエラーメッセージが表示される場合があります。

ステージとアクションを使用して CI/CD パイプラインを定義する

で自動リリースプロセスを定義するには AWS CodePipeline、パイプラインを作成します。パイプラインは、ソフトウェアの変更がリリースプロセスをどのように通過するかを説明するワークフロー構造です。パイプラインは、設定するステージおよびアクションで構成されます。

> **Note**
>
> CodePipeline が提供するデフォルトオプションに加えて、ビルド、デプロイ、テストまたは呼び出しステージを追加する場合、パイプラインと使用するためにすでに作成したカスタムアクションを選択できます。カスタムアクションは、社内で開発したビルドプロセスやテストスイートを実行する等のタスクに使用できます。プロバイダーリストのカスタムアクションの異なるバージョンを区別できるよう、バージョン識別子が含まれています。詳細については、「CodePipeline でカスタムアクションを作成および追加する」を参照してください。

パイプラインを作成する前に、まずはCodePipeline の使用開始のステップを完了する必要があります。

パイプラインの詳細については、CodePipeline の概念 「」、CodePipeline チュートリアル「」、および「」を参照してください。 AWS CLI を使用してパイプラインを作成する場合は、「」を参照してくださいパイプライン、ステージ、アクションを作成する。パイプラインのリストを表示するには、CodePipeline でパイプラインと詳細を表示する を参照してください。

トピック

- パイプライン、ステージ、アクションを作成する
- CodePipeline でパイプラインを編集する
- CodePipeline でパイプラインと詳細を表示する
- CodePipeline でパイプラインを作成します。
- 別の AWS アカウントのリソースを使用する CodePipeline でパイプラインを作成する
- ポーリングパイプラインをイベントベースの変更検出の使用に移行する
- CodePipeline サービスロールを作成する
- リソースのタグ付け

- CodePipeline でパイプラインにタグ付けする
- 通知ルールの作成

パイプライン、ステージ、アクションを作成する

AWS CodePipeline コンソールまたは AWS CLI を使用してパイプラインを作成できます。パイプラインには少なくとも 2 つのステージが必要です。パイプラインの第 1 ステージは、ソースステージである必要があります。パイプラインには、ビルドまたはデプロイステージである他のステージが少なくとも 1 つ必要です。

> **Important**
>
> パイプライン作成の一環として、CodePipeline は、ユーザーが指定した S3 アーティファクトバケットをアーティファクトとして使用します (これは S3 ソースアクションで使用するバケットとは異なります)。S3 アーティファクトバケットがパイプラインのアカウントとは異なるアカウントにある場合は、S3 アーティファクトバケットが によって所有 AWS アカウント されており、安全で信頼できることを確認してください。

パイプライン AWS リージョン とは異なる にあるアクションをパイプラインに追加できます。クロスリージョンアクションは、 AWS のサービス がアクションのプロバイダーであり、アクションタイプまたはプロバイダータイプがパイプラインとは異なる AWS リージョンにあるアクションです。詳細については、「CodePipeline でクロスリージョンアクションを追加する」を参照してください。

Amazon ECS をデプロイプロバイダとして使用して、コンテナベースのアプリケーションを構築およびデプロイするパイプラインを作成することもできます。Amazon ECS でコンテナベースのアプリケーションをデプロイするパイプラインを作成する前に、イメージ定義ファイルのリファレンスの説明に従ってイメージ定義ファイルを作成する必要があります。

CodePipeline は、ソースコードの変更がプッシュされたときにパイプラインを開始するように、変更検出方法を使用してします。この検出方法はソースタイプに基づいています。

- CodePipeline は Amazon CloudWatch Events を使用して、CodeCommit ソースリポジトリとブランチの変更や S3 ソースバケットの変更を検出します。

> **Note**
>
> コンソールを使用してパイプラインを作成または編集すると、変更検出リソースが作成されます。 AWS CLI を使用してパイプラインを作成する場合は、追加のリソースを自分で作成する必要があります。詳細については、「CodeCommit ソースアクションと EventBridge」を参照してください。

トピック

- カスタムパイプラインを作成する (コンソール)
- パイプラインを作成する (CLI)
- 静的テンプレートからパイプラインを作成する

カスタムパイプラインを作成する (コンソール)

コンソールでカスタムパイプラインを作成するには、アクションで使用するソースファイルの場所とプロバイダーに関する情報を指定する必要があります。

コンソールを使用してパイプラインを作成する場合、ソースステージに加えて、以下のいずれかまたは両方が必要です。

- ビルドステージ
- デプロイステージ

パイプラインウィザードを使用する場合、CodePipeline はステージの名前 (ソース、ビルド、ステージング) を作成します。これらの名前は変更できません。ステージの後半で、より詳細な名前 (たとえば、BuildToGamma または DeployToProd) を使用することもできます。

ステップ 1: パイプラインの作成と名前付け

1. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [Welcome (ようこそ)] ページで、[Create pipeline (パイプラインの作成)] を選択します。

   CodePipeline を初めて使用する場合は、Get Started を選択します。

3. [ステップ 1: 作成オプションを選択する] ページの [作成オプション] で、[カスタムパイプラインを構築する] オプションを選択します。[次へ] を選択します。
4. [ステップ 2: パイプラインの設定を選択する] ページで、[パイプライン名] にパイプラインの名前を入力します。

   1 つの AWS アカウントで、 AWS リージョンで作成する各パイプラインには一意の名前が必要です。名前は、異なるリージョンのパイプラインに再利用できます。

   > **Note**
   >
   > パイプラインを作成したら、その名前を変更することはできません。その他の制限についての詳細については、「AWS CodePipeline のクォータ」を参照してください。

5. [パイプラインのタイプ] で、次のいずれかのオプションを選択します。パイプラインのタイプによって特徴および価格が異なります。詳細については、「パイプラインのタイプ」を参照してください。

   - V1 タイプのパイプラインは、標準のパイプライン、ステージ、アクションレベルのパラメータを含む JSON 構造になっています。
   - V2 タイプのパイプラインは、V1 タイプと同じ構造で、Git タグやパイプラインレベルの変数に対するトリガーなど、追加のパラメータがサポートされています。

6. [Service role (サービスロール)] で、次のいずれかの操作を行います。

   - New service role を選択して、CodePipeline に IAM での新しいサービスロールの作成を許可します。
   - IAM で作成済みのサービスロールを使用するには、[Existing service role (既存のサービスロール)] を選択します。[ロール ARN] で、リストからサービスロール ARN を選択します。

   > **Note**
   >
   > サービスロールが作成された日時によっては、追加の をサポートするためにアクセス許可を更新する必要がある場合があります AWS のサービス。詳細については、「CodePipeline サービスロールにアクセス許可を追加する」を参照してください。

   サービスロールとそのポリシーステートメントの詳細については、「CodePipeline サービスロールを管理する」を参照してください。

7. (オプション) [変数] で [変数を追加] を選択し、パイプラインレベルの変数を追加します。

   パイプラインレベルの変数の詳細については、「変数リファレンス」を参照してください。パイプラインの実行時に渡されるパイプラインレベルの変数のチュートリアルについては、「チュートリアル: パイプラインレベルの変数を使用する」を参照してください。

   > **Note**
   >
   > パイプラインレベルの変数の追加はオプションですが、パイプラインレベルの変数に値が設定されていない場合、パイプラインの実行は失敗します。

8. (オプション) [詳細設定] を展開します。
9. [アーティファクトストア] で、以下のいずれかの操作を行います。

   a. デフォルトの場所を選択して、パイプラインに AWS リージョン 選択した のパイプラインに、デフォルトとして指定された S3 アーティファクトバケットなどのデフォルトのアーティファクトストアを使用します。

   b. S3 アーティファクトバケットなどのアーティファクトストアがパイプラインと同じリージョンに既に存在する場合は、[Custom location (カスタムの場所)] を選択します。[バケット] で、バケット名を選択します。

   > **Note**
   >
   > これはソースコードのソースバケットではありません。パイプラインのアーティファクトストアです。パイプラインごとに S3 バケットなどの個別のアーティファクトストアが必要です。パイプラインを作成または編集するときは、パイプラインリージョンにアーティファクトバケットと、アクションを実行している AWS リージョンごとに 1 つのアーティファクトバケットが必要です。
   > 詳細については、「入力および出力アーティファクト」および「CodePipeline パイプライン構造リファレンス」を参照してください。

10. [暗号化キー] で、次のいずれかの操作を行います。

    a. CodePipeline のデフォルトを使用してパイプラインアーティファクトストア (S3 バケット) 内のデータを暗号化 AWS KMS key するには、デフォルトの AWS マネージドキーを選択します。

b. カスタマーマネージドキーを使用してパイプラインアーティファクトストア (S3 バケット) 内のデータを暗号化するには、[カスタマーマネージドキー] を選択します。 キー ID、キー ARN、またはエイリアスの ARN を選択します。

11. [次へ] を選択します。

ステップ 2: ソースステージを作成する

1. [ステップ 3: ソースステージの追加] ページの [ソースプロバイダー] で、ソースコードが保存されているリポジトリのタイプを選択し、必要なオプションを指定します。選択したソースプロバイダーに応じて、次のような追加のフィールドが表示されます。

   - Bitbucket Cloud、GitHub (GitHub アプリ経由)、GitHub Enterprise Server、GitLab.com、または GitLab セルフマネージドの場合:

     1. 接続 で、既存の接続を選択するか、新規の接続を作成します。GitHub ソースアクション用の接続を作成または管理するには、「GitHub コネクション」を参照してください。
     2. パイプラインのソース場所として使用するリポジトリを選択します。

        トリガーを追加するか、トリガータイプをフィルタリングするかを選択し、パイプラインを開始します。トリガーの操作の詳細については、「コードプッシュまたはプルリクエストイベントタイプを使用してトリガーを追加する」を参照してください。glob パターンを使用したフィルタ処理の詳細については、「構文での glob パターンの使用」を参照してください。
     3. Output artifact format で、アーティファクトのフォーマットを選択します。
        - デフォルトのメソッドを使用して GitHub アクションからの出力アーティファクトを保存するには、CodePipeline default を選択します。アクションは、Bitbucket リポジトリからファイルにアクセスし、パイプラインアーティファクトストアの ZIP ファイルにアーティファクトを保存します。
        - リポジトリへの URL 参照を含む JSON ファイルを保存して、ダウンストリームのアクションで Git コマンドを直接実行できるようにするには、[Full clone (フルクローン)] を選択します。このオプションは、CodeBuild ダウンストリームアクションでのみ使用できます。

          このオプションを選択した場合は、CodePipeline のトラブルシューティング で示されるように CodeBuild プロジェクトサービスロールの権限を更新する必要があります。Full clone オプションを使用する方法を示すチュートリアルについては、チュートリアル: CodeCommit パイプラインソースで完全なクローンを使用する を参照してください。

- Amazon S3 については：

  1. [Amazon S3 の場所] で、S3 バケットの名前と、バージョニングが有効になっているバケット内のオブジェクトへのパスを指定します。バケット名とパスの形式は以下のようになります。

     ```
     s3://bucketName/folderName/objectName
     ```

     > **Note**
     >
     > Amazon S3 がパイプラインのソースプロバイダーである場合、ソースファイルあるいはファイルを 1 つの [.zip] に圧縮し、その [.zip] をソースバケットにアップロードすることができます。解凍されたファイルを 1 つアップロードすることもできます。ただし、.zip ファイルを想定するダウンストリームアクションは失敗します。

  2. S3 ソースバケットを選択すると、CodePipeline は Amazon CloudWatch Events ルールと、このパイプライン用に作成される AWS CloudTrail 証跡を作成します。[Change detection options (変更検出オプション)] で、デフォルト値を受け入れます。これにより、CodePipeline は Amazon CloudWatch Events と AWS CloudTrail を使用して、新しいパイプラインの変更を検出できます。[次へ] を選択します。

- AWS CodeCommit の場合:

  - Repository name で、パイプラインのソース場所として使用する CodeCommitリポジトリの名前を選択します。[Branch name] で、ドロップダウンリストから使用するブランチを選択します。
  - Output artifact format で、アーティファクトのフォーマットを選択します。
    - デフォルトのメソッドを使用して CodeCommit アクションからの出力アーティファクトを保存するには、CodePipeline default を選択します。アクションは、CodeCommit リポジトリからファイルにアクセスし、パイプラインアーティファクトストアの ZIP ファイル中にアーティファクトを保存します。
    - リポジトリへの URL 参照を含む JSON ファイルを保存して、ダウンストリームのアクションで Git コマンドを直接実行できるようにするには、[Full clone (フルクローン)] を選択します。このオプションは、CodeBuild ダウンストリームアクションでのみ使用できます。

      このオプションを選択する場合、`codecommit:GitPull` に示すように、CodeBuild サービスロールに CodeBuild GitClone のアクセス権限を CodeCommit ソースアクションに追

加します。 の許可を追加する必要があります。また、codecommit:GetRepository に示すように、CodePipeline のサービス・ロールに CodePipeline サービスロールにアクセス許可を追加する の許可を追加する必要もあります。Full clone オプションを使用する方法を示すチュートリアルについては、チュートリアル: CodeCommit パイプラインソースで完全なクローンを使用する を参照してください。

- CodeCommit リポジトリ名とブランチを選択すると、Change detection options にメッセージが表示されて、このパイプライン用に作成される Amazon CloudWatch Events ルールが示されます。[Change detection options (変更検出オプション)] で、デフォルト値を受け入れます。これにより、CodePipeline は、 Amazon CloudWatch Events を使用して、新しいパイプラインの変更を検出できます。

- Amazon ECR については :
  - Repository name で、Amazon ECR リポジトリの名前を選択します。
  - [Image tag] で、イメージの名前とバージョンを指定します (最新でない場合)。
  - [出力アーティファクト] で、デフォルトの出力アーティファクト (例: MyApp) を選択します。これには、次のステージで使用するイメージの名前およびリポジトリの URI が含まれます。

    Amazon ECR ソースステージを含む、CodeDeploy ブルー - グリーンデプロイを用いての Amazon ECS のパイプラインの作成に関するチュートリアルについては、チュートリアル: Amazon ECR ソース、ECS - CodeDeploy 間のデプロイでパイプラインを作成する を参照してください。

パイプラインに Amazon ECR のソースステージを含めると、変更をコミットしたときにソースアクションによって、出力アーティファクトとして imageDetail.json のファイルが生成されます。imageDetail.json ファイルの詳細については、「Amazon ECS Blue/Green デプロイアクション用の imageDetail.json ファイル」を参照してください。

> **Note**
>
> オブジェクトとファイルのタイプは、使用するデプロイシステム (Elastic Beanstalk や CodeDeploy など) と互換性があることが必要です。サポートされているファイルのタイプは .zip、.tar、.tgz ファイルなどです。Elastic Beanstalk でサポートされているコンテナのタイプの詳細については、Customizing and Configuring Elastic Beanstalk Environments と Supported Platforms を参照してください。CodeDeploy によるデプロ

イのリビジョンの詳細については、Uploading Your Application Revision と Prepare a Revision を参照してください。

2. 自動再試行のステージを設定するには、[ステージ障害時の自動再試行を有効にする] を選択します。自動再試行の詳細については、「ステージ障害時の自動再試行を設定する」を参照してください。
3. [次へ] を選択します。

ステップ 4: ビルドステージを作成する

デプロイステージを作成する予定の場合、このステップはオプションです。

1. [ステップ 4: ビルドステージの追加] ページで、以下のいずれかの操作を行い、[次へ] を選択します。

   - テストまたはデプロイステージを作成する予定の場合は、[ビルドステージをスキップ] を選択します。
   - ビルドステージのコマンドアクションを選択するには、[コマンド] を選択します。

     > **Note**
     >
     > コマンドアクションを実行すると、 AWS CodeBuildで別途料金が発生します。CodeBuild アクションの一部としてビルドコマンドを挿入する場合は、[その他のビルドプロバイダー] を選択し、[CodeBuild] を選択します。

     [コマンド] で、アクションのシェルコマンドを入力します。コマンドアクションの詳細については、「コマンドアクションリファレンス」を参照してください。
   - CodeBuild などの他のビルドプロバイダーを選択するには、[その他のプロバイダー] を選択します。[ビルドプロバイダー] から、ビルドサービスのカスタムアクションプロバイダーを選択し、そのプロバイダーの設定詳細を入力します。Jenkins をビルドプロバイダとして追加する方法の例については、「チュートリアル: 4 ステージのパイプラインを作成する」を参照してください。
   - [ビルドプロバイダ] から、[AWS CodeBuild] を選択します。

     リージョンで、リソースが存在する AWS リージョンを選択します。リージョンフィールドは、このアクションタイプとプロバイダータイプに対して AWS リソースが作成される場所を

指定します。このフィールドは、アクションプロバイダーが AWS のサービスである場合にのみ表示されます。リージョンフィールドは、デフォルトでパイプラインと同じ AWS リージョンになります。

[プロジェクト名] で、ビルドプロジェクトを選択します。CodeBuild にビルドプロジェクトがすでにある場合は、それを選択します。または、CodeBuild でビルドプロジェクトを作成し、その後でこのタスクに戻ります。CodeBuild User Guide 中の Create a Pipeline That Uses CodeBuild の指示に従います。

[ビルド仕様] の CodeBuild buildspec ファイルはオプションで、代わりにコマンドを入力できます。[ビルドコマンドの挿入] で、アクションのシェルコマンドを入力します。ビルドコマンドの使用に関する考慮事項の詳細については、「コマンドアクションリファレンス」を参照してください。他のフェーズでコマンドを実行する場合、またはコマンドのリストが長い場合は、[buildspec ファイルを使用する] を選択します。

Environment variables で、ビルドアクションに CodeBuild 環境変数を追加するには、Add environment variable を選択します。各変数は、次の 3 つのエントリで構成されます。

- [名前] には、環境変数の名前またはキーを入力します。
- [値] には、環境変数の値を入力します。変数タイプのパラメータを選択した場合は、この値が AWS Systems Manager パラメータストアに既に保存されているパラメータの名前であることを確認します。

> **Note**
>
> 機密情報、特に AWS 認証情報を保存するために環境変数を使用することは強くお勧めしません。CodeBuild コンソールまたは CLI AWS を使用すると、環境変数がプレーンテキストで表示されます。機密の値の場合は、代わりに [パラメータ] 型を使用することをお勧めします。

- (オプション) [型] に、環境変数の型を入力します。有効な値は、[プレーンテキスト] または [パラメータ] です。デフォルトは [プレーンテキスト] です。

(オプション) Build type で、次のいずれかを選択します。

- 各ビルドを 1 回のビルドアクション実行で実行するには、Single build を選択します。
- 同じビルドアクションの実行で複数のビルドを実行するには、Batch build を選択します。

(オプション) バッチビルドを選択した場合、Combine all artifacts from batch into a single location を選択して、すべてのビルドアーティファクトを 1 つの出力アーティファクトに配置します。

2. 自動再試行のステージを設定するには、[ステージ障害時の自動再試行を有効にする] を選択します。自動再試行の詳細については、「ステージ障害時の自動再試行を設定する」を参照してください。
3. [次へ] を選択します。

ステップ 5: テストステージを作成する

ビルドまたはデプロイステージを作成する予定の場合、このステップはオプションです。

1. [ステップ 5: テストステージの追加] ページで、以下のいずれかの操作を行い、[次へ] を選択します。
   - ビルドまたはデプロイステージを作成する予定の場合は、[テストステージをスキップ] を選択します。
   - [テストプロバイダー] でテストアクションプロバイダーを選択し、適切なフィールドに入力します。
2. [次へ] を選択します。

ステップ 6: デプロイステージを作成する

ビルドステージをすでに作成している場合、このステップはオプションです。

1. [ステップ 6: デプロイステージの追加] ページで、以下のいずれかの操作を行い、[次へ] を選択します。
   - 前のステップでビルドまたはテストステージを作成した場合は、[導入段階をスキップ] を選択します。

> **Note**
>
> すでにビルドまたはテストステージをスキップしている場合、このオプションは表示されません。

- [デプロイプロバイダ] で、デプロイプロバイダ用に作成したカスタムアクションを選択します。

  リージョンでは、クロスリージョンアクションに対してのみ、リソースが作成される AWS リージョンを選択します。[リージョン] フィールドは、このアクションタイプとプロバイダータイプに対して作成済みの AWS リソースの場所を示します。このフィールドには、アクションプロバイダーが AWS のサービスであるアクションのみが表示されます。リージョンフィールドは、デフォルトでパイプラインと同じ AWS リージョンになります。

- [デプロイプロバイダ] で、デフォルトプロバイダ用の以下のフィールドを使用できます。

  - CodeDeploy

    Application name で、既存の CodeDeploy アプリケーションの名前を入力または選択します。[デプロイグループ] に、アプリケーションのデプロイグループの名前を入力します。[次へ] を選択します。アプリケーション、デプロイグループ、または両方を CodeDeploy コンソールで作成することもできます。

  - AWS Elastic Beanstalk

    Application name で、既存の Elastic Beanstalk アプリケーションの名前を入力または選択します。[環境名] に、アプリケーションの環境を入力します。[次へ] を選択します。アプリケーション、環境、または両方を Elastic Beanstalk コンソールで作成することもできます。

  - AWS OpsWorks Stacks

    [スタック] で、使用するスタックの名前を入力または選択します。[レイヤー] で、ターゲットインスタンスがあるレイヤーを選択します。[デプロイ] で、更新およびデプロイするアプリケーションを選択します。アプリケーションを作成する必要がある場合は、[Create a new one in AWS OpsWorks] を選択します。

    スタックとレイヤーにアプリケーションを追加する方法については AWS OpsWorks、「AWS OpsWorks ユーザーガイド」の「アプリケーションの追加」を参照してください。

    CodePipeline でシンプルなパイプラインをレイヤーで実行するコードのソースとして使用する方法のend-to-endの例については、CodePipeline の使用 AWS OpsWorks Stacks」を参照してください。 AWS OpsWorks

  - AWS CloudFormation

    次のいずれかを行います:

- アクションモードで、スタックの作成または更新を選択し、スタック名とテンプレートファイル名を入力し、 が引き受け AWS CloudFormation るロールの名前を選択します。必要に応じて、設定ファイルの名前を入力し、IAM 機能オプションを選択します。
- アクションモードで、変更セットを作成または置き換え、スタック名と変更セット名を入力し、 が引き受け AWS CloudFormation るロールの名前を選択します。必要に応じて、設定ファイルの名前を入力し、IAM 機能オプションを選択します。

CodePipeline で AWS CloudFormation 機能をパイプラインに統合する方法については、AWS CloudFormation 「 ユーザーガイド」のCodePipeline を使用した継続的デリバリー」を参照してください。

- Amazon ECS

Cluster name で、既存の Amazon ECS クラスターの名前を入力または選択します。[サービス名] で、クラスターで実行されているサービスの名前を入力または選択します。クラスターとサービスを作成することもできます。[イメージのファイル名] で、サービスのコンテナとイメージを説明するイメージ定義ファイルの名前を入力します。

> **Note**
>
> Amazon ECS デプロイアクションでは、デプロイアクションへの入力として `imagedefinitions.json` のファイルが必要です。ファイルのデフォルトのファイル名は imagedefinitions.json です。別のファイル名を使用することを選択した場合は、パイプラインデプロイステージを作成するときにそのファイル名を指定する必要があります。詳細については、「Amazon ECS 標準デプロイアクション用の imagedefinitions.json ファイル」を参照してください。

[Next] (次へ) を選択します。

> **Note**
>
> Amazon ECS クラスターが 2 つ以上のインスタンスで設定されていることを確認してください。Amazon ECS クラスターには、少なくとも 2 つのインスタンスが必要です。1 つはプライマリインスタンスとして維持し、もう 1 つは新しいデプロイに対応するために使用します。

パイプラインを使用したコンテナベースのアプリケーションのデプロイに関するチュートリアルについては、Tutorial: Continuous Deployment with CodePipeline を参照してください。

- Amazon ECS (Blue/Green)

  CodeDeploy アプリケーションとデプロイグループ、 Amazon ECS タスク定義、AppSpec ファイル情報を入力して、Next を選択します。

  > **Note**
  >
  > Amazon ECS (Blue/Green) アクションには、デプロイアクションの入力アーティファクトとして imageDetail.json ファイルが必要です。Amazon ECR ソースアクションがこのファイルを作成するので、Amazon ECR ソースアクションを持つパイプラインは、`imageDetail.json` のファイルを提供する必要はありません。詳細については、「Amazon ECS Blue/Green デプロイアクション用の imageDetail.json ファイル」を参照してください。

  CodeDeploy を使用して Amazon ECS クラスターへの blue-green デプロイ用のパイプラインを作成するためのチュートリアルについては、チュートリアル: Amazon ECR ソース、ECS - CodeDeploy 間のデプロイでパイプラインを作成する を参照してください。

- AWS Service Catalog

  コンソールのフィールドを使用して設定を指定する場合は [Enter deployment configuration (デプロイ設定の入力)] を選択します。あるいは、個別の設定ファイルがある場合は [設定ファイル] を選択します。製品と設定の情報を入力し、[次へ] を選択します。

  パイプラインを使用して製品の変更を Service Catalog にデプロイする方法のチュートリアルについては、「チュートリアル: Service Catalog にデプロイするパイプラインを作成する」を参照してください。

- Alexa Skills Kit

  [Alexa Skill ID (Alexa スキル ID)] に Alexa スキルのスキル ID を入力します。[クライアント ID] と [クライアントシークレット] に、Login with Amazon (LWA) セキュリティプロファイルを使用して生成された認証情報を入力します。[Refresh token (更新トークン)] に、ASK CLI コマンドを使用して生成した更新トークンを入力します。[次へ] を選択します。

パイプラインにより Alexa スキルをデプロイする方法、LWA 認証情報を生成する方法のチュートリアルについては、「チュートリアル: Amazon Alexa Skill をデプロイするパイプラインを作成する」を参照してください。

- Amazon S3

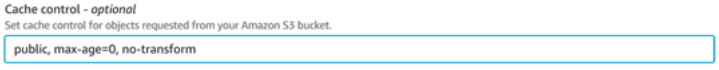
  [バケット] に、使用する S3 バケットの名前を入力します。デプロイステージへの入力アーティファクトが ZIP ファイルの場合は、[Extract file before deploy (デプロイ前にファイルを展開)] を選択します。[Extract file before deploy (デプロイ前にファイルを展開)] を選択した場合は、オプションで [Deployment path (デプロイパス)] に ZIP ファイルの解凍先を入力できます。選択されていない場合は、S3 オブジェクトキーに値を入力する必要があります。

  > **Note**
  >
  > ほとんどのソースステージおよびビルドステージの出力アーティファクトは圧縮されます。Amazon S3 zip を除くすべてのパイプラインソースプロバイダーは、ソースファイルを Zip してから、次のアクションに入力アーティファクトとして提供します。

  (オプション) Canned ACL で、canned ACL を入力し、Amazon S3 にデプロイされたオブジェクトに適用します。

  > **Note**
  >
  > 既定 ACL を適用すると、オブジェクトに適用された既存の ACL が上書きされます。

  (オプション) [キャッシュコントロール] で、バケットからオブジェクトをダウンロードするリクエストのキャッシュコントロールパラメータを指定します。有効な値のリストについては、HTTP オペレーションの `Cache-Control` ヘッダーフィールドを参照してください。[Cache control (キャッシュコントロール)] に複数の値を入力するには、各値の間にカンマを使用します。この例に示すように、各カンマの後にスペースを追加できます (オプション)。



上記のエントリ例は、CLI に次のように表示されます。

```
"CacheControl": "public, max-age=0, no-transform"
```

[次へ] を選択します。

Amazon S3 デプロイアクションプロバイダとして を使用するパイプラインを作成する方法のチュートリアルについては、チュートリアル: Amazon S3 をデプロイプロバイダとして使用するパイプラインを作成する を参照してください。

2. 自動再試行のステージを設定するには、[ステージ障害時の自動再試行を有効にする] を選択します。自動再試行の詳細については、「ステージ障害時の自動再試行を設定する」を参照してください。
3. 自動ロールバックのステージを設定するには、[ステージ障害時の自動ロールバックを設定] を選択します。自動ロールバックの詳細については、「ステージの自動ロールバックを設定する」を参照してください。
4. [次のステップ] を選択します。

ステップ 7: パイプラインを確認する

- ステップ 7: 確認ページでパイプライン設定を確認し、パイプラインの作成を選択してパイプラインを作成するか、以前の を選択して選択内容に戻って編集します。パイプラインを作成せずにウィザードを終了するには、[Cancel] を選択します。

パイプラインが作成され、コンソールで表示できるようになりました。パイプラインは、作成後に実行されます。詳細については、「CodePipeline でパイプラインと詳細を表示する」を参照してください。パイプラインを変更する方法の詳細については、「CodePipeline でパイプラインを編集する」を参照してください。

パイプラインを作成する (CLI)

を使用してパイプライン AWS CLI を作成するには、JSON ファイルを作成してパイプライン構造を定義し、 `--cli-input-json`パラメータを指定して create-pipeline コマンドを実行します。

> **Important**
>
> を使用して AWS CLI 、パートナーアクションを含むパイプラインを作成することはできません。代わりに CodePipeline コンソールを使用する必要があります。

パイプライン構造に関する詳細については、CodePipeline パイプライン構造リファレンス および CodePipeline API Reference の create-pipeline を参照してください。

JSON ファイルを作成するには、同じパイプラインの JSON ファイルを使用して編集し、create-pipeline コマンドを実行するときにこのファイルを呼び出します。

前提条件:

CodePipeline の使用開始 で CodePipeline 用に作成したサービスロールの ARN が必要です。create-pipeline のコマンドの実行時、パイプライン JSON ファイル中の CodePipeline サービスロールの ARN を使用します。サービスロールの作成の詳細については、「CodePipeline サービスロールを作成する」を参照してください。コンソールとは異なり、 で create-pipeline コマンドを実行すると、CodePipeline サービスロールを作成するオプション AWS CLI はありません。サービスロールがすでに存在している必要があります。

パイプラインのアーティファクトの保存先である S3 バケットの名前が必要です。このバケットはパイプラインと同じリージョンに存在する必要があります。バケット名は、create-pipeline コマンドの実行時にパイプライン JSON ファイルで使用します。コンソールとは異なり、 で create-pipeline コマンドを実行して AWS CLI も、アーティファクトを保存するための S3 バケットは作成されません。バケットが存在している必要があります。

> **Note**
>
> get-pipeline コマンドを使用して、パイプラインの JSON 構造のコピーを取得し、プレーンテキストエディタで構造を変更することもできます。

JSON ファイルを作成するには

1. ターミナル (Linux、 macOS、あるいは Unix)またはコマンドプロンプト(Windows)で、ローカルディレクトリに新規のテキストファイルを作成します。

2. (オプション) パイプラインレベルでは 1 つ以上の変数を追加できます。この値は CodePipeline アクションの設定で参照できます。パイプラインを作成するときに変数名と値を追加できます。また、コンソールでパイプラインを開始するときに値を割り当てることもできます。

> **Note**
>
> パイプラインレベルの変数の追加はオプションですが、パイプラインレベルの変数に値が設定されていない場合、パイプラインの実行は失敗します。

パイプラインレベルの変数は、パイプラインの実行時に解決されます。すべての変数は不変であり、値が割り当てられた後は更新できません。パイプラインレベルの変数のうち値が解決されたものは、実行ごとの履歴に表示されます。

パイプライン構造の変数属性を使用して、パイプラインレベルの変数を指定します。以下の例では、変数 `Variable1` の値は `Value1` です。

```
    "variables": [
         {
             "name": "Timeout",
             "defaultValue": "1000",
             "description": "description"
         }
     ]
```

この構造をパイプラインの JSON に追加するか、次のステップのサンプルの JSON に追加します。名前空間の情報など変数の詳細については、「変数リファレンス」を参照してください。

3. プレーンテキストエディタでファイルを開き、作成する構造を反映するように値を編集します。少なくとも、パイプラインの名前を変更する必要があります。また、変更するかを考慮する必要もあります。

- このパイプラインのアーティファクトの保存先の S3 バケット。
- コードのソースの場所。
- デプロイのプロバイダ。
- コードをデプロイする方法。
- パイプラインのタグ。

以下の 2 ステージのサンプルパイプライン構造では、変更を検討する必要があるパイプラインの値を強調表示しています。パイプラインには多くの場合、2 つ以上のステージが含まれます。

```
{
    "pipeline": {
        "roleArn": "arn:aws:iam::80398EXAMPLE::role/AWS-CodePipeline-Service",
        "stages": [
            {
                "name": "Source",
                "actions": [
                    {
                        "inputArtifacts": [],
                        "name": "Source",
                        "actionTypeId": {
                            "category": "Source",
                            "owner": "AWS",
                            "version": "1",
                            "provider": "S3"
                        },
                        "outputArtifacts": [
                            {
                                "name": "MyApp"
                            }
                        ],
                        "configuration": {
                            "S3Bucket": "amzn-s3-demo-source-bucket",
                            "S3ObjectKey": "ExampleCodePipelineSampleBundle.zip",
                            "PollForSourceChanges": "false"
                        },
                        "runOrder": 1
                    }
                ]
            },
            {
                "name": "Staging",
                "actions": [
                    {
                        "inputArtifacts": [
                            {
                                "name": "MyApp"
                            }
                        ],
```

```
                    "name": "Deploy-CodeDeploy-Application",
                    "actionTypeId": {
                        "category": "Deploy",
                        "owner": "AWS",
                        "version": "1",
                        "provider": "CodeDeploy"
                    },
                    "outputArtifacts": [],
                    "configuration": {
                        "ApplicationName": "CodePipelineDemoApplication",
                        "DeploymentGroupName": "CodePipelineDemoFleet"
                    },
                    "runOrder": 1
                }
            ]
        }
    ],
    "artifactStore": {
        "type": "S3",
        "location": "codepipeline-us-east-2-250656481468"
    },
    "name": "MyFirstPipeline",
    "version": 1,
    "variables": [
        {
            "name": "Timeout",
            "defaultValue": "1000",
            "description": "description"
        }
      ]
    },
    "triggers": [
        {
            "providerType": "CodeStarSourceConnection",
            "gitConfiguration": {
                "sourceActionName": "Source",
                "push": [
                    {
                        "tags": {
                            "includes": [
                                "v1"
                            ],
                            "excludes": [
                                "v2"
```

```
                                ]
                            }
                        }
                    ]
                }
            }
        ]
    "metadata": {
        "pipelineArn": "arn:aws:codepipeline:us-
east-2:80398EXAMPLE:MyFirstPipeline",
        "updated": 1501626591.112,
        "created": 1501626591.112
    },
    "tags": [{
      "key": "Project",
      "value": "ProjectA"
    }]
}
```

この例では、パイプラインの `Project` タグキーと `ProjectA` 値を含めることによって、タグ付けをパイプラインに追加します。CodePipeline のタグ付けリソースのさらなる詳細については、リソースのタグ付け を参照してください。

JSON ファイルの `PollForSourceChanges` パラメータが次のように設定されていることを確認します。

```
                "PollForSourceChanges": "false",
```

CodePipeline は、Amazon CloudWatch Events を使用して、CodeCommit ソースリポジトリとブランチの変更、あるいは S3 ソースバケットの変更を検出します。次のステップには、パイプラインにこれらのリソースを手動で作成する手順が含まれています。フラグを `false` に設定すると、定期的なチェックが無効になります。これは、推奨される変更検出メソッドを使用している場合には、必要ではありません。

4. パイプラインとは異なるリージョンでビルド、テスト、またはデプロイアクションを作成するには、パイプライン構造に以下を追加する必要があります。手順については、「CodePipeline でクロスリージョンアクションを追加する」を参照してください。

   - `Region` パラメータをアクションのパイプライン構造に追加します。
   - `artifactStores` パラメータを使用して、アクションがある各 AWS リージョンのアーティファクトバケットを指定します。

5. その構造で問題がなければ、**pipeline.json** のような名前でファイルを保存します。

パイプラインを作成するには

1. 先ほど作成した JSON ファイルを `--cli-input-json` パラメータで指定して、create-pipeline コマンドを実行します。

   MySecondPipeline という名前を JSON 内の `name` のための値として含む [pipeline.json] という名前の JSON ファイルを使用して *MySecondPipeline* という名前のパイプラインを作成するには、コマンドは以下のように見えます：

   ```
   aws codepipeline create-pipeline --cli-input-json file://pipeline.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。

   このコマンドは、作成したパイプライン全体の構造を返します。
2. パイプラインを表示するには、CodePipeline コンソールを開いてパイプラインのリストから選択するか、get-pipeline-state のコマンドを使用します。詳細については、「CodePipeline でパイプラインと詳細を表示する」を参照してください。
3. パイプラインの作成に CLI を使用する場合には、推奨される変更検出リソースを手動でパイプラインに作成する必要があります。

   - CodeCommit リポジトリを使用するパイプラインのために、CodeCommit ソースに対する EventBridge ルールを作成する (CLI) で述べられている CloudWatch Events ルールを手動で作成する必要があります。
   - Amazon S3 ソースを持つパイプラインの場合、「」で説明されているように、CloudWatch Events ルールと AWS CloudTrail 証跡を手動で作成する必要がありますEventBridge と を使用する Amazon S3 ソースアクションへの接続 AWS CloudTrail。

静的テンプレートからパイプラインを作成する

テンプレートを使用して、指定したソースコードとプロパティでパイプラインを設定するパイプラインをコンソールで作成できます。アクションに使用するソースファイルの場所とソースプロバイダー

に関する情報を指定する必要があります。Amazon ECR または GitHub AWS CodeConnectionsなどの でサポートされているサードパーティーリポジトリのソースアクションを指定できます。

テンプレートは、次のリソースを含むパイプライン CloudFormation の にスタックを作成します。

- パイプラインを V2 パイプラインタイプで作成します。[パイプラインのタイプ] で、次のいずれかのオプションを選択します。パイプラインのタイプによって特徴および価格が異なります。詳細については、「パイプラインのタイプ」を参照してください。
- サービスロールをパイプライン用に作成し、テンプレートで参照します。
- アーティファクトストアを作成します。これを作成するには、パイプライン用に選択した AWS リージョン でパイプラインのデフォルトのアーティファクトストア (デフォルトとして指定した S3 アーティファクトバケットなど) を使用します。

静的テンプレート作成ウィザードに使用されるオープンソースのスターターテンプレートのコレクションを表示するには、https://github.com/aws/codepipeline-starter-templates のリポジトリを参照してください。

静的テンプレートを使用してパイプラインを作成する場合、ユースケースのニーズに応じてテンプレートごとにパイプライン構造が設定されます。例えば、 へのデプロイ用の テンプレート CloudFormation は、この手順の例として使用されます。テンプレートは、以下の構造を持つ DeployToCloudFormationService という名前のパイプラインを生成します。

- ウィザードで指定した設定のソースアクションを含むビルドステージ。
- CloudFormationのデプロイアクションおよび関連リソーススタックを含むデプロイステージ。

静的テンプレートを使用してパイプラインを作成する場合、CodePipeline はステージ (ソース、ビルド、ステージング) の名前を作成します。これらの名前は変更できません。ステージの後半で、より詳細な名前 (たとえば、BuildToGamma または DeployToProd) を使用することもできます。

ステップ 1: コンソールにアクセスする

1. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [Welcome (ようこそ)] ページで、[Create pipeline (パイプラインの作成)] を選択します。

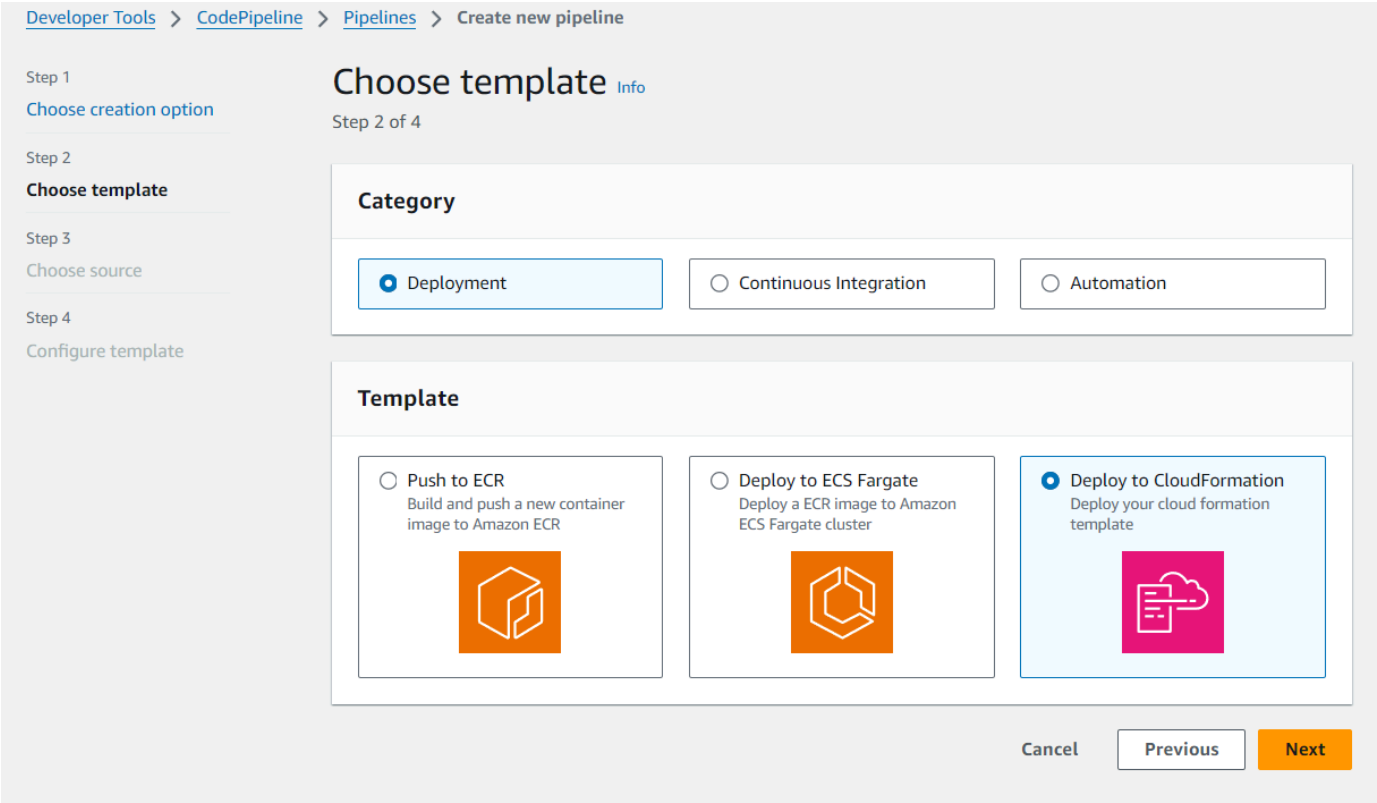
   CodePipeline を初めて使用する場合は、Get Started を選択します。

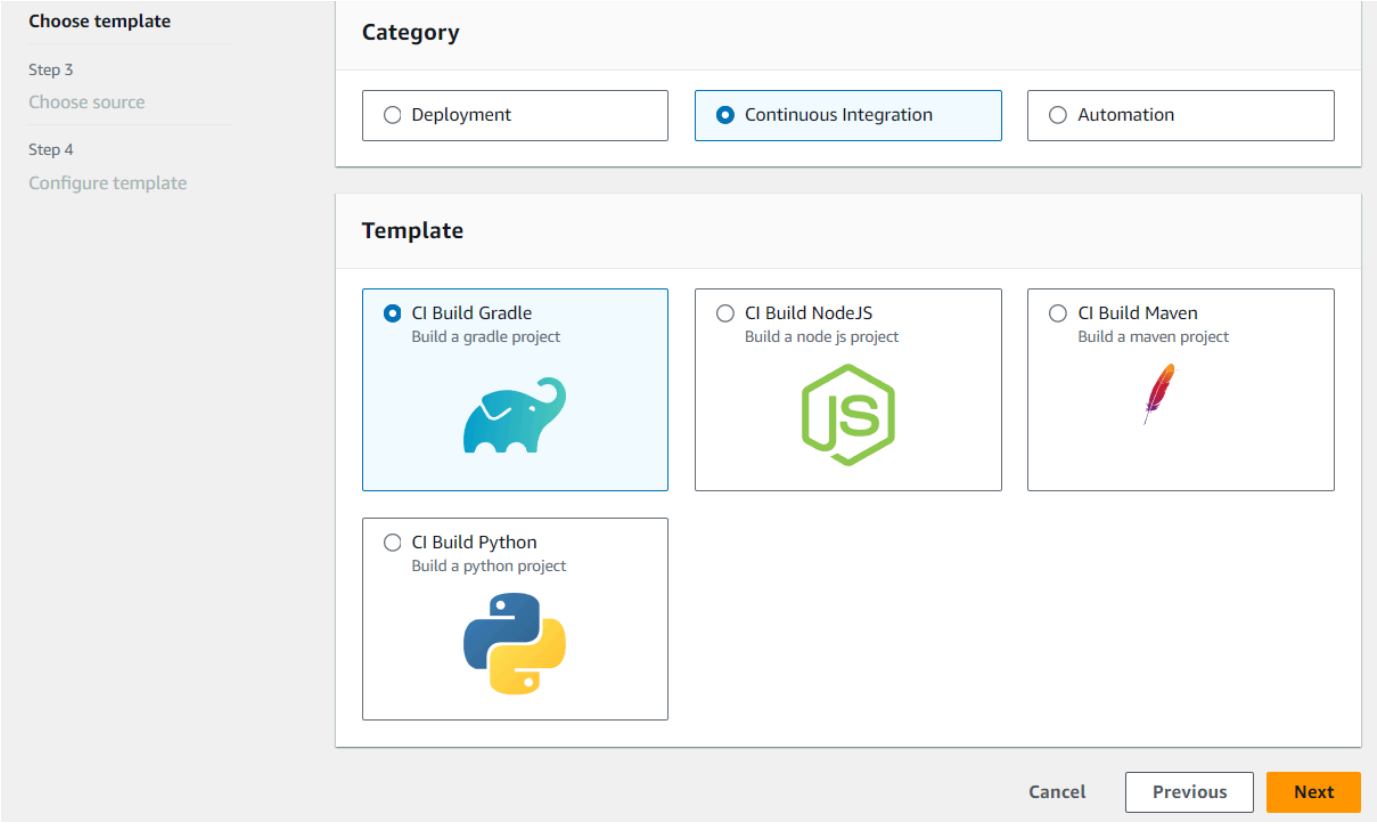
ステップ 2: テンプレートを選択する

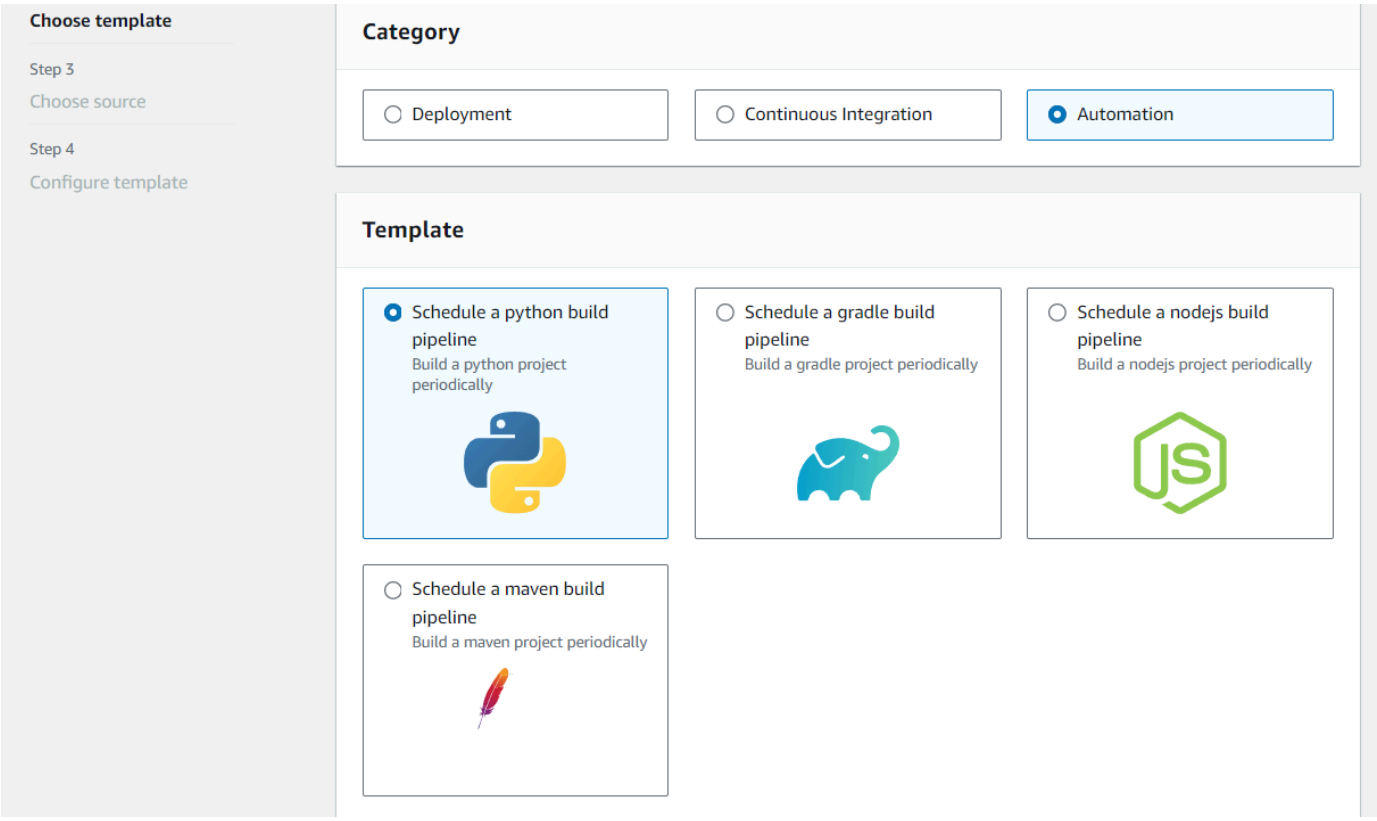
テンプレートを選択し、デプロイステージ、オートメーション、または CI パイプラインを使用してパイプラインを作成します。

1. [ステップ 1: オプションを作成する] ページで、次のいずれかの操作を行い、[次へ] を選択します。

   - デプロイステージを作成する場合は、[デプロイ] を選択します。ECR または CloudFormation にデプロイするテンプレートのオプションを表示します。この例では、[デプロイ] を選択し、CloudFormation にデプロイすることを選択します。
   - CI パイプラインを作成する場合は、[継続的インテグレーション] を選択します。Gradle への構築など、CI パイプラインのオプションを表示します。
   - 自動パイプラインを作成する場合は、[オートメーション] を選択します。Python ビルドのスケジュールなど、オートメーションのオプションを表示します。

2. 


Choose template
Step 3
Choose source
Step 4
Configure template
Category
Deployment
Continuous Integration
Automation
Template
CI Build Gradle
Build a gradle project
CI Build NodeJS
Build a node js project
CI Build Maven
Build a maven project
CI Build Python
Build a python project
Cancel
Previous
Next


Choose template
Step 3
Choose source
Step 4
Configure template
Category
Deployment
Continuous Integration
Automation
Template
Schedule a python build pipeline
Build a python project periodically
Schedule a gradle build pipeline
Build a gradle project periodically
Schedule a nodejs build pipeline
Build a nodejs project periodically
Schedule a maven build pipeline
Build a maven project periodically

ステップ 3: ソースを選択する

- [ステップ 3: ソースを選択する] ページの [ソースプロバイダー] で、ソースコードが保存されているリポジトリのプロバイダーを選択し、必要なオプションを指定して [次のステップ] を選択します。

  - Bitbucket Cloud、GitHub (GitHub アプリ経由)、GitHub Enterprise Server、GitLab.com、または GitLab セルフマネージドの場合:

    1. 接続 で、既存の接続を選択するか、新規の接続を作成します。GitHub ソースアクション用の接続を作成または管理するには、「GitHub コネクション」を参照してください。
    2. パイプラインのソース場所として使用するリポジトリを選択します。

       トリガーを追加するか、トリガータイプをフィルタリングするかを選択し、パイプラインを開始します。トリガーの操作の詳細については、「コードプッシュまたはプルリクエストイベントタイプを使用してトリガーを追加する」を参照してください。glob パターンを使用したフィルタ処理の詳細については、「構文での glob パターンの使用」を参照してください。
    3. Output artifact format で、アーティファクトのフォーマットを選択します。

       - デフォルトのメソッドを使用して GitHub アクションからの出力アーティファクトを保存するには、CodePipeline default を選択します。アクションは、Bitbucket リポジトリからファイルにアクセスし、パイプラインアーティファクトストアの ZIP ファイルにアーティファクトを保存します。
       - リポジトリへの URL 参照を含む JSON ファイルを保存して、ダウンストリームのアクションで Git コマンドを直接実行できるようにするには、[Full clone (フルクローン)] を選択します。このオプションは、CodeBuild ダウンストリームアクションでのみ使用できます。

         このオプションを選択した場合は、CodePipeline のトラブルシューティング で示されるように CodeBuild プロジェクトサービスロールの権限を更新する必要があります。[完全クローン] オプションを使用する方法を示すチュートリアルについては、チュートリアル: CodeCommit パイプラインソースで完全なクローンを使用する を参照してください。

  - Amazon ECR については :

    - Repository name で、Amazon ECR リポジトリの名前を選択します。
    - [Image tag] で、イメージの名前とバージョンを指定します (最新でない場合)。

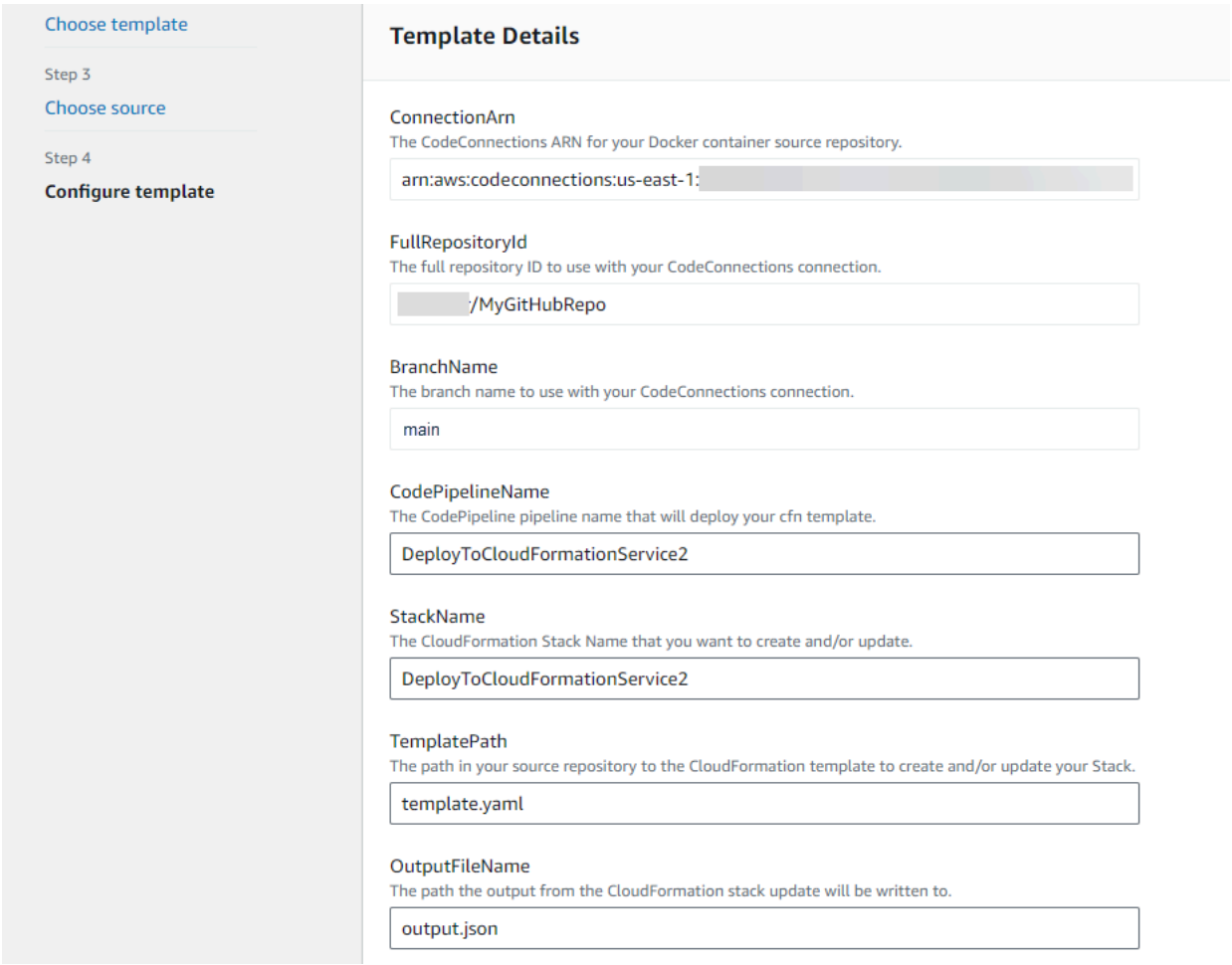
- [出力アーティファクト] で、デフォルトの出力アーティファクト (例: MyApp) を選択します。これには、次のステージで使用するイメージの名前およびリポジトリの URI が含まれます。

パイプラインに Amazon ECR のソースステージを含めると、変更をコミットしたときにソースアクションによって、出力アーティファクトとして `imageDetail.json` のファイルが生成されます。`imageDetail.json` ファイルの詳細については、「Amazon ECS Blue/Green デプロイアクション用の imageDetail.json ファイル」を参照してください。

> **Note**
>
> オブジェクトとファイルのタイプは、使用するデプロイシステム (Elastic Beanstalk や CodeDeploy など) と互換性があることが必要です。サポートされているファイルのタイプは .zip、.tar、.tgz ファイルなどです。Elastic Beanstalk でサポートされているコンテナのタイプの詳細については、Customizing and Configuring Elastic Beanstalk Environments と Supported Platforms を参照してください。CodeDeploy によるデプロイのリビジョンの詳細については、Uploading Your Application Revision と Prepare a Revision を参照してください。

ステップ 4: テンプレートを設定する

この例では、CloudFormation へのデプロイが選択されています。このステップでは、テンプレートの設定を追加します。



1. [ステップ 4: テンプレートを設定する] で、[スタック名] にパイプラインの名前を入力します。
2. テンプレートに適用するアクセス許可のプレースホルダー IAM ポリシーを編集します。
3. [パイプラインをテンプレートから作成する] を選択する
4. パイプラインリソースを作成中であることを示すメッセージが表示されます。

ステップ 5: パイプラインを表示する

- パイプラインを作成したら、CodePipeline でパイプラインを表示し、 CloudFormationでスタックを確認できます。パイプラインは、作成後に実行されます。詳細については、「CodePipeline でパイプラインと詳細を表示する」を参照してください。パイプラインを変更する方法の詳細については、「CodePipeline でパイプラインを編集する」を参照してください。

CodePipeline でパイプラインを編集する

パイプラインは、実行する必要があるステージやアクションなど、AWS CodePipeline が従うリリースプロセスを記述します。パイプラインを編集して、要素を追加または削除することができます。ただし、パイプラインを編集するときには、パイプライン名またはパイプラインメタデータなどの値を変更することはできません。

パイプライン編集ページを使用して、パイプラインタイプ、変数、およびトリガーを編集できます。パイプラインのステージとアクションを追加または変更することもできます。

パイプラインを作成する場合とは異なり、パイプラインを編集しても、パイプラインを通して最新のリビジョンが実行されることはありません。編集後のパイプラインを通して、最新のリビジョンを実行する場合は、手動で再実行する必要があります。それ以外の場合、編集後のパイプラインはソースステージで設定されているソースの場所の次回変更時に実行されます。詳細については、「パイプラインを手動で開始する」を参照してください。

パイプラインとは異なる AWS リージョン内にあるアクションをパイプラインに追加できます。AWS のサービス がアクションプロバイダーであり、このアクションタイプ/プロバイダータイプがパイプラインとは異なる AWS リージョンにある場合、これはクロスリージョンアクションです。クロスリージョンアクションの詳細については、「CodePipeline でクロスリージョンアクションを追加する」を参照してください。

CodePipeline は、ソースコードの変更がプッシュされたときに変更検出方法を使用してパイプラインを開始します。この検出方法はソースタイプに基づいています。

- CodePipeline は、Amazon CloudWatch Events を利用して、CodeCommit ソースリポジトリや Amazon S3 ソースバケットの変更を検出します。

> **Note**
>
> 変更検出リソースは、コンソールを使用するときに自動的に作成されます。コンソールを使用してパイプラインを作成または編集すると、追加のリソースが作成されます。AWS CLI を使用してパイプラインを作成するときは、追加のリソースを各自で作成する必要があります。CodeCommit パイプラインの作成または更新の詳細については、「CodeCommit ソースに対する EventBridge ルールを作成する (CLI)」を参照してください。CLI を使用した Amazon S3 パイプラインの作成または更新の詳細については、「Amazon S3 ソースに対する EventBridge ルールを作成する (CLI)」を参照してください。

トピック

- パイプラインを編集する (コンソール)
- パイプラインを編集する (AWS CLI)

パイプラインを編集する (コンソール)

CodePipeline コンソールを使用して、パイプラインのステージやステージ内のアクションを追加、編集、または削除できます。

パイプラインを更新すると、CodePipeline はすべての実行中のアクションを正常に完了し、実行中のアクションが完了したステージとパイプラインの実行に失敗します。パイプラインが更新されたら、パイプラインを再実行する必要があります。パイプラインの実行に関する詳細については、「パイプラインを手動で開始する」を参照してください。

パイプラインを編集するには

1. AWS マネジメントコンソール にサインインして、CodePipeline コンソール (https://console.aws.amazon.com/codesuite/codebuild/home) を開きます。

   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。
2. [名前] で、編集するパイプラインの名前を選択します。これにより、パイプラインの詳細ビューが開いて、パイプラインの各ステージの各アクションの状態などがわかります。
3. パイプライン詳細ページで、[編集] を選択します。
4. パイプラインタイプを編集するには、[編集: パイプラインのプロパティ] カードで [編集: を選択します。次のいずれかのオプションを選択し、[完了] を選択します。

   - V1 タイプのパイプラインは、標準のパイプライン、ステージ、アクションレベルのパラメータを含む JSON 構造になっています。
   - V2 タイプのパイプラインは、V1 タイプと同じ構造ですが、トリガーとパイプラインレベルの変数など、追加のパラメータがサポートされています。

   パイプラインのタイプによって特徴および価格が異なります。詳細については、「パイプラインのタイプ」を参照してください。
5. パイプライン変数を編集するには、[編集: 変数] カードの[変数の編集] を選択します。パイプラインレベルの変数を追加または変更し、[完了] を選択します。

パイプラインレベルの変数の詳細については、「変数リファレンス」を参照してください。パイプラインの実行時に渡されるパイプラインレベルの変数のチュートリアルについては、「チュートリアル: パイプラインレベルの変数を使用する」を参照してください。

> **Note**
>
> パイプラインレベルの変数の追加はオプションですが、パイプラインレベルの変数に値が設定されていない場合、パイプラインの実行は失敗します。

6. パイプライントリガーを編集するには、[編集: トリガー] カードで [トリガーの編集] を選択します。トリガーを追加または変更し、[完了] を選択します。

   トリガーの追加の詳細については、Bitbucket Cloud、GitHub (GitHub アプリ経由)、GitHub Enterprise Server、GitLab.com、または GitLab セルフマネージド、GitHub コネクション などへの接続を作成する手順を参照してください。

7. [編集] ページでステージとアクションを編集するには、次のいずれかの操作を行います。

   - ステージを編集するには、[ステージを編集] を選択します。アクションは既存のアクションに対して直列にも並列にも追加できます。

     これらのアクションの編集アイコンを選択して、このビューでアクションを編集することもできます。アクションを削除するには、そのアクションの削除アイコンを選択します。

   - アクションを編集するには、そのアクションの編集アイコンを選択し、[アクションの編集] で値を変更します。アスタリスク (*) の付いた項目は必須です。

     - CodeCommit リポジトリ名およびブランチでは、このパイプラインに対して作成される Amazon CloudWatch Events ルールを示すメッセージが表示されます。CodeCommit ソースを削除すると、Amazon CloudWatch Events ルールが削除されることを示すメッセージが表示されます。

     - Amazon S3 ソースバケットに対して、Amazon CloudWatch Events ルールおよび AWS CloudTrail 証跡がこのパイプラインに作成されることを示すメッセージが表示されます。Amazon S3 ソースを削除すると、Amazon CloudWatch Events ルールおよび AWS CloudTrail 証跡が削除されることを示すメッセージが表示されます。AWS CloudTrail 証跡が他のパイプラインで使用されている場合には、この証跡は削除されませんが、データイベントは削除されます。

- ステージを追加するには、ステージを追加するパイプラインのポイントで [+ Add stage (+ ステージを追加)] を選択します。ステージの名前を入力し、少なくとも 1 つのアクションをそのステージに追加します。アスタリスク (*) の付いた項目は必須です。
- ステージを削除するには、そのステージの削除アイコンを選択します。ステージとそのすべてのアクションが削除されます。
- 障害時に自動的にロールバックするようにステージを設定するには、[ステージを編集] を選択し、[ステージ障害時の自動ロールバックを設定] チェックボックスをオンにします。

たとえば、パイプラインのステージにシリアルアクションを追加する場合は、以下のようにします。

1. アクションを追加するステージで、[Edit stage (ステージを編集)] を選択し、[+ Add action group (+ アクショングループの追加)] を選択します。
2. [アクションを編集] の、[アクション名] でアクションの名前を入力します。[Action provider (アクションプロバイダー)] リストには、プロバイダーのオプションがカテゴリ別に表示されます。[デプロイ] などのカテゴリを探します。カテゴリで、プロバイダー (AWS CodeDeploy など) を選択します。[リージョン] で、リソースの作成先または作成予定先の AWS リージョンを選択します。Region フィールドは、このアクションタイプとプロバイダータイプに対して作成済みの AWS のリソースの場所を示します。このフィールドには、アクションプロバイダーが AWS のサービス であるアクションのみが表示されます。Region フィールドは、デフォルトでパイプラインと同じ AWS のリージョンになります。

   アクションプロバイダーの追加例と各プロバイダーのデフォルトフィールドの使用例については、「カスタムパイプラインを作成する (コンソール)」を参照してください。

   CodeBuild をビルドアクションまたはテストアクションとしてステージに追加するには、CodeBuild ユーザーガイド の CodeBuild で CodePipeline を使用して、コードをテストしビルドを実行する を参照してください。

   > **Note**
   >
   > GitHub などの一部のアクションプロバイダーでは、アクションの設定を完了するために、プロバイダーのウェブサイトに接続する必要があります。プロバイダーのウェブサイトに接続するときは、そのウェブサイトの認証情報を使用していることを確認します。AWS の認証情報を使用しないでください。

3. アクションの設定が完了したら、[保存] を選択します。

> **Note**
>
> コンソールビューでステージの名前を変更することはできません。新しいステージを変更後の名前で追加し、その後に古いステージを削除できます。古いステージを削除する前に、必要なすべてのアクションを新しいステージに追加したことを確認してください。

8. パイプラインの編集が終わったら、[保存] を選択して概要ページに戻ります。

> **Important**
>
> 変更を保存したら、元に戻すことはできません。パイプラインを再度編集する必要があります。変更を保存するときにパイプラインによりリビジョンが実行されている場合、その実行は完了しません。編集したパイプラインにより特定のコミットまたは変更を実行する場合は、パイプラインから手動で実行する必要があります。それ以外の場合、次のコミットまたは変更はパイプラインにより自動的に実行されます。

9. アクションをテストするには、[Release change] を選択して、パイプラインによりそのコミットを処理し、パイプラインのソースステージで指定したソースに変更をコミットします。または パイプラインを手動で開始する を使用して手動で変更をリリースする場合、AWS CLI で次の手順に従います。

パイプラインを編集する (AWS CLI)

パイプラインを編集するには、update-pipeline コマンドを使用します。

パイプラインを更新すると、CodePipeline はすべての実行中のアクションを正常に完了し、実行中のアクションが完了したステージとパイプラインの実行に失敗します。パイプラインが更新されたら、パイプラインを再実行する必要があります。パイプラインの実行に関する詳細については、「パイプラインを手動で開始する」を参照してください。

> **Important**
>
> パートナーアクションを含むパイプラインの編集に AWS CLI を使用することはできますが、パートナーアクションの JSON を手動で編集することはできません。これを行った場合、パートナーアクションはパイプライン更新後に失敗します。

パイプラインを編集するには

1. ターミナルセッション (Linux、macOS または Unix) またはコマンドプロンプト (Windows) を開き、get-pipeline コマンドを実行して、パイプライン構造を JSON ファイルにコピーします。たとえば、**MyFirstPipeline** という名前のパイプラインに対して、以下のコマンドを入力します。

   ```
   aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
   ```

   このコマンドは何も返しませんが、作成したファイルは、コマンドを実行したディレクトリにあります。

2. 任意のプレーンテキストエディタで JSON ファイルを開き、パイプラインに加える変更を反映するようにファイルの構造を変更します。たとえば、ステージを追加または削除すること、または、既存のステージに別のアクションを追加することができます。

   以下の例では、pipeline.json ファイルに別のデプロイステージを追加する方法を示しています。このステージは、*Staging* という名前の最初のデプロイステージの後に実行されます。

   > **Note**
   >
   > ここで示しているのは、そのファイルの一部であり、構造全体ではありません。詳細については、「CodePipeline パイプライン構造リファレンス」を参照してください。

   ```
   ,
           {
               "name": "Staging",
               "actions": [
                       {
                           "inputArtifacts": [
                               {
   ```

```
                            "name": "MyApp"
                        }
                    ],
                    "name": "Deploy-CodeDeploy-Application",
                    "actionTypeId": {
                        "category": "Deploy",
                        "owner": "AWS",
                        "version": "1",
                        "provider": "CodeDeploy"
                    },
                    "outputArtifacts": [],
                    "configuration": {
                        "ApplicationName": "CodePipelineDemoApplication",
                        "DeploymentGroupName": "CodePipelineDemoFleet"
                    },
                    "runOrder": 1
                }
            ]
        },
    {
        "name": "Production",
        "actions":  [
                {
                    "inputArtifacts": [
                        {
                            "name": "MyApp"
                        }
                    ],
                    "name": "Deploy-Second-Deployment",
                    "actionTypeId": {
                        "category": "Deploy",
                        "owner": "AWS",
                        "version": "1",
                        "provider": "CodeDeploy"
                    },
                    "outputArtifacts": [],
                    "configuration": {
                    "ApplicationName": "CodePipelineDemoApplication",
                    "DeploymentGroupName": "CodePipelineProductionFleet"
                    },
                    "runOrder": 1
                }
            ]
    }
```

```
    ]
}
```

CLI を使用して承認アクションをパイプラインに追加する方法については、「CodePipeline でパイプラインにマニュアルの承認アクションを追加する」を参照してください。

JSON ファイルの PollForSourceChanges パラメータが次のように設定されていることを確認します。

```
                "PollForSourceChanges": "false",
```

CodePipeline は、Amazon CloudWatch Events を利用して、CodeCommit ソースリポジトリやブランチ、または Amazon S3 ソースバケットの変更を検出します。次のステップには、手動でこれらのリソースを作成する手順が含まれています。フラグを false に設定すると、定期的なチェックが無効になります。これは、推奨される変更検出メソッドを使用する場合には必須ではありません。

3. パイプラインとは異なるリージョンにビルド、テスト、またはデプロイアクションを追加するには、パイプライン構造に以下を追加する必要があります。詳細な手順については、「CodePipeline でクロスリージョンアクションを追加する」を参照してください。

   - Region パラメータをアクションのパイプライン構造に追加します。
   - アクションの作成先のリージョンごとにアーティファクトバケットを指定するには、artifactStores パラメータを使用します。

4. get-pipeline コマンドを使用して取得したパイプライン構造を使用している場合、JSON ファイルの構造を変更する必要があります。metadata コマンドが使用できるように、ファイルから update-pipeline 行を削除する必要があります。JSON ファイルのパイプライン構造からセクションを削除します ("metadata": { } 行と、"created"、"pipelineARN"、および"updated"フィールド)。

   例えば、構造から以下の行を削除します。

```
"metadata": {
  "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
  "created": "date",
  "updated": "date"
  }
```

   ファイルを保存します。

5. パイプラインの編集に CLI を使用する場合には、推奨される変更検出リソースを手動でパイプライン用に管理する必要があります。

   - CodeCommit リポジトリでは、 CloudWatch Events ルールを作成する必要があります。詳細については、「CodeCommit ソースに対する EventBridge ルールを作成する (CLI)」を参照してください。
   - Amazon S3 ソースは、CloudWatch Events ルールおよび AWS CloudTrail 証跡を作成する必要があります。詳細については、「EventBridge と を使用する Amazon S3 ソースアクションへの接続 AWS CloudTrail」を参照してください。

6. 変更を適用するには、パイプライン JSON ファイルを指定して、update-pipeline コマンドを実行します。

   > **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 codepipeline update-pipeline --cli-input-json file://pipeline.json
   ```

   このコマンドは、編集したパイプラインの構造全体を返します。

   > **Note**
   >
   > update-pipeline コマンドは、パイプラインを停止します。update-pipeline コマンドを実行したときにパイプラインによりリビジョンが実行されている場合、その実行は停止します。更新されたパイプラインによりそのリビジョンを実行するには、パイプラインを手動で開始する必要があります。

7. CodePipeline コンソールを開き、編集したパイプラインを選択します。

   そのパイプラインには、行った変更が示されます。ソース場所を次に変更すると、修正した構造のパイプラインによりそのリビジョンが実行されます。

8. 修正した構造のパイプラインによりその最新のリビジョンを手動で実行するには、start-pipeline-execution コマンドを実行します。詳細については、「パイプラインを手動で開始する」を参照してください。

パイプラインの構造および想定値の詳細については、「CodePipeline パイプライン構造リファレンス」および AWS CodePipeline API リファレンス を参照してください。

CodePipeline でパイプラインと詳細を表示する

AWS CodePipeline コンソールまたは を使用して AWS CLI 、 AWS アカウントに関連付けられたパイプラインの詳細を表示できます。

トピック

- パイプラインを表示する (コンソール)
- パイプラインのアクションの詳細を表示する (コンソール)
- パイプラインの ARN とサービスロール ARN (コンソール) を表示します。
- パイプラインの詳細と履歴を表示する (CLI)
- 実行履歴でステージ条件のルール結果を表示する

パイプラインを表示する (コンソール)

パイプラインのステータス、移行、およびアーティファクトの更新を表示できます。

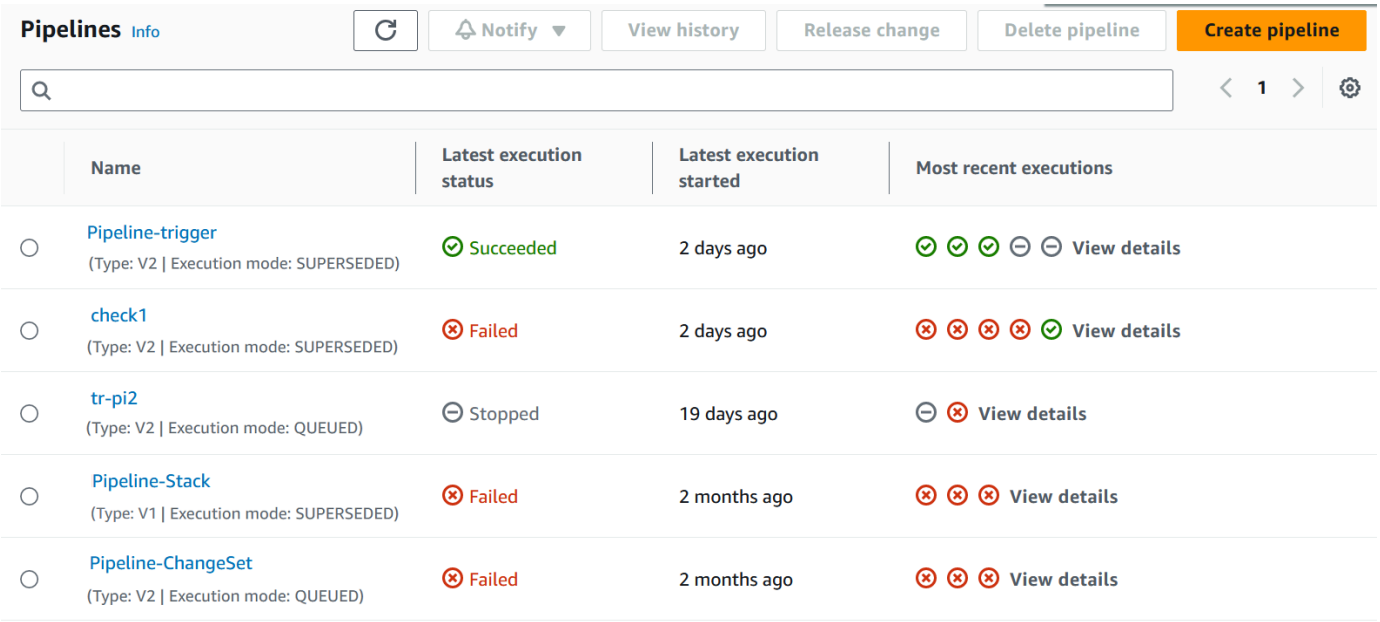
> **Note**
>
> 1 時間後には、パイプラインの詳細ビューがブラウザで自動的に更新されなくなります。現在の情報を表示するには、ページを更新します。

パイプラインを表示するには

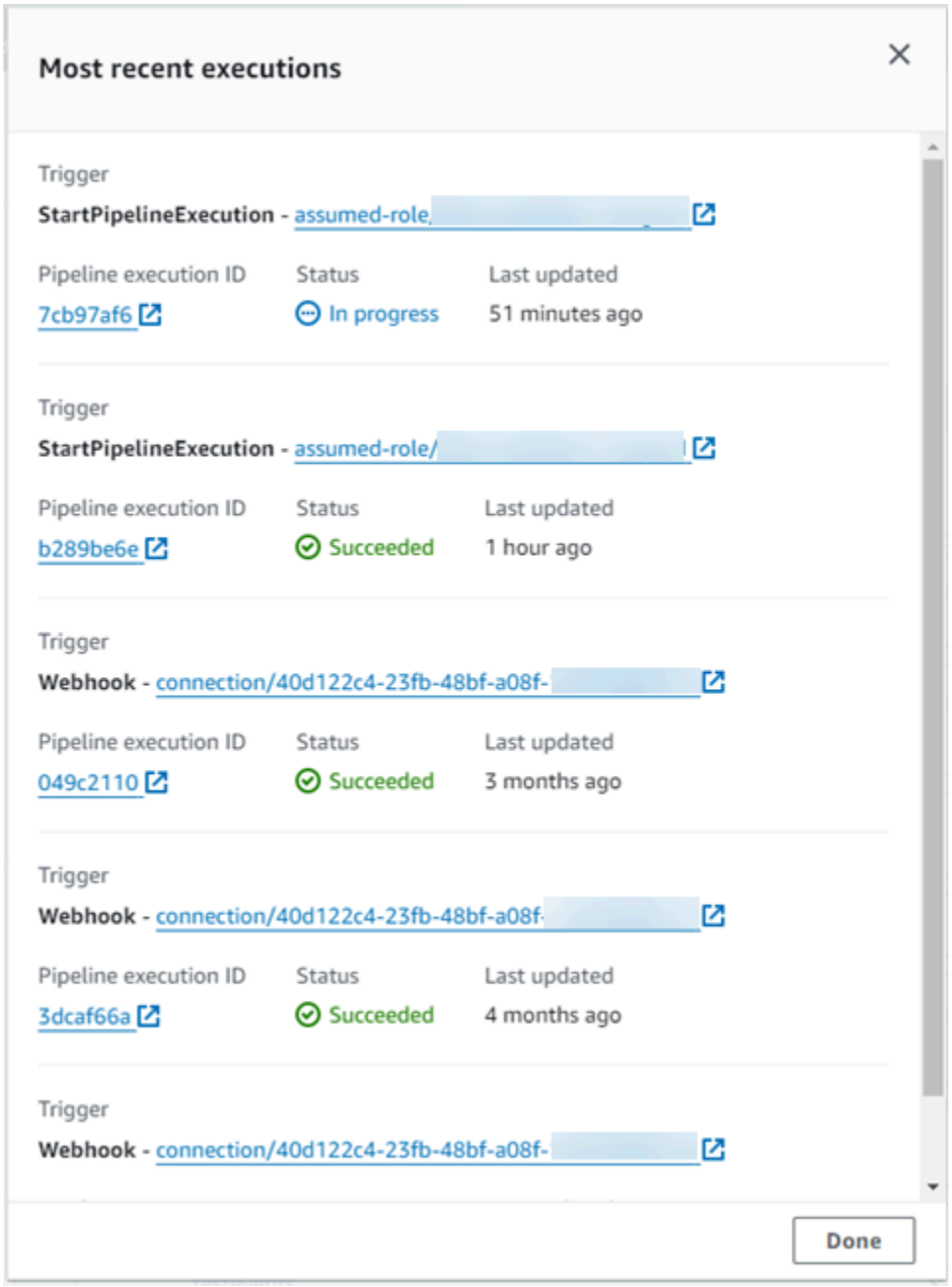
1. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

   [パイプライン] ページが表示されます。そのリージョンのすべてのパイプラインのリストが表示されます。

   AWS アカウントに関連付けられているすべてのパイプラインの名前、タイプ、ステータス、バージョン、作成日、最終変更日、および最後に開始された実行時刻が表示されます。
2. 直近 5 回の実行のステータスが表示されます。



特定の行の横にある [詳細を表示] を選択すると、最新の実行を一覧表示する詳細ダイアログボックスが表示されます。

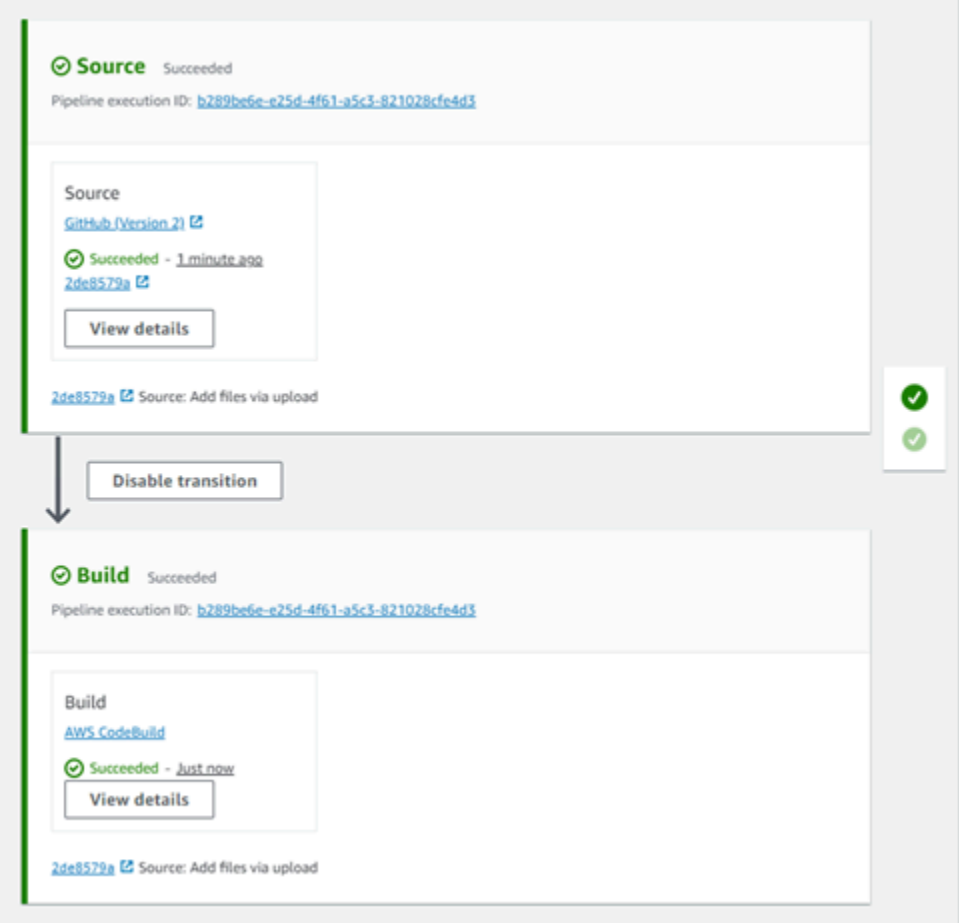


パイプラインの最新の実行の詳細を表示するには、[履歴の表示] を選択します。過去の実行については、実行 ID、ステータス、開始時刻と終了時刻、継続時間、コミット ID、メッセージなど、ソースアーティファクトに関連するリビジョンの詳細を表示できます。

> **Note**
>
> PARALLEL 実行モードのパイプラインの場合、メインパイプラインビューにはパイプライン構造や進行中の実行は表示されません。PARALLEL 実行モードのパイプラインの場合、パイプライン構造にアクセスするには、表示する実行の ID を実行履歴ページ

で選択します。左側のナビゲーションで [履歴] を選択し、並列実行の実行 ID を選択して、[可視化] タブでパイプラインを表示します。

3. 1 つのパイプラインの詳細を表示するには、[Name] でパイプラインを選択します。パイプラインの詳細ビューが開いて、各ステージの各アクションの状態や遷移の状態などが表示されます。



グラフィカルビューには、各ステージについて以下の情報が表示されます。

- ステージ名です。
- ステージ用に設定されたすべてのアクション。
- ステージ間の遷移の状態 (有効または無効)。ステージ間の矢印の状態によって示されます。有効な移行は矢印とその横にある [移行を無効にする] ボタンとともに示されます。無効にされた移行は、下に取り消し線が付いた矢印、およびその横の [移行を有効にする] ボタンで示されます。
- ステージのステータスを示すカラーバー:
  - グレー: まだ実行はなし

- 青: 進行中
- 緑: 成功
- 赤: 失敗

グラフィカルビューには、各ステージのアクションについて以下の情報も表示されます。

- アクションの名前。
- CodeDeploy などのアクションのプロバイダー
- アクションが最後に実行されたとき。
- アクションが成功したか失敗したか。
- アクションの最後の実行について、他の詳細へのリンク (使用可能であれば)。
- 最新のパイプラインの実行によりステージで実行されているソースリビジョンの詳細、または、CodeDeploy デプロイの場合は、ターゲットインスタンスにデプロイされた最新のソースリビジョンの詳細
- [詳細を表示] ボタンをクリックすると、アクションの実行、ログ、アクション設定に関する詳細を含むダイアログボックスが開きます。

> **Note**
>
> [ログ] タブは、CodeBuild とパイプラインのアカウントで実行された CloudFormation アクションで使用できます。

4. アクションのプロバイダーの詳細を表示するには、プロバイダーを選択します。例えば、上記の例のパイプラインの [Staging] または [Production] ステージで CodeDeploy を選択した場合、そのステージ用に設定されたデプロイグループの CodeDeploy コンソールのページが表示されます。
5. アクションの進捗状況を表示するには、進行中のアクション ([進行中] メッセージによって示される) の横に表示される [詳細] を選択します。アクションが進行中の場合は、段階的な進行状況と、実行されたステップまたはアクションが表示されます。
6. 手動承認用に設定されたアクションを承認または拒否するには、[確認] を選択します。
7. 正常に完了しなかったステージでアクションを再試行するには、[Retry] を選択します。
8. アクションが最後に実行されたときのステータスが、そのアクションの結果も含めて ([成功] または [失敗]) 表示されます。

パイプラインのアクションの詳細を表示する (コンソール)

各ステージのアクションの詳細など、パイプラインの詳細を表示できます。

> **Note**
>
> 1 時間後には、パイプラインの詳細ビューがブラウザで自動的に更新されなくなります。現在の情報を表示するには、ページを更新します。

パイプラインのアクションの詳細を表示するには

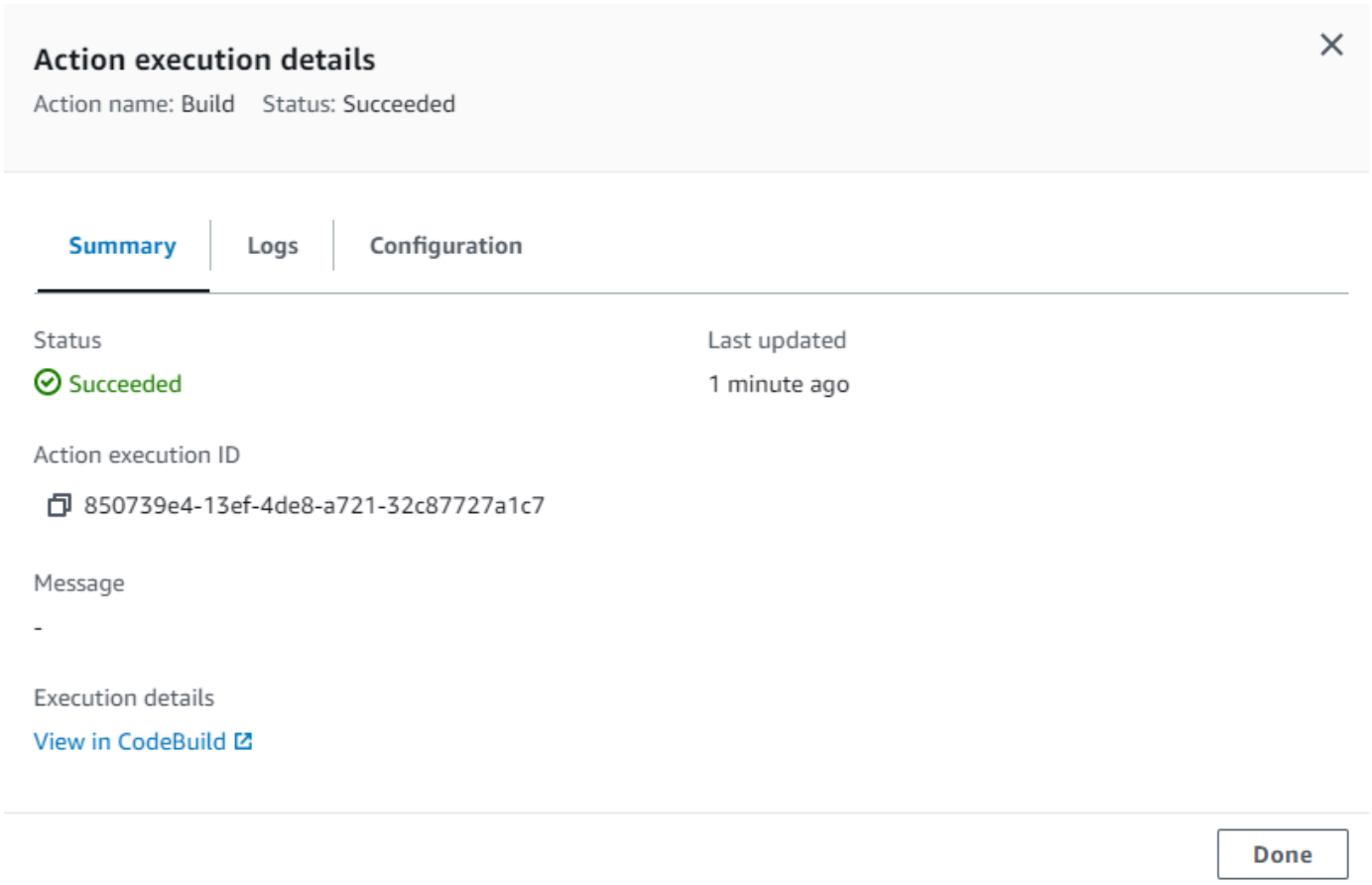
1. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

   [パイプライン] ページが表示されます。

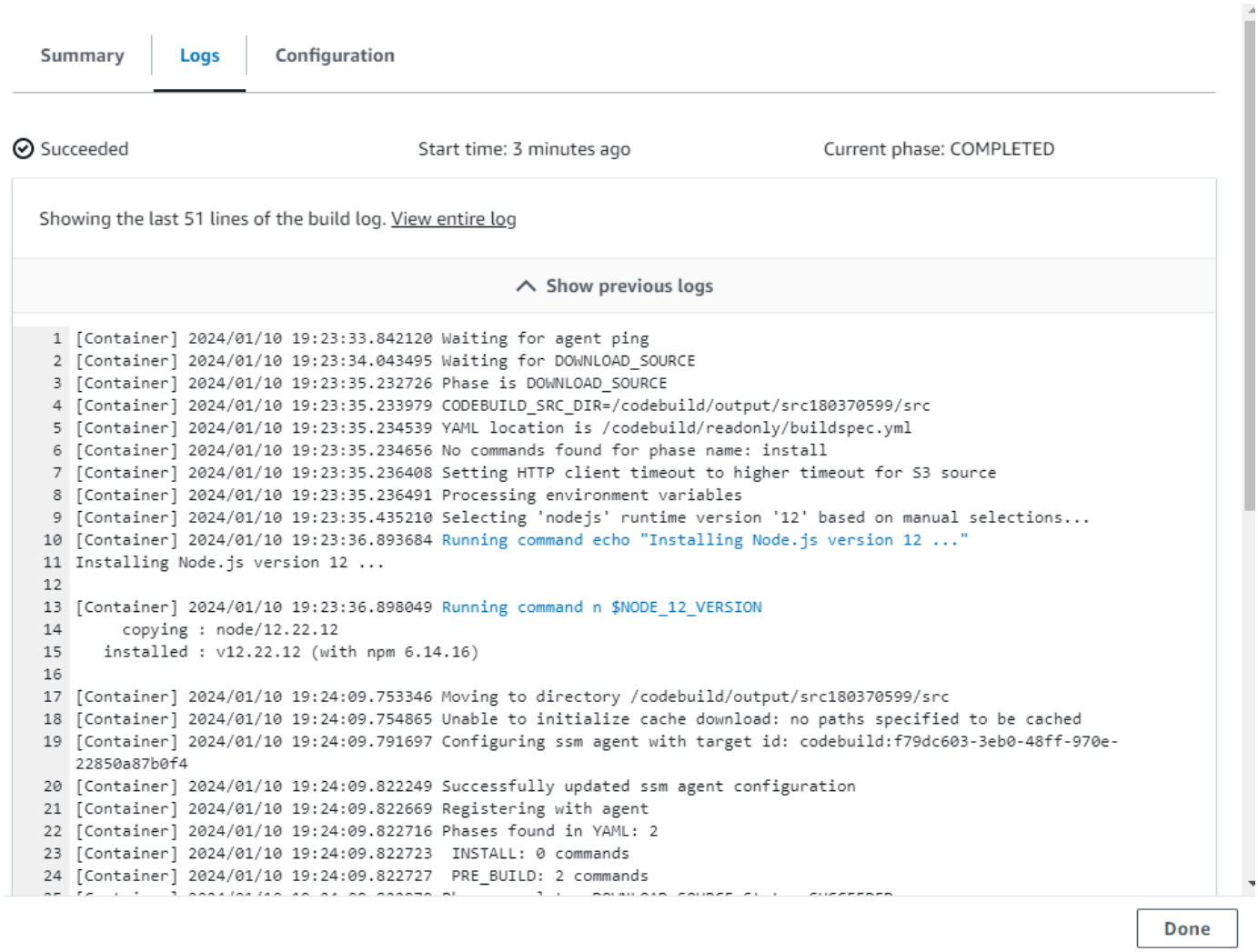
2. どのアクションでも、[詳細を表示] を選択すると、アクションの実行、ログ、およびアクション設定に関する詳細を含むダイアログボックスが開きます。

   > **Note**
   >
   > [ログ] タブは CodeBuild と CloudFormation アクションで使用できます。

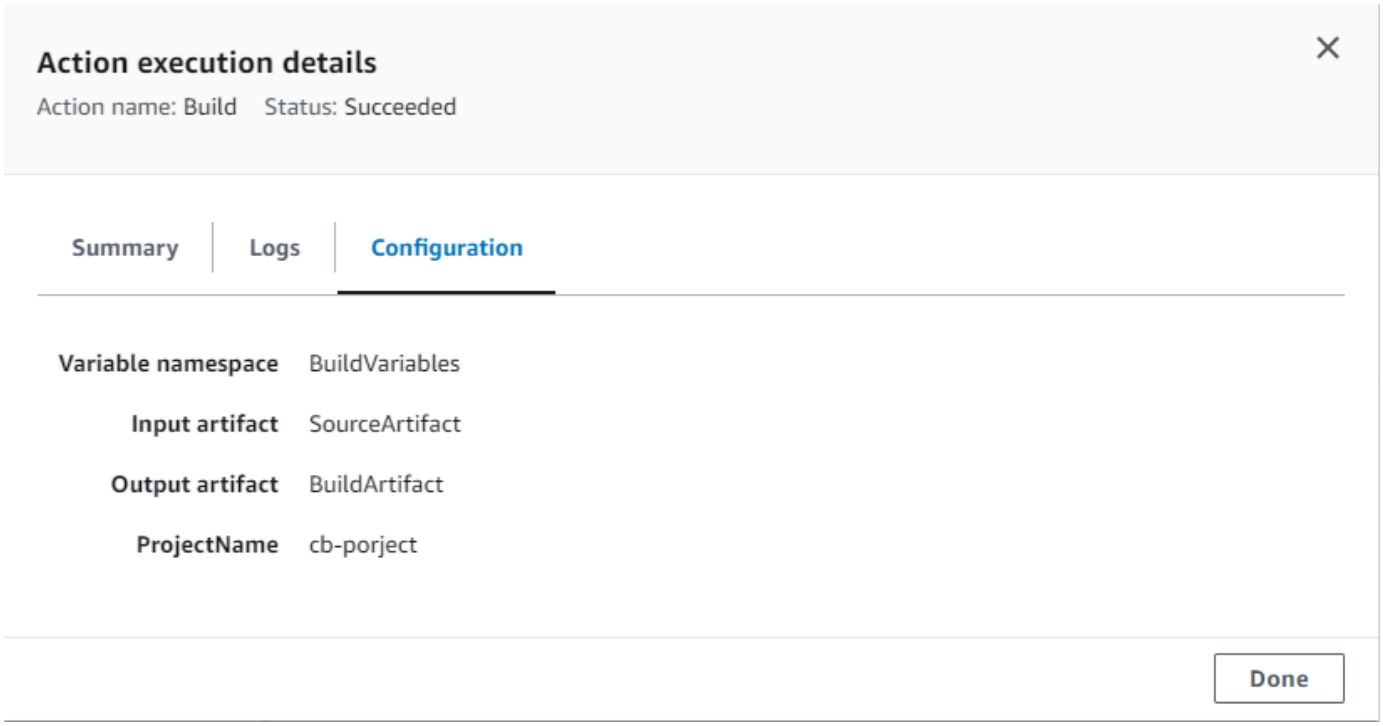
3. パイプラインのステージにあるアクションについてアクションの概要を表示するには、アクションの [詳細を表示] を選択し、[概要] タブを選択します。



4. ログ付きのアクションのアクションログを表示するには、アクションの [詳細を表示] を選択し、[ログ] タブを選択します。



5. アクションの設定の詳細を表示するには、[設定] タブを選択します。



パイプラインの ARN とサービスロール ARN (コンソール) を表示します。

コンソールを使用して、パイプライン ARN、サービスロール ARN、パイプラインアーティファクトストアなどの、パイプライン設定を表示できます。

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

   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。
2. パイプラインの名前を選択し、左側のナビゲーションペイン の Settings を選択します。ページでは以下を示します。

   - パイプライン名
   - パイプラインの Amazon リソースネーム (ARN)

     パイプライン ARN はこの形式で作成されます。

     arn:aws:codepipeline:*region*:*account*:pipeline-name

     パイプライン ARN の例:

     ```
     arn:aws:codepipeline:us-east-2:80398EXAMPLE:MyFirstPipeline
     ```

- パイプラインの CodePipeline サービスロール ARN
- パイプラインのバージョン
- パイプラインのためのアーティファクトストアの名前と場所

パイプラインの詳細と履歴を表示する (CLI)

パイプラインとパイプラインの実行の詳細を表示するには、以下のコマンドを実行します。

- list-pipelines AWS アカウントに関連付けられているすべてのパイプラインの概要を表示する コマンド。
- get-pipeline コマンドは、1 つのパイプラインの詳細を表示します。
- list-pipeline-executions パイプラインの最新の実行の概要を取得します。
- get-pipeline-execution パイプラインの実行に関する情報 (アーティファクト、パイプラインの実行 ID、パイプラインの名前、バージョン、ステータスなど) を表示します。
- get-pipeline-state コマンドでパイプライン、ステージ、およびアクションのステータスを表示します。
- list-action-executions でパイプラインのアクション実行の詳細を表示します。

1. ターミナル (Linux、macOS、または Unix) またはコマンドプロンプト (Windows) を開き、AWS CLI を使用してlist-pipelinesコマンドを実行します。

   ```
   aws codepipeline list-pipelines
   ```

   このコマンドは、 AWS アカウントに関連付けられているすべてのパイプラインのリストを返します。

2. パイプラインの詳細を表示するには、パイプラインの一意の名前を指定して、get-pipeline コマンドを実行します。例えば、*MyFirstPipeline* という名前のパイプラインの詳細を表示するには、以下のように入力します:

   ```
   aws codepipeline get-pipeline --name MyFirstPipeline
   ```

   このコマンドは、パイプラインの構造を返します。

実行履歴でステージ条件のルール結果を表示する

実行のルール結果により、ステージ条件がルールを実行してステージの結果 (ロールバックや失敗など) を適用したことを確認できます。

条件とルールの有効なステータス値は次のとおりです: `InProgress | Failed | Errored | Succeeded | Cancelled | Abandoned | Overridden`

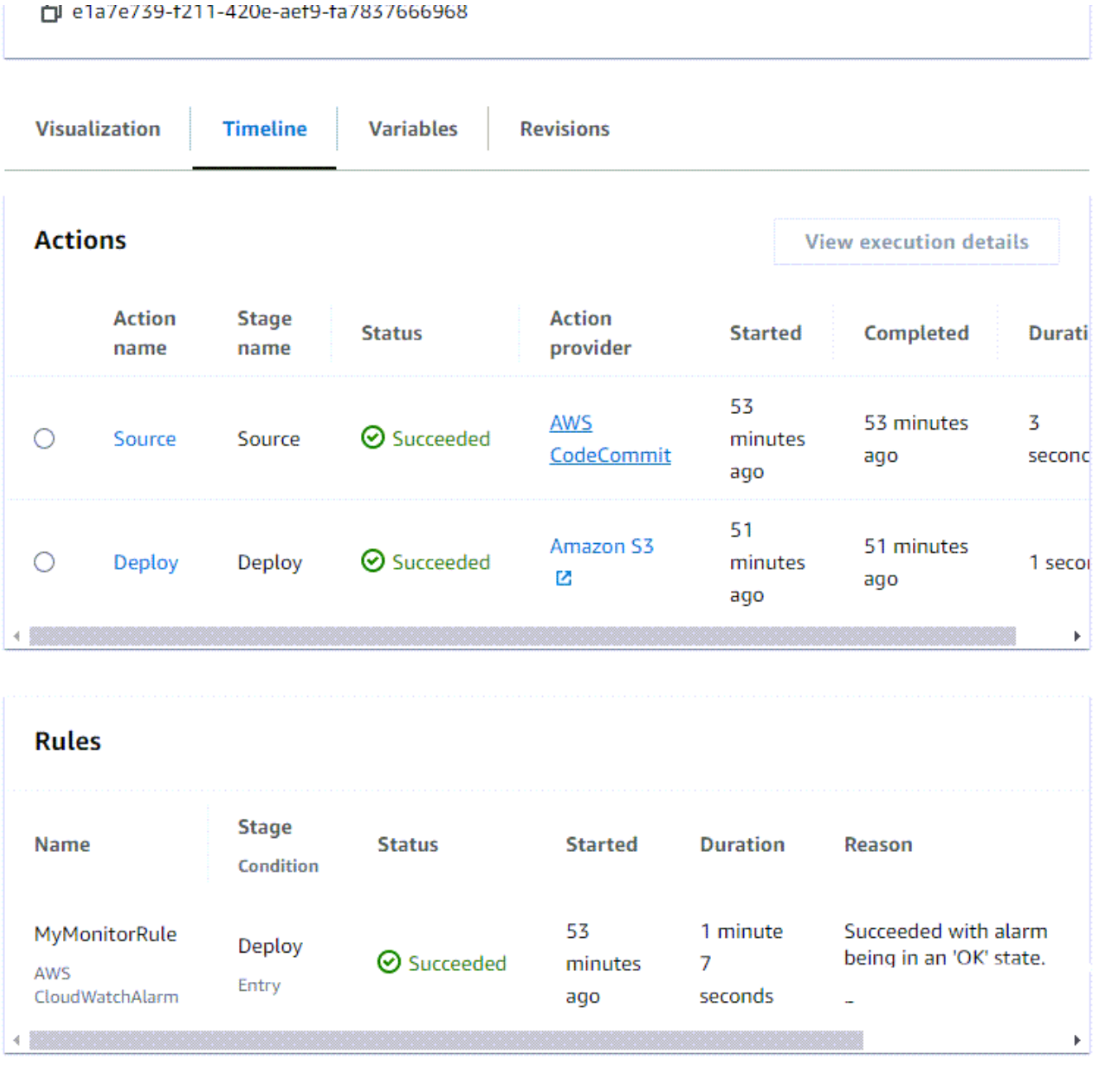
実行履歴でステージ条件のルール結果を表示する (コンソール)

コンソールを使用して実行のルール結果を表示し、ステージ条件がルールを実行してステージの結果を適用したことを確認できます。

ステージ条件のルール結果を表示する (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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

   AWS アカウント に関連付けられているすべてのパイプラインの名前とステータスが表示されます。
2. [名前] で、表示するパイプラインの名前を選択します。
3. [履歴] を選択し、実行を選択します。[履歴] ページで、[タイムライン] タブを選択します。[ルール] で、実行のルール結果を表示します。



ステージ条件のルール結果を **list-rule-executions** で表示する (CLI)

CLI を使用して実行のルール結果を表示し、ステージ条件がルールを実行してステージの結果を適用したことを確認できます。

- ターミナル (Linux、macOS、Unix) またはコマンドプロンプト (Windows) を開きます。次に AWS CLI を使用し、*MyPipeline* という名前のパイプラインに対して list-rule-executions コマンドを実行します。

```
aws codepipeline list-rule-executions --pipeline-name MyFirstPipeline
```

このコマンドは、パイプラインに関連するすべての完了済みルール実行のリストを返します。

次の例は、ステージ条件で *MyMonitorRule* という名前のルールを使用して返されたパイプラインに関するデータを示しています。

```
{
    "ruleExecutionDetails": [
        {
            "pipelineExecutionId": "e1a7e739-f211-420e-aef9-fa7837666968",
            "ruleExecutionId": "3aafc0c7-0e1c-44f1-b357-d1b16a28e483",
            "pipelineVersion": 9,
            "stageName": "Deploy",
            "ruleName": "MyMonitorRule",
            "startTime": "2024-07-29T15:55:01.271000+00:00",
            "lastUpdateTime": "2024-07-29T15:56:08.682000+00:00",
            "status": "Succeeded",
            "input": {
                "ruleTypeId": {
                    "category": "Rule",
                    "owner": "AWS",
                    "provider": "CloudWatchAlarm",
                    "version": "1"
                },
                "configuration": {
                    "AlarmName": "CWAlarm",
                    "WaitTime": "1"
                },
                "resolvedConfiguration": {
                    "AlarmName": "CWAlarm",
                    "WaitTime": "1"
                },
                "region": "us-east-1",
                "inputArtifacts": []
            },
            "output": {
                "executionResult": {
                    "externalExecutionSummary": "Succeeded with alarm 'CWAlarm'
 being i
n an 'OK' state."
                }
```

```
            }
        }
```

CodePipeline でパイプラインを作成します。

パイプラインは好きなときに編集して機能を変更することができますが、削除することもできます。AWS CodePipeline コンソール、または delete-pipeline の AWS CLI コマンドを使用して、パイプラインを削除できます。

トピック

- パイプラインを削除する (コンソール)
- パイプラインを削除する (CLI)

パイプラインを削除する (コンソール)

パイプラインを削除するには

1. AWS マネジメントコンソール にサインインして、CodePipeline コンソール (https://console.aws.amazon.com/codesuite/codebuild/home を開きます。

   AWS アカウントに関連付けられているすべてのパイプラインの名前とステータスが表示されます。
2. [Name] で、削除するパイプラインの名前を選択します。
3. パイプライン詳細ページで、[編集] を選択します。
4. [Edit] ページで、[Delete] を選択します。
5. フィールドに「**delete**」と入力して確認し、[Delete (削除)] を選択します。

> **Important**
>
> このアクションを元に戻すことはできません。

パイプラインを削除する (CLI)

AWS CLI を使用してパイプラインを手動で削除するには、[パイプラインを削除する] コマンドを使用します。

> **Important**
>
> パイプラインを削除すると、元に戻すことはできません。確認ダイアログボックスは表示されません。コマンド実行後、パイプラインは削除されますが、パイプラインで使用したリソースは削除されません。これにより、そのようなリソースを使用する新しいパイプラインを作成し、ソフトウェアのリリースを自動化しやすくなります。

パイプラインを削除するには

1. ターミナル (Linux、macOS、または Unix) またはコマンドプロンプト (Windows) を開き、AWS CLI を使用して delete-pipeline コマンドを実行し、削除するパイプラインの名前を指定します。たとえば、[*MyFirstPipeline*] という名前のパイプラインを削除するには、以下のようにします。

   ```
   aws codepipeline delete-pipeline --name MyFirstPipeline
   ```

   このコマンドは何も返しません。

2. 不要になったリソースをすべて削除します。

   > **Note**
   >
   > パイプラインを削除しても、パイプラインで使用されているリソースは削除されません。たとえば、コードのデプロイに使用した CodeDeploy または Elastic Beanstalk アプリケーションは削除されません。また、CodePipeline コンソールからパイプラインを作成した場合は、パイプラインのアーティファクトの保存用に作成された Amazon S3 バケット CodePipeline も削除されません。不要なリソースは必ず削除し、今後課金されないようにしてください。たとえば、コンソールを使用して初めてパイプラインを作成する場合、1 つの Amazon S3 バケットを CodePipeline で作成し、すべてのパイプラインのすべてのアーティファクトを保存します。すべてのパイプラインを削除した場合は、「バケットの削除」のステップに従います。

別の AWS アカウントのリソースを使用する CodePipeline でパイプラインを作成する

他の AWS アカウントによって作成し、管理されるリソースを使用するパイプラインを作成します。例えば、一つのアカウントにパイプラインを、別のアカウントに CodeDeploy リソースを使用します。

> **Note**
>
> 複数のアカウントからのアクションを使用してパイプラインを作成する場合は、クロスアカウントパイプラインの制限内で、それらが引き続き案件にアクセスできるようにアクションを構成する必要があります。クロスアカウントのアクションには、以下の制限が適用されます。
>
> - 一般的に、アクションは次の場合にのみ、アーティファクトを消費できます。
>   - アーティファクトがパイプラインアカウントと同じアカウントにある
>   - アーティファクトが別のアカウントのアクションに対してパイプラインアカウントで作成されている
>   - アーティファクトが、アクションと同じアカウントで前のアクションによって生成されている
>
>   つまり、どちらのアカウントもパイプラインアカウントでない場合は、あるアカウントから別のアカウントにアーティファクトを渡すことはできません。
> - 以下のアクションタイプでは、クロスアカウントアクションはサポートされていません。
>   - Jenkins ビルドアクション

この例では、使用する AWS Key Management Service (AWS KMS) キーを作成し、パイプラインにキーを追加し、クロスアカウントアクセスを有効にするようにアカウントポリシーとロールを設定する必要があります。 AWS KMS キーの場合、キー ID、キー ARN、またはエイリアス ARN を使用できます。

> **Note**
>
> エイリアスは、カスタマーマスターキー (KMS) を作成したアカウントでのみ認識されます。クロスアカウントアクションの場合、キー ID またはキー ARN のみを使用してキーを識別で

きます。クロスアカウントアクションには他のアカウント (AccountB) のロールを使用するため、キー ID を指定すると他のアカウント (AccountB) のキーが使用されます。

このウォークスルーおよびサンプルでは、*AccountA*は、パイプラインの作成に使用したアカウントです。パイプラインアーティファクトの保存に使用される Amazon S3 バケットと、 が使用するサービスロールにアクセスできます AWS CodePipeline。*AccountB* は、CodeDeploy アプリケーション、デプロイグループ、および CodeDeploy によって使用されるサービスロールを作成するために使用したアカウントです。

AccountA でパイプラインを編集して、*AccountB* で作成した CodeDeploy アプリケーションを使用するには、*AccountA* を以下のようにします。

- *AccountB* の ARN またはアカウント ID をリクエストします (このウォークスルーで、*AccountB* ID は*012ID_ACCOUNT_B*)。
- パイプラインの リージョンで AWS KMS カスタマーマネージドキーを作成または使用し、そのキーを使用するアクセス許可をサービスロール (*CodePipeline_Service_Role*) と *AccountB* に付与します。
- *AccountB* に許可する Amazon S3 バケットポリシーを作成し、Amazon S3 バケットへのアクセス (例えば、 *codepipeline-us-east-2-1234567890*)。
- *AccountA* に許可されたポリシーが *AccountB* によって設定されているロールを前提に、そのポリシーをサービスロール (*CodePipeline_Service_Role*) にアタッチします。
- パイプラインを編集して、デフォルト AWS KMS キーの代わりにカスタマーマネージドキーを使用します。

AccountB のリソースが、*AccountA* で作成されているパイプラインにアクセスできるようにするには、*AccountB* は以下のように行います。

- *AccountA* の ARN またはアカウント ID をリクエストします (このウォークスルーで、*AccountA* ID は *012ID_ACCOUNT_A*)。
- に適用するポリシーを作成します。Amazon EC2 インスタンスロール Amazon S3 バケットへのアクセスを許可する CodeDeploy (*codepipeline-us-east-2-1234567890*)。
- CodeDeploy 用に設定された Amazon EC2 インスタンスロールに適用されるポリシーを作成し、*AccountA* のパイプラインアーティファクトの暗号化に使用される AWS KMS カスタマーマネージドキーへのアクセスを許可します。

- IAM ロール (*CrossAccount_Role*) を設定して信頼関係ポリシーにアタッチし、*AccountA* の CodePipeline サービスロールがロールを引き受けることを許可します。
- パイプラインで必要なデプロイリソースにアクセスできるようにするポリシーを作成し、*CrossAccount_Role* にアタッチします。
- Amazon S3 バケット (*codepipeline-us-east-2-1234567890*) にアクセスできるようにするポリシーを作成し、それを *CrossAccount_Role*にアタッチします。

トピック

- 前提条件: AWS KMS 暗号化キーを作成する
- ステップ 1: アカウントポリシーおよびロールをセットアップする
- ステップ 2: パイプラインを編集する

前提条件: AWS KMS 暗号化キーを作成する

カスタマーマネージドキーは、すべての AWS KMS キーと同様に、リージョンに固有です。パイプラインが作成されたリージョンと同じリージョン (など) にカスタマーマネージド AWS KMS キーを作成する必要がありますus-east-2。

でカスタマーマネージドキーを作成するには AWS KMS

1. *AccountA* AWS マネジメントコンソール を使用して にサインインし、 コンソールを開きます AWS KMS 。
2. 左側で [カスタマー管理キー]を選択します。
3. [キーの作成] を選択します。[キーの設定] で、[対称] のデフォルトを選択したまま、[次] を選択します。
4. エイリアス に、このキーに使用するエイリアス(*PipelineName-Key*など)を入力します。必要に応じて、このキーの説明とタグを入力し、[次] を選択します。
5. [キー管理アクセス許可の定義] で、このキーの管理者となるロールを選択し、[次へ] を選択します。
6. [キーの利用方法許可の定義] の [このアカウント] で、パイプラインのサービスロールの名前 (例えば CodePipeline_Service_Role など) を選択します。「その他の AWS アカウント」で、「別の AWS アカウントの追加」を選択します。ARN の一部として *AccountB* のアカウント ID を入力し、[次へ] を選択します。
7. [キーポリシーの確認と編集]で、ポリシーを確認し、[完了]を選択します。

8. キーのリストから、キーのエイリアスを選択し、その ARN (*arn:aws:kms:us-east-2:012ID_ACCOUNT_A:key/2222222-3333333-4444-556677EXAMPLE*など) にコピーします。この ARN は、パイプラインの編集時とポリシーの設定時に必要になります。

ステップ 1: アカウントポリシーおよびロールをセットアップする

AWS KMS キーを作成したら、クロスアカウントアクセスを有効にするポリシーを作成してアタッチする必要があります。作成するには、*AccountA*および*AccountB*の両方からアクションを行う必要があります。

トピック

- パイプラインを作成するポリシーおよびロールをアカウントに設定する (AccountA)
- AWS リソースを所有するアカウントでポリシーとロールを設定する (AccountB)

パイプラインを作成するポリシーおよびロールをアカウントに設定する (*AccountA*)

別の AWS アカウントに関連付けられた CodeDeploy リソースを使用するパイプラインを作成するには、*AccountA* がアーティファクトの保存に使用される Amazon S3 バケットと CodePipeline のサービスロールの両方のポリシーを設定する必要があります。

AccountB へのアクセスを許可する Amazon S3 バケットのポリシーを作成するには (コンソール)

1. *AccountA* AWS マネジメントコンソール で にサインインし、https://console.aws.amazon.com/s3/ で Amazon S3 コンソールを開きます。
2. Amazon S3 バケットのリストで、パイプラインのアーティファクトが保存される Amazon S3バケットを選択します。このバケットの名前は です。*region* はパイプラインを作成した AWS リージョンでcodepipeline-*region*-*1234567EXAMPLE*、*1234567EXAMPLE* はバケット名が一意であることを保証する 10 桁の乱数です (例: *codepipeline-us-east-2-1234567890*)。
3. Amazon S3 バケットの詳細ページで、[Properties] を選択します。
4. プロパティペインで、[アクセス許可] を展開し、[バケットポリシーの追加]を選択します。

> **Note**
>
> ポリシーがAmazon S3バケットにすでにアタッチされている場合は、バケットポリシーの編集 を選択します。以下の例のステートメントを既存のポリシーに追加できます。新

しいポリシーを追加するには、リンクを選択し、 AWS ポリシージェネレーターの指示に従います。詳細については、「IAMポリシーの概要」を参照してください。

5. バケットポリシーエディターウィンドウで *AccountB* にパイプラインアーティファクトへのアクセスを許可するポリシーを作成し、カスタムソースやビルドアクションなどのアクションによってアーティファクトが作成された場合、出力アーティファクトを追加する機能を *AccountB* に追加します。
6. [保存] を選択したら、ポリシーエディタを閉じます。
7. [保存] を選択して、Amazon S3 バケットに対するアクセス権限を保存します。

CodePipeline のサービスロールのポリシーを作成するには (コンソール)

1. *AccountA* AWS マネジメントコンソール で にサインインし、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)] を選択します。
3. [ロール名] の下にあるロールのリストで、 CodePipeline のサービスロールの名前を選択します。
4. [アクセス許可] タブで [インラインポリシーの追加]を選択します。
5. [JSON] タブをクリックし、次のポリシーを入力して許可します。 *AccountB* を選択してロールを引き受けることができます。次の例では、*012ID_ACCOUNT_B*は、*AccountB*の ARN です。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "sts:AssumeRole",
        "Resource": [
            "arn:aws:iam::111122223333:role/*"
        ]
    }
}
```

6. [ポリシーの確認]を選択します。
7. [名前] に、このポリシーの名前を入力します。[Create policy]を選択します。

AWS リソースを所有するアカウントでポリシーとロールを設定する (*AccountB*)

CodeDeploy にアプリケーション、デプロイ、デプロイグループを作成したら、あわせて [Amazon EC2 インスタンスロール] を作成します。([Run Deployment Walkthrough]ウィザードを使用している場合はこのロールが作成されますが、手動で作成することもできます。) *AccountA* で作成されたパイプラインは *AccountB* で CodeDeploy リソース作成時に使用されます。以下を実行する必要があります。

- パイプラインアーティファクトが保存されている Amazon S3 バケットにアクセスできるようにするインスタンスロールのポリシーを設定します。
- クロスアカウントアクセス用に設定されている*AccountB*に 2 番目のロールを作成します。

  この 2 番目のロールは、*AccountA* の Amazon S3 バケットにアクセスできるだけでなく、CodeDeploy リソースへのアクセスを許可するポリシーと、*AccountA* の CodePipeline サービスロールにロールを引き受けることを許可するポリシーを含んでいる必要があります。

> **Note**
>
> これらのポリシーは、別の AWS アカウントを使用して作成されたパイプラインで使用する CodeDeploy リソースの設定に固有です。その他の AWS リソースには、リソース要件に固有のポリシーが必要です。

CodeDeploy (コンソール) 用に設定した Amazon EC2 インスタンスロールのポリシーを作成するには

1. *AccountB* AWS マネジメントコンソール を使用して にサインインし、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)]を選択します。
3. [ロール名] の下にあるロールのリストで、 CodeDeploy アプリケーションの Amazon EC2 インスタンスロールとして使用するサービスロールの名前を選択します。このロール名はさまざまで、デプロイグループで複数のインスタンスロールを使用できます。詳細については、「Amazon EC2 インスタンスの 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。
4. [Permissions] (アクセス許可) タブで [Add inline policy] (インラインポリシーの追加) を選択します。

5. JSON タブで、以下のポリシーを入力して、 Amazon S3 バケットへのアクセスを許可します。*AccountA* パイプラインのアーティファクトを保存するには (この例では、*codepipeline-us-east-2-1234567890*):

   JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
     {
       "Effect": "Allow",
       "Action": [
         "s3:Get*"
       ],
       "Resource": [
         "arn:aws:s3:::amzn-s3-demo-bucket/*"
       ]
     },
     {
       "Effect": "Allow",
       "Action": [
         "s3:ListBucket"
       ],
       "Resource": [
         "arn:aws:s3:::amzn-s3-demo-bucket"
       ]
     }
   ]
 }
```

6. [ポリシーの確認]を選択します。
7. [名前] に、このポリシーの名前を入力します。[Create policy]を選択します。
8. の 2 番目のポリシーを作成します AWS KMS 。ここで、 ***arn:aws:kms:us-east-1:012ID_ACCOUNT_A:key/2222222-3333333-4444-556677EXAMPLE***は *AccountA* で作成され、*AccountB* が使用できるように設定されたカスタマーマネージドキーの ARN です。

   JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:DescribeKey",
                "kms:GenerateDataKey*",
                "kms:Encrypt",
                "kms:ReEncrypt*",
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-
east-1:111122223333:key/2222222-3333333-4444-556677EXAMPLE"
            ]
        }
    ]
}
```

> **Important**
>
> 次に示すように、このポリシーで *AccountA* のアカウント ID を AWS KMS キーのリソース ARN の一部として使用する必要があります。使用しない場合、ポリシーは機能しません。

9. [ポリシーの確認]を選択します。
10. [名前] に、このポリシーの名前を入力します。[Create policy]を選択します。

ここで、クロスアカウントアクセスに使用する IAMロールを作成し、*AccountA* の CodePipeline サービスロールがロールを引き受けられるように設定します。このロールは、CodeDeploy リソースへのアクセスを許可するポリシーと、*AccountA* でアーティファクトの保存に使用される Amazon S3 バケットが含まれている必要があります。

IAM でクロスアカウントロールを設定するには

1. *AccountB* AWS マネジメントコンソール を使用して にサインインし、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。[ロールの作成] を選択します。

3. [Select type of trusted entity](信頼できるエンティティのタイプを選択) で、[Another AWS account](別のアカウント) を選択します。このロールを使用できるアカウントを指定する のアカウント ID に、CodePipeline (AWS Account*AccountA*) でパイプラインを作成するアカウントのアカウント ID を入力し、次へ: アクセス許可を選択します。

   > **Important**
   >
   > この手順では、*AccountB*と*AccountA*との間に信頼関係ポリシーを作成します。ただし、これによりアカウントへのルートレベルのアクセスが許可されるため、CodePipeline は *AccountA* の CodePipeline サービスロールにスコープを絞ることを推奨しています。ステップ 16 に従ってアクセス許可を制限します。

4. [Attach permissions policies (アクセス許可ポリシーのアタッチ)] ページで、[AmazonS3ReadOnlyAccess]、[Next :タグ] の順に選択します。

   > **Note**
   >
   > これは、使用するポリシーではありません。ウィザードを完了するために、ポリシーを選択する必要があります。

5. [Next: Review (次へ: レビュー)]を選択します。ロール名 に (*CrossAccount_Role##*) のロールを名前に入力します。IAMの命名規則に従う限り、このロールの名前は任意に指定できます。ロールの目的が明確になる名前を付けることを検討してください。[Create Role]を選択します。

6. ロールのリストから、作成したポリシー (*CrossAccount_Role* など) を選択して、そのロールの [Summary] ページを開きます。

7. [Permissions] (アクセス許可) タブで [Add inline policy] (インラインポリシーの追加) を選択します。

8. [JSON] タブで、以下のポリシーを入力して、CodeDeploy リソースへのアクセスを許可します。

   JSON

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

```
      "Action": [
        "codedeploy:CreateDeployment",
        "codedeploy:GetDeployment",
        "codedeploy:GetDeploymentConfig",
        "codedeploy:GetApplicationRevision",
        "codedeploy:RegisterApplicationRevision"
      ],
      "Resource": "*"
    }
  ]
}
```

9. [ポリシーの確認]を選択します。
10. [名前] に、このポリシーの名前を入力します。[Create policy]を選択します。
11. [Permissions] (アクセス許可) タブで [Add inline policy] (インラインポリシーの追加) を選択します。
12. [JSON] タブを選択し、 以下のポリシーを入力して、このロールに *AccountA* の Amazon S3 バケットに対する入力アーティファクトの取得と出力アーティファクトの保存を許可します。

    JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
     {
       "Effect": "Allow",
       "Action": [
         "s3:GetObject*",
         "s3:PutObject",
         "s3:PutObjectAcl"
       ],
       "Resource": [
         "arn:aws:s3:::amzn-s3-demo-bucket/*"
       ]
     }
   ]
}
```

13. [ポリシーの確認]を選択します。
14. [名前] に、このポリシーの名前を入力します。[Create policy]を選択します。

15. [アクセス許可] タブで、[ポリシー名] の下のポリシーの一覧で AmazonS3ReadOnlyAccess を探し、ポリシーの横にある削除アイコン (X) を選択します。プロンプトが表示されたら、[Detach] を選択します。
16. [信頼関係] タブ、[信頼ポリシーを編集] の順に選択します。左側の列で [プリンシパルを追加] オプションを選択します。[プリンシパルタイプ] で、[IAM ロール] を選択し、*AccountA* の CodePipeline サービスロールの ARN を指定します。[AWS プリンシパル] のリストから `arn:aws:iam::Account_A:root` を削除し、[ポリシーを更新] を選択します。

ステップ 2: パイプラインを編集する

CodePipeline コンソールを使用して、別の AWS アカウントに関連付けられたリソースを使用するパイプラインを作成または編集することはできません。ただし、 コンソールを使用してパイプラインの一般的な構造を作成し、 を使用してパイプライン AWS CLI を編集し、それらのリソースを追加できます。または、既存のパイプラインの構造を使用して、リソースを手動で追加することもできます。

別の AWS アカウントに関連付けられたリソースを追加するには (AWS CLI)

1. ターミナル (Linux, macOS , or Unix) またはコマンドプロンプト (Windows) で、リソースを追加するパイプラインに対して get-pipeline コマンドを実行します。コマンドの出力を JSON ファイルにコピーします。例えば、MyFirstPipeline という名前のパイプラインに対しては、以下のようなコマンドを入力します。

   ```
   aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
   ```

   出力は、*pipeline.json* ファイルを開きます。
2. 任意のプレーンテキストエディタで JSON ファイルを開きます。アーティファクトストアの "type": "S3" の後に、KMS 暗号化キー、ID、タイプ情報を追加します。ここで、*codepipeline-us-east-2-1234567890* は、Amazon S3 バケット名で使用されるパイプラインのアーティファクトの保存にされ、***arn:aws:kms:us-east-1:012ID_ACCOUNT_A:key/2222222-3333333-4444-556677EXAMPLE*** は、先ほど作成したカスタマー管理キーの ARN です。

   ```
   {
     "artifactStore": {
       "location": "codepipeline-us-east-2-1234567890",
       "type": "S3",
   ```

```
    "encryptionKey": {
      "id": "arn:aws:kms:us-
east-1:012ID_ACCOUNT_A:key/2222222-3333333-4444-556677EXAMPLE",
      "type": "KMS"
    }
  },
```

3. *AccountB* に関連付けられている CodeDeploy リソースを使用するためのデプロイアクションをステージに追加して、roleArn 値など作成したクロスアカウントロール (*CrossAccount_Role*) に指定します。

   以下の例では、*ExternalDeploy*という名前のデプロイアクションを追加する JSON を示しています。また、*AccountB*で作成された CodeDeploy リソースを Staging という名前の*####*で使用しています。以下の例では、*AccountB*の ARN は*012ID_ACCOUNT_B*です。

```
,
            {
                "name": "Staging",
                "actions": [
                    {
                        "inputArtifacts": [
                            {
                                "name": "MyAppBuild"
                            }
                        ],
                        "name": "ExternalDeploy",
                        "actionTypeId": {
                            "category": "Deploy",
                            "owner": "AWS",
                            "version": "1",
                            "provider": "CodeDeploy"
                        },
                        "outputArtifacts": [],
                        "configuration": {
                            "ApplicationName": "AccountBApplicationName",
                            "DeploymentGroupName": "AccountBApplicationGroupName"
                        },
                        "runOrder": 1,
                        "roleArn":
 "arn:aws:iam::012ID_ACCOUNT_B:role/CrossAccount_Role"
                    }
                ]
```

```
            }
```

> **Note**
>
> これは、パイプライン全体の JSON ではなく、ステージでのアクションの構造です。

4. metadata コマンドが使用できるように、ファイルから update-pipeline 行を削除する必要があります。JSON ファイルのパイプライン構造からセクションを削除します ("metadata": { } 行と、"created"、"pipelineARN"、および"updated"フィールド)。

   例えば、構造から以下の行を削除します。

```
"metadata": {
  "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
  "created": "date",
  "updated": "date"
  }
```

   ファイルを保存します。

5. 変更を適用するには、以下のように、パイプライン JSON ファイルを指定して、 update-pipeline コマンドを実行します。

> **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 codepipeline update-pipeline --cli-input-json file://pipeline.json
```

   このコマンドは、編集したパイプラインの構造全体を返します。

別の AWS アカウントに関連付けられたリソースを使用するパイプラインをテストするには

1. ターミナル (Linux, macOS , or Unix) またはコマンドプロンプト (Windows) で、以下のように、パイプラインの名前を指定して、start-pipeline-execution コマンドを実行します。

```
aws codepipeline start-pipeline-execution --name MyFirstPipeline
```

詳細については、「パイプラインを手動で開始する」を参照してください。

2. *AccountA* AWS マネジメントコンソール で にサインインし、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。

3. [Name]で、先ほど編集したパイプラインの名前を選択します。これにより、パイプラインの詳細ビューが開いて、パイプラインの各ステージの各アクションの状態などがわかります。

4. パイプラインの進行状況を監視します。別の AWS アカウントに関連付けられたリソースを使用するアクションの成功メッセージを待ちます。

> **Note**
>
> *AccountA* を使用してサインインしている間にアクションの詳細を表示しようとすると、エラーが発生します。サインアウトしてから、*AccountB* でサインインして、CodeDeploy でデプロイの詳細を表示します。

ポーリングパイプラインをイベントベースの変更検出の使用に移行する

AWS CodePipeline は、完全なend-to-end継続的配信をサポートします。これには、コードが変更されるたびにパイプラインを開始することが含まれます。コードの変更時にパイプラインを開始するために、イベントベースの変更検出とポーリングの 2 つの方法がサポートされています。パイプラインにはイベントベースの変更検出を使用することをお勧めします。

ここで説明する手順を使用して、ポーリングパイプラインからイベントベースの変更検出方法に移行 (更新) します。

パイプラインに推奨されるイベントベースの変更検出方法は、CodeCommit などのパイプラインソースによって決まります。その場合、例えば、ポーリングパイプラインを EventBridge によるイベントベースの変更検出に移行する必要があります。

ポーリングパイプラインを移行する方法

ポーリングパイプラインを移行するには、ポーリングパイプラインを選択した後、推奨されるイベントベースの変更検出方法を選択します。

- アカウント内のポーリングパイプラインの表示 のステップを使用して、ポーリングパイプラインを選択します。
- 表でパイプラインのソースタイプを見つけて、ポーリングパイプラインの移行に使用する実装の手順を選択します。各セクションには、CLI や CloudFormationの使用など、複数の移行方法があります。

パイプラインを推奨される変更検出方法に移行する方法		
パイプラインソース	イベントベースの検出 (推奨方法)	移行手順
AWS CodeCommit	EventBridge (推奨)。	「CodeCommit ソースを使用してポーリングパイプラインを移行する」を参照してください。
Amazon S3	EventBridge およびイベント通知を有効にしたバケット (推奨)。	「イベントに対応した S3 ソースを使用してポーリングパイプラインを移行する」を参照してください。
Amazon S3	EventBridge と証 AWS CloudTrail 跡。	「S3 ソースと CloudTrail 証跡を使用してポーリングパイプラインを移行する」を参照してください。
GitHub (GitHub アプリ経由)	Connections (推奨)	「GitHub (OAuth アプリ経由) ソースアクションのポーリングパイプラインを接続に移行する」を参照してください。
GitHub (OAuth アプリ経由)	ウェブフック	「GitHub (OAuth アプリ経由) ソースアクションのポーリングパイプラインを Webhook に移行する」を参照してください。

Important

GitHub (OAuth アプリ経由) アクションを含むパイプラインなど、該当するパイプラインアクション設定の更新では、ソースアクションの設定内で明示的に `PollForSourceChanges`パ

ラメータを false に設定して、パイプラインのポーリングを停止する必要があります。その結果、例えば、EventBridge ルールを設定すると同時に PollForSourceChanges パラメータを省略することで、イベントベースの変更検出とポーリングの両方を使用するようにパイプラインを誤って設定する可能性があります。これにより、パイプラインが重複して実行される可能性があり、パイプラインはポーリング中のパイプラインの合計数の制限に対してカウントされます。この制限はデフォルトではイベントベースのパイプラインよりもかなり低くなっています。詳細については、「AWS CodePipeline のクォータ」を参照してください。

アカウント内のポーリングパイプラインの表示

最初のステップとして、以下のスクリプトのいずれかを使用して、アカウント内のどのパイプラインに対してポーリングが設定されているかを確認します。これらがイベントベースの変更検出に移行するパイプラインです。

アカウント内のポーリングパイプラインの表示 (スクリプト)

以下の手順でスクリプトを使用して、アカウントでポーリングを使用しているパイプラインを特定します。

1. ターミナルウィンドウを開き、次のいずれかの操作を行います。

   - 以下のコマンドを実行して、PollingPipelinesExtractor.sh という名前の新しいスクリプトを作成します。

     ```
     vi PollingPipelinesExtractor.sh
     ```

   - Python スクリプトを使用するには、以下のコマンドを実行して、PollingPipelinesExtractor.py という名前の新しい Python スクリプトを作成します。

     ```
     vi PollingPipelinesExtractor.py
     ```

2. 以下のコードをコピーして、PollingPipelinesExtractor スクリプトに貼り付けます。次のいずれかを行います。

   - 以下のコードをコピーして、PollingPipelinesExtractor.sh スクリプトに貼り付けます。

     ```
     #!/bin/bash
     ```

```
set +x

POLLING_PIPELINES=()
LAST_EXECUTED_DATES=()
NEXT_TOKEN=null
HAS_NEXT_TOKEN=true
if [[ $# -eq 0 ]] ; then
    echo 'Please provide region name'
    exit 0
fi
REGION=$1


while [ "$HAS_NEXT_TOKEN" != "false" ]; do
    if [ "$NEXT_TOKEN" != "null" ];
        then
            LIST_PIPELINES_RESPONSE=$(aws codepipeline list-pipelines --region
 $REGION --next-token $NEXT_TOKEN)
        else
            LIST_PIPELINES_RESPONSE=$(aws codepipeline list-pipelines --region
 $REGION)
    fi
    LIST_PIPELINES=$(jq -r '.pipelines[].name' <<< "$LIST_PIPELINES_RESPONSE")
    NEXT_TOKEN=$(jq -r '.nextToken' <<< "$LIST_PIPELINES_RESPONSE")
    if [ "$NEXT_TOKEN" == "null" ];
        then
            HAS_NEXT_TOKEN=false
    fi

    for pipline_name in $LIST_PIPELINES
    do
        PIPELINE=$(aws codepipeline get-pipeline --name $pipline_name --region
 $REGION)
        HAS_POLLABLE_ACTIONS=$(jq '.pipeline.stages[].actions[] |
 select(.actionTypeId.category == "Source") | select(.actionTypeId.owner
 == ("ThirdParty","AWS")) | select(.actionTypeId.provider ==
 ("GitHub","S3","CodeCommit")) | select(.configuration.PollForSourceChanges ==
 ("true",null))' <<< "$PIPELINE")
        if [ ! -z "$HAS_POLLABLE_ACTIONS" ];
        then
            POLLING_PIPELINES+=("$pipline_name")
            PIPELINE_EXECUTIONS=$(aws codepipeline list-pipeline-executions --
pipeline-name $pipline_name --region $REGION)
```

```
            LAST_EXECUTION=$(jq -r '.pipelineExecutionSummaries[0]' <<<
 "$PIPELINE_EXECUTIONS")
            if [ "$LAST_EXECUTION" != "null" ];
                then
                    LAST_EXECUTED_TIMESTAMP=$(jq -r '.startTime' <<<
 "$LAST_EXECUTION")
                    LAST_EXECUTED_DATE="$(date -r ${LAST_EXECUTED_TIMESTAMP%.*})"
                else
                    LAST_EXECUTED_DATE="Not executed in last year"
            fi
            LAST_EXECUTED_DATES+=("$LAST_EXECUTED_DATE")
        fi
    done

done

fileName=$REGION-$(date +%s)
printf "| %-30s | %-30s |\n" "Polling Pipeline Name" "Last Executed Time"
printf "| %-30s | %-30s |\n" "_____________________" "___________________"
for i in "${!POLLING_PIPELINES[@]}"; do
  printf "| %-30s | %-30s |\n" "${POLLING_PIPELINES[i]}"
 "${LAST_EXECUTED_DATES[i]}"
  printf "${POLLING_PIPELINES[i]}," >> $fileName.csv
done

printf "\nSaving Polling Pipeline Names to file $fileName.csv."
```

- 以下のコードをコピーして、PollingPipelinesExtractor.py スクリプトに貼り付けます。

```
import boto3
import sys
import time
import math

hasNextToken = True
nextToken = ""
pollablePipelines = []
lastExecutedTimes = []
if len(sys.argv) == 1:
    raise Exception("Please provide region name.")
session = boto3.Session(profile_name='default', region_name=sys.argv[1])
codepipeline = session.client('codepipeline')

def is_pollable_action(action):
```

```
    actionTypeId = action['actionTypeId']
    configuration = action['configuration']
    return actionTypeId['owner'] in {"AWS", "ThirdParty"}
 and actionTypeId['provider'] in {"GitHub", "CodeCommit",
 "S3"} and ('PollForSourceChanges' not in configuration or
 configuration['PollForSourceChanges'] == 'true')

def has_pollable_actions(pipeline):
    hasPollableAction = False
    pipelineDefinition = codepipeline.get_pipeline(name=pipeline['name'])
['pipeline']
    for action in pipelineDefinition['stages'][0]['actions']:
        hasPollableAction = is_pollable_action(action)
        if hasPollableAction:
            break
    return hasPollableAction

def get_last_executed_time(pipelineName):

 pipelineExecutions=codepipeline.list_pipeline_executions(pipelineName=pipelineName)
['pipelineExecutionSummaries']
    if pipelineExecutions:
        return pipelineExecutions[0]['startTime'].strftime("%A %m/%d/%Y, %H:%M:
%S")
    else:
        return "Not executed in last year"

while hasNextToken:
    if nextToken=="":
        list_pipelines_response = codepipeline.list_pipelines()
    else:
        list_pipelines_response =
 codepipeline.list_pipelines(nextToken=nextToken)
    if 'nextToken' in list_pipelines_response:
        nextToken = list_pipelines_response['nextToken']
    else:
        hasNextToken= False
    for pipeline in list_pipelines_response['pipelines']:
        if has_pollable_actions(pipeline):
            pollablePipelines.append(pipeline['name'])
            lastExecutedTimes.append(get_last_executed_time(pipeline['name']))

fileName="{region}-
{timeNow}.csv".format(region=sys.argv[1],timeNow=math.trunc(time.time()))
```

```
file = open(fileName, 'w')

print ("{:<30} {:<30} {:<30}".format('Polling Pipeline Name', '|','Last Executed
 Time'))
print ("{:<30} {:<30} {:<30}".format('_____________________',
 '|','___________________'))
for i in range(len(pollablePipelines)):
    print("{:<30} {:<30} {:<30}".format(pollablePipelines[i], '|',
 lastExecutedTimes[i]))
    file.write("{pipeline},".format(pipeline=pollablePipelines[i]))
file.close()
print("\nSaving Polling Pipeline Names to file
 {fileName}".format(fileName=fileName))
```

3. パイプラインがあるリージョンごとに、そのリージョンに対してこのスクリプトを実行する必要があります。スクリプトを実行するには、以下のいずれかの操作を行います。

   - 以下のコマンドを実行して、PollingPipelinesExtractor.sh という名前のスクリプトを実行します。この例で、リージョンは us-west-2 です。

     ```
     ./PollingPipelinesExtractor.sh us-west-2
     ```

   - Python スクリプトを使用する場合は、以下のコマンドを実行して、PollingPipelinesExtractor.py という名前の Python スクリプトを実行します。この例で、リージョンは us-west-2 です。

     ```
     python3 PollingPipelinesExtractor.py us-west-2
     ```

   スクリプトからの以下のサンプル出力では、リージョン us-west-2 からポーリングパイプラインのリストが返され、各パイプラインの最後の実行時間が表示されています。

```
 % ./pollingPipelineExtractor.sh us-west-2

| Polling Pipeline Name | Last Executed Time           |
| _____________________ | ___________________          |
| myCodeBuildPipeline   | Wed Mar 8 09:35:49 PST 2023  |
| myCodeCommitPipeline  | Mon Apr 24 22:32:32 PDT 2023 |
| TestPipeline          | Not executed in last year    |
```

```
Saving list of polling pipeline names to us-west-2-1682496174.csv...%
```

スクリプトの出力を分析し、リスト内のパイプラインごとに、ポーリングソースを推奨されるイベントベースの変更検出方法に更新します。

> **Note**
>
> ポーリングパイプラインは、PollForSourceChanges パラメータのパイプラインのアクション設定によって決まります。パイプラインソース設定で PollForSourceChangesパラメータが省略されている場合、CodePipeline はデフォルトでソースの変更についてリポジトリをポーリングします。この動作は、PollForSourceChanges が含まれており、true に設定されている場合と同じです。詳細については、Amazon S3 ソースアクションリファレンス で「Amazon S3 のソースアクションの設定パラメータ」など、「パイプラインのソースアクションの設定パラメータ」を参照してください。

このスクリプトは、アカウント内のポーリングパイプラインのリストを含む .csv ファイルも生成し、その .csv ファイルを現在の作業フォルダに保存します。

CodeCommit ソースを使用してポーリングパイプラインを移行する

EventBridge を使用してポーリングパイプラインを移行して、CodeCommit ソースリポジトリまたは Amazon S3 ソースバケットの変更を検出できるようにすることができます。

CodeCommit - CodeCommit ソースを使用するパイプラインでは、変更検出が EventBridge を通じて自動化されるようにパイプラインを修正します。以下のいずれかの方法で移行を実装します。

- コンソール: ポーリングパイプラインを移行する (CodeCommit または Amazon S3 ソース) (コンソール)
- CLI: ポーリングパイプラインを移行する (CodeCommit ソース) (CLI)
- CloudFormation: ポーリングパイプラインの移行 (CodeCommit ソース) (CloudFormation テンプレート)

ポーリングパイプラインを移行する (CodeCommit または Amazon S3 ソース) (コンソール)

CodePipeline コンソールで、EventBridge を使用して CodeCommit ソースリポジトリまたは Amazon S3 ソースバケットの変更を検出するように、パイプラインを更新できます。

> **Note**
>
> コンソールを使用して CodeCommit ソースリポジトリや Amazon S3 ソースバケットを含むパイプラインを編集すると、ルールと IAM ロールが自動的に作成されます。を使用してパイプライン AWS CLI を編集する場合は、EventBridge ルールと IAM ロールを自分で作成する必要があります。詳細については、「CodeCommit ソースアクションと EventBridge」を参照してください。

定期的なチェックを使用しているパイプラインを編集するには、これらの手順を使用します。パイプラインを作成する場合は、「パイプライン、ステージ、アクションを作成する」を参照してください。

パイプラインソースステージを編集するには

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

   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。
2. [名前] で、編集するパイプラインの名前を選択します。これにより、パイプラインの詳細ビューが開いて、パイプラインの各ステージの各アクションの状態などがわかります。
3. パイプライン詳細ページで、[編集] を選択します。
4. [Edit (編集)] ステージで、ソースアクションの編集アイコンを選択します。
5. [Change Detection Options (変更検出オプション)] を展開し、[Use CloudWatch Events to automatically start my pipeline when a change occurs (recommended)] を選択します。

   このパイプラインに対して作成される EventBridge ルールを示すメッセージが表示されます。[更新] を選択します。

   Amazon S3 ソースを含むパイプラインを更新する場合は、以下のメッセージが表示されます。[更新] を選択します。

6. パイプラインの編集が終わったら、[パイプラインの変更を保存] を選択して概要ページに戻ります。

   パイプラインに対して作成される EventBridge ルールの名前を示すメッセージが表示されます。[保存して続行] を選択します。

7. アクションをテストするには、 を使用して変更をリリース AWS CLI し、パイプラインのソースステージで指定されたソースに変更をコミットします。

ポーリングパイプラインを移行する (CodeCommit ソース) (CLI)

EventBridge ルールを使用してパイプラインを開始するためにポーリング (定期的なチェック) を使用しているパイプラインを編集するには、以下の手順に従います。パイプラインを作成する場合は、「パイプライン、ステージ、アクションを作成する」を参照してください。

CodeCommit でイベント駆動型パイプラインを構築するには、パイプラインの `PollForSourceChanges` パラメータを編集してから、以下のリソースを作成します。

- EventBridge イベント
- このイベントによるパイプラインの開始を許可する IAM ロール

パイプラインの PollForSourceChanges パラメータを編集するには

> **Important**
>
> このメソッドを使用してパイプラインを作成すると、`PollForSourceChanges` パラメータはデフォルトで true になります (ただし、明示的に false に設定した場合は除きます)。イベントベースの変更検出を追加する場合は、このパラメータを出力に追加する必要があります。ポーリングを無効にするには、このパラメータを false に設定します。そうしないと、1 つのソース変更に対してパイプラインが 2 回起動されます。詳細については、「`PollForSourceChanges` パラメータの有効な設定」を参照してください

1. get-pipeline コマンドを実行して、パイプライン構造を JSON ファイルにコピーします。例えば、`MyFirstPipeline` という名前のパイプラインに対して、以下のコマンドを実行します。

```
aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
```

このコマンドは何も返しませんが、作成したファイルは、コマンドを実行したディレクトリにあります。

2. 任意のプレーンテキストエディタで JSON ファイルを開き、以下に示しているように、PollForSourceChanges パラメータを false に変更してソースステージを編集します。

   この変更を行う理由 このパラメータを false に変更すると、定期的チェックがオフになるため、イベントベースの変更検出のみ使用することができます。

```
"configuration": {
    "PollForSourceChanges": "false",
    "BranchName": "main",
    "RepositoryName": "MyTestRepo"
},
```

3. get-pipeline コマンドを使用して取得したパイプライン構造を使用している場合、JSON ファイルから metadata 行を削除します。それ以外の場合は、update-pipeline コマンドで使用することはできません。"metadata": { } 行と、"created"、"pipelineARN"、"updated" フィールドを削除します。

   例えば、構造から以下の行を削除します。

```
"metadata": {
    "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
    "created": "date",
    "updated": "date"
},
```

   ファイルを保存します。

4. 変更を適用するには、パイプライン JSON ファイルを指定して、update-pipeline コマンドを実行します。

> **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 codepipeline update-pipeline --cli-input-json file://pipeline.json
```

このコマンドは、編集したパイプラインの構造全体を返します。

> **Note**
>
> update-pipeline コマンドは、パイプラインを停止します。update-pipeline コマンドを実行したときにパイプラインによりリビジョンが実行されている場合、その実行は停止します。更新されたパイプラインによりそのリビジョンを実行するには、パイプラインを手動で開始する必要があります。パイプラインを手動で開始するには **start-pipeline-execution** コマンドを使用します。

CodeCommit をイベントソースとして、CodePipeline をターゲットとして EventBridge ルールを作成するには

1. EventBridge が CodePipeline を使用してルールを呼び出すためのアクセス許可を追加します。詳細については、 デベロッパーガイドの [Amazon EventBridge のリソースベースのポリシーを使用する] を参照してください。

   a. 次のサンプルを使用して、EventBridge にサービスロールの引き受けを許可する信頼ポリシーを作成します。信頼ポリシーに `trustpolicyforEB.json` と名前を付けます。

      JSON

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

   b. 次のコマンドを使用して、`Role-for-MyRule` ロールを作成し、信頼ポリシーをアタッチします。

```
aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document
 file://trustpolicyforEB.json
```

c. 次のサンプルに示すように、MyFirstPipeline というパイプラインに対して、アクセス権限ポリシー JSON を作成します。アクセス権限ポリシーに permissionspolicyforEB.json と名前を付けます。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:StartPipelineExecution"
            ],
            "Resource": [
                "arn:aws:codepipeline:us-
west-2:111122223333:MyFirstPipeline"
            ]
        }
    ]
}
```

d. 次のコマンドを使用して、Role-for-MyRule ロールに CodePipeline-Permissions-Policy-for-EB アクセス権限ポリシーをアタッチします。

この変更を行う理由 ロールにこのポリシーを追加すると、EventBridge に対するアクセス許可が作成されます。

```
aws iam put-role-policy --role-name Role-for-MyRule --policy-name CodePipeline-
Permissions-Policy-For-EB --policy-document file://permissionspolicyforEB.json
```

2. put-rule コマンドを呼び出し、--name、--event-pattern、--role-arn パラメータを含めます。

この変更を行う理由 このコマンドでは、 CloudFormation でイベントを作成することができます。

次のサンプルコマンドは、MyCodeCommitRepoRule というルールを作成します。

```
aws events put-rule --name "MyCodeCommitRepoRule" --event-pattern "{\"source\":
[\"aws.codecommit\"],\"detail-type\":[\"CodeCommit Repository State Change\"],
\"resources\":[\"repository-ARN\"],\"detail\":{\"referenceType\":[\"branch\"],
\"referenceName\":[\"main\"]}}" --role-arn "arn:aws:iam::ACCOUNT_ID:role/Role-for-
MyRule"
```

3. CodePipeline をターゲットとして追加するには、put-targets コマンドを呼び出し、次のパラメータを含めます。

   - --rule パラメータは、put-rule を使用して作成した rule_name で使用されます。
   - --targets パラメータは、ターゲットリストのリスト Id とターゲットパイプラインの ARN で使用されます。

   次のサンプルコマンドでは、MyCodeCommitRepoRule と呼ばれるルールに対して指定し、ターゲット Id は 1 番で構成されています。これは、ルールのターゲットのリストが何であるかを示し、この場合は ターゲット 1 です。このサンプルコマンドでは、パイプラインのサンプルの ARN も指定されます。パイプラインは、リポジトリ内に変更が加えられると開始します。

```
aws events put-targets --rule MyCodeCommitRepoRule --targets
 Id=1,Arn=arn:aws:codepipeline:us-west-2:80398EXAMPLE:TestPipeline
```

4. (オプション) 特定のイメージ ID のソースオーバーライドを使用して入力トランスフォーマーを設定するには、CLI コマンドで次の JSON を使用します。次の例では、以下のオーバーライドを設定します。

   - actionName は (この例では Source)、ソースイベントから派生したものではなく、パイプラインの作成時に定義された動的な値です。
   - revisionType は (この例では COMMIT_ID)、ソースイベントから派生したものではなく、パイプラインの作成時に定義された動的な値です。
   - revisionValue は (この例では <*revisionValue*>)、ソースイベント変数から派生しています。

```
{
    "Rule": "my-rule",
    "Targets": [
```

```
        {
            "Id": "MyTargetId",
            "Arn": "pipeline-ARN",
            "InputTransformer": {
                "sourceRevisions": {
                    "actionName": "Source",
                    "revisionType": "COMMIT_ID",
                    "revisionValue": "<revisionValue>"
                },
                "variables": [
                    {
                        "name": "Branch_Name",
                        "value": "value"
                    }
                ]
            }
        }
    ]
}
```

ポーリングパイプラインの移行 (CodeCommit ソース) (CloudFormation テンプレート)

を使用してイベント駆動型パイプラインを構築するには AWS CodeCommit、パイプラインの `PollForSourceChanges`パラメータを編集し、テンプレートに次のリソースを追加します。

- EventBridge ルール
- EventBridge ルールの IAM ロール

CloudFormation を使用してパイプラインを作成および管理する場合、テンプレートには次のようなコンテンツが含まれます。

> **Note**
>
> `PollForSourceChanges` と呼ばれるソースステージの `Configuration` プロパティ。プロパティがテンプレートに含まれていない場合、`PollForSourceChanges` はデフォルトで `true` に設定されます。

YAML

```
Resources:
  AppPipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      Name: codecommit-polling-pipeline
      RoleArn:
        !GetAtt CodePipelineServiceRole.Arn
      Stages:
        -
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: AWS
                Version: 1
                Provider: CodeCommit
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                BranchName: !Ref BranchName
                RepositoryName: !Ref RepositoryName
                PollForSourceChanges: true
              RunOrder: 1
```

JSON

```
"Stages": [
    {
        "Name": "Source",
  "Actions": [{
      "Name": "SourceAction",
      "ActionTypeId": {
    "Category": "Source",
    "Owner": "AWS",
    "Version": 1,
    "Provider": "CodeCommit"
      },
      "OutputArtifacts": [{
          "Name": "SourceOutput"
```

```
      }],
      "Configuration": {
          "BranchName": {
        "Ref": "BranchName"
    },
    "RepositoryName": {
        "Ref": "RepositoryName"
    },
    "PollForSourceChanges": true
            },
            "RunOrder": 1
        }]
    },
```

パイプライン CloudFormation テンプレートを更新して EventBridge ルールを作成するには

1. テンプレートの でResources、 AWS::IAM::Role CloudFormation リソースを使用して、イベントがパイプラインを開始できるようにする IAM ロールを設定します。このエントリによって、2 つのポリシーを使用するロールが作成されます。

   - 最初のポリシーでは、ロールを引き受けることを許可します。
   - 2 つめのポリシーでは、パイプラインを開始するアクセス権限が付与されます。

   この変更を行う理由 AWS::IAM::Role リソースを追加すると CloudFormation 、 は EventBridge のアクセス許可を作成できます。このリソースは CloudFormation スタックに追加されます。

   YAML

   ```
   EventRole:
     Type: AWS::IAM::Role
     Properties:
       AssumeRolePolicyDocument:
         Version: 2012-10-17
         Statement:
           -
             Effect: Allow
             Principal:
               Service:
                 - events.amazonaws.com
   ```

```
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref
 'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]
```

JSON

```
"EventRole": {
  "Type": "AWS::IAM::Role",
  "Properties": {
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": [
              "events.amazonaws.com"
            ]
          },
          "Action": "sts:AssumeRole"
        }
      ]
    },
    "Path": "/",
    "Policies": [
      {
        "PolicyName": "eb-pipeline-execution",
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Effect": "Allow",
              "Action": "codepipeline:StartPipelineExecution",
```

```
              "Resource": {
                "Fn::Join": [
                  "",
                  [
                    "arn:aws:codepipeline:",
                    {
                      "Ref": "AWS::Region"
                    },
                    ":",
                    {
                      "Ref": "AWS::AccountId"
                    },
                    ":",
                    {
                      "Ref": "AppPipeline"
                    }
                  ]

...
```

2. テンプレートの でResources、 AWS::Events::Rule CloudFormation リソースを使用して EventBridge ルールを追加します。このイベントパターンは、リポジトリへの変更のプッシュをモニタリングするイベントを作成します。EventBridge でリポジトリの状態の変更が検出されると、ルールはターゲットパイプラインで StartPipelineExecution を呼び出します。

   この変更を行う理由 AWS::Events::Rule リソースを追加すると CloudFormation 、 は イベントを作成できます。このリソースは CloudFormation スタックに追加されます。

   YAML

```
 EventRule:
   Type: AWS::Events::Rule
   Properties:
     EventPattern:
       source:
         - aws.codecommit
       detail-type:
         - 'CodeCommit Repository State Change'
       resources:
         - !Join [ '', [ 'arn:aws:codecommit:', !Ref 'AWS::Region', ':', !Ref
'AWS::AccountId', ':', !Ref RepositoryName ] ]
       detail:
         event:
```

```
            - referenceCreated
            - referenceUpdated
          referenceType:
            - branch
          referenceName:
            - main
      Targets:
        -
          Arn:
            !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref AppPipeline ] ]
          RoleArn: !GetAtt EventRole.Arn
          Id: codepipeline-AppPipeline
```

JSON

```
"EventRule": {
  "Type": "AWS::Events::Rule",
  "Properties": {
    "EventPattern": {
      "source": [
        "aws.codecommit"
      ],
      "detail-type": [
        "CodeCommit Repository State Change"
      ],
      "resources": [
        {
          "Fn::Join": [
            "",
            [
              "arn:aws:codecommit:",
              {
                "Ref": "AWS::Region"
              },
              ":",
              {
                "Ref": "AWS::AccountId"
              },
              ":",
              {
                "Ref": "RepositoryName"
```

```
            }
          ]
        ]
      }
    ],
    "detail": {
      "event": [
        "referenceCreated",
        "referenceUpdated"
      ],
      "referenceType": [
        "branch"
      ],
      "referenceName": [
        "main"
      ]
    }
  },
  "Targets": [
    {
      "Arn": {
        "Fn::Join": [
          "",
          [
            "arn:aws:codepipeline:",
            {
              "Ref": "AWS::Region"
            },
            ":",
            {
              "Ref": "AWS::AccountId"
            },
            ":",
            {
              "Ref": "AppPipeline"
            }
          ]
        ]
      },
      "RoleArn": {
        "Fn::GetAtt": [
          "EventRole",
          "Arn"
        ]
```

```
        },
        "Id": "codepipeline-AppPipeline"
      }
    ]
  }
},
```

3. (オプション) 特定のイメージ ID のソースオーバーライドを使用して入力トランスフォーマーを設定するには、次の YAML スニペットを使用します。次の例では、以下のオーバーライドを設定します。

   - `actionName` は (この例では `Source`)、ソースイベントから派生したものではなく、パイプラインの作成時に定義された動的な値です。
   - `revisionType` は (この例では `COMMIT_ID`)、ソースイベントから派生したものではなく、パイプラインの作成時に定義された動的な値です。
   - `revisionValue` は (この例では <*revisionValue*>)、ソースイベント変数から派生しています。
   - `BranchName` および `Value` の出力変数が指定されます。

```
Rule: my-rule
Targets:
- Id: MyTargetId
  Arn: pipeline-ARN
  InputTransformer:
    sourceRevisions:
      actionName: Source
      revisionType: COMMIT_ID
      revisionValue: <revisionValue>
    variables:
    - name: BranchName
      value: value
```

4. 更新されたテンプレートをローカルコンピュータに保存し、 CloudFormation コンソールを開きます。
5. スタックを選択し、[既存スタックの変更セットの作成] を選択します。
6. テンプレートをアップロードし、 CloudFormationに示された変更を表示します。これらがスタックに加えられる変更です。新しいリソースがリストに表示されています。
7. [実行] を選択してください。

パイプラインの PollForSourceChanges パラメータを編集するには

> **Important**
>
> 多くの場合、パイプラインの作成時に PollForSourceChanges パラメータはデフォルトで true になります。イベントベースの変更検出を追加する場合は、このパラメータを出力に追加する必要があります。ポーリングを無効にするには、このパラメータを false に設定します。そうしないと、1 つのソース変更に対してパイプラインが 2 回起動されます。詳細については、「PollForSourceChanges パラメータの有効な設定」を参照してください

- テンプレートで、PollForSourceChanges を false に変更します。パイプライン定義に PollForSourceChanges が含まれていなかった場合は、追加して false に設定します。

  この変更を行う理由 このパラメータを false に変更すると、定期的チェックがオフになるため、イベントベースの変更検出のみ使用することができます。

  YAML

```
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: AWS
                Version: 1
                Provider: CodeCommit
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                BranchName: !Ref BranchName
                RepositoryName: !Ref RepositoryName
                PollForSourceChanges: false
              RunOrder: 1
```

  JSON

```
{
  "Name": "Source",
  "Actions": [
```

```
    {
      "Name": "SourceAction",
      "ActionTypeId": {
        "Category": "Source",
        "Owner": "AWS",
        "Version": 1,
        "Provider": "CodeCommit"
      },
      "OutputArtifacts": [
        {
          "Name": "SourceOutput"
        }
      ],
      "Configuration": {
        "BranchName": {
          "Ref": "BranchName"
        },
        "RepositoryName": {
          "Ref": "RepositoryName"
        },
        "PollForSourceChanges": false
      },
      "RunOrder": 1
    }
  ]
},
```

Example

これらのリソースを で作成すると CloudFormation、リポジトリ内のファイルが作成または更新されたときにパイプラインがトリガーされます。以下に示しているのは、最終的なテンプレートスニペットです。

YAML

```
Resources:
  EventRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
```

```
        Statement:
          -
            Effect: Allow
            Principal:
              Service:
                - events.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region',
':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]
 EventRule:
   Type: AWS::Events::Rule
   Properties:
     EventPattern:
       source:
         - aws.codecommit
       detail-type:
         - 'CodeCommit Repository State Change'
       resources:
         - !Join [ '', [ 'arn:aws:codecommit:', !Ref 'AWS::Region', ':', !Ref
'AWS::AccountId', ':', !Ref RepositoryName ] ]
       detail:
         event:
           - referenceCreated
           - referenceUpdated
         referenceType:
           - branch
         referenceName:
           - main
     Targets:
       -
         Arn:
           !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
'AWS::AccountId', ':', !Ref AppPipeline ] ]
         RoleArn: !GetAtt EventRole.Arn
```

```
          Id: codepipeline-AppPipeline
  AppPipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      Name: codecommit-events-pipeline
      RoleArn:
        !GetAtt CodePipelineServiceRole.Arn
      Stages:
        -
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: AWS
                Version: 1
                Provider: CodeCommit
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                BranchName: !Ref BranchName
                RepositoryName: !Ref RepositoryName
                PollForSourceChanges: false
              RunOrder: 1


...
```

JSON

```
    "Resources": {

...

        "EventRole": {
            "Type": "AWS::IAM::Role",
            "Properties": {
                "AssumeRolePolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
```

```
                        "Principal": {
                            "Service": [
                                "events.amazonaws.com"
                            ]
                        },
                        "Action": "sts:AssumeRole"
                    }
                ]
            },
            "Path": "/",
            "Policies": [
                {
                    "PolicyName": "eb-pipeline-execution",
                    "PolicyDocument": {
                        "Version": "2012-10-17",
                        "Statement": [
                            {
                                "Effect": "Allow",
                                "Action": "codepipeline:StartPipelineExecution",
                                "Resource": {
                                    "Fn::Join": [
                                        "",
                                        [
                                            "arn:aws:codepipeline:",
                                            {
                                                "Ref": "AWS::Region"
                                            },
                                            ":",
                                            {
                                                "Ref": "AWS::AccountId"
                                            },
                                            ":",
                                            {
                                                "Ref": "AppPipeline"
                                            }
                                        ]
                                    ]
                                }
                            }
                        ]
                    }
                }
            ]
        }
```

```
        },
        "EventRule": {
            "Type": "AWS::Events::Rule",
            "Properties": {
                "EventPattern": {
                    "source": [
                        "aws.codecommit"
                    ],
                    "detail-type": [
                        "CodeCommit Repository State Change"
                    ],
                    "resources": [
                        {
                            "Fn::Join": [
                                "",
                                [
                                    "arn:aws:codecommit:",
                                    {
                                        "Ref": "AWS::Region"
                                    },
                                    ":",
                                    {
                                        "Ref": "AWS::AccountId"
                                    },
                                    ":",
                                    {
                                        "Ref": "RepositoryName"
                                    }
                                ]
                            ]
                        }
                    ],
                    "detail": {
                        "event": [
                            "referenceCreated",
                            "referenceUpdated"
                        ],
                        "referenceType": [
                            "branch"
                        ],
                        "referenceName": [
                            "main"
                        ]
                    }
```

```
            },
            "Targets": [
                {
                    "Arn": {
                        "Fn::Join": [
                            "",
                            [
                                "arn:aws:codepipeline:",
                                {
                                    "Ref": "AWS::Region"
                                },
                                ":",
                                {
                                    "Ref": "AWS::AccountId"
                                },
                                ":",
                                {
                                    "Ref": "AppPipeline"
                                }
                            ]
                        ]
                    },
                    "RoleArn": {
                        "Fn::GetAtt": [
                            "EventRole",
                            "Arn"
                        ]
                    },
                    "Id": "codepipeline-AppPipeline"
                }
            ]
        }
    },
    "AppPipeline": {
        "Type": "AWS::CodePipeline::Pipeline",
        "Properties": {
            "Name": "codecommit-events-pipeline",
            "RoleArn": {
                "Fn::GetAtt": [
                    "CodePipelineServiceRole",
                    "Arn"
                ]
            },
            "Stages": [
```

```
                    {
                        "Name": "Source",
                        "Actions": [
                            {
                                "Name": "SourceAction",
                                "ActionTypeId": {
                                    "Category": "Source",
                                    "Owner": "AWS",
                                    "Version": 1,
                                    "Provider": "CodeCommit"
                                },
                                "OutputArtifacts": [
                                    {
                                        "Name": "SourceOutput"
                                    }
                                ],
                                "Configuration": {
                                    "BranchName": {
                                        "Ref": "BranchName"
                                    },
                                    "RepositoryName": {
                                        "Ref": "RepositoryName"
                                    },
                                    "PollForSourceChanges": false
                                },
                                "RunOrder": 1
                            }
                        ]
                    },

...
```

イベントに対応した S3 ソースを使用してポーリングパイプラインを移行する

Amazon S3 ソースを含むパイプラインでは、変更検出が EventBridge を通じて自動化され、イベント通知が有効になっているソースバケットを使用するように、パイプラインを修正します。これは、CLI または を使用してパイプライン CloudFormation を移行する場合に推奨される方法です。

> **Note**
>
> この方法では、イベント通知が有効になっているバケットを使用するため、別途 CloudTrail 証跡を作成する必要はありません。コンソールを使用する場合は、イベントルールと CloudTrail 証跡が自動的に設定されます。これらの手順については、「S3 ソースと CloudTrail 証跡を使用してポーリングパイプラインを移行する」を参照してください。

- CLI: S3 ソースと CloudTrail 証跡を使用してポーリングパイプラインを移行する (CLI)
- CloudFormation: S3 ソースと CloudTrail 証跡 (CloudFormation テンプレート) を使用してポーリングパイプラインを移行する

イベントに対応した S3 ソースを使用してポーリングパイプラインを移行する (CLI)

ポーリング (定期的なチェック) を使用しているパイプラインを、EventBridge のイベントを使用するように編集するには、次の手順に従います。パイプラインを作成する場合は、「パイプライン、ステージ、アクションを作成する」を参照してください。

Amazon S3 でイベント駆動型パイプラインを構築するには、パイプラインの `PollForSourceChanges` パラメータを編集してから、以下のリソースを作成します。

- EventBridge イベントバス
- EventBridge イベントによるパイプラインの開始を許可する IAM ロール

Amazon S3 をイベントソース、CodePipeline をターゲットとする EventBridge ルールを作成し、アクセス許可ポリシーを適用するには

1. EventBridge が CodePipeline を使用してルールを呼び出すためのアクセス許可を付与します。詳細については、 デベロッパーガイドの [Amazon EventBridge のリソースベースのポリシーを使用する] を参照してください。

   a. 次のサンプルを使用して、EventBridge にサービスロールの引き受けを許可する信頼ポリシーを作成します。このスクリプトに `trustpolicyforEB.json` という名前を付けます。

JSON

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

b. 次のコマンドを使用して、Role-for-MyRule ロールを作成し、信頼ポリシーをアタッチします。

この変更を行う理由 ロールにこの信頼ポリシーを追加すると、EventBridge に対するアクセス許可が作成されます。

```
aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document
 file://trustpolicyforEB.json
```

c. 次に示すように、MyFirstPipeline という名前のパイプラインに対してアクセス許可ポリシー JSON を作成します。アクセス権限ポリシーに permissionspolicyforEB.json と名前を付けます。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:StartPipelineExecution"
            ],
            "Resource": [
```

```
                "arn:aws:codepipeline:us-
west-2:111122223333:MyFirstPipeline"
            ]
        }
    ]
}
```

d. 次のコマンドを実行して、作成した Role-for-MyRule ロールに新しい CodePipeline-Permissions-Policy-for-EB アクセス権限ポリシーをアタッチします。

```
aws iam put-role-policy --role-name Role-for-MyRule --policy-name CodePipeline-
Permissions-Policy-For-EB --policy-document file://permissionspolicyforEB.json
```

2. put-rule コマンドを呼び出し、--name、--event-pattern、--role-arn パラメータを含めます。

   次のサンプルコマンドでは、EnabledS3SourceRule という名前のルールが作成されます。

```
aws events put-rule --name "EnabledS3SourceRule" --event-pattern "{\"source\":
[\"aws.s3\"],\"detail-type\":[\"Object Created\"],\"detail\":{\"bucket\":{\"name\":
[\"amzn-s3-demo-source-bucket\"]}}}" --role-arn "arn:aws:iam::ACCOUNT_ID:role/Role-
for-MyRule"
```

3. CodePipeline をターゲットとして追加するには、put-targets コマンドを呼び出し、--rule および --targets パラメータを含めます。

   次のコマンドでは、EnabledS3SourceRule という名前のルールに対して指定し、ターゲット Id は 1 番で構成されています。これは、ルールのターゲットのリストが何であるかを示し、この場合は ターゲット 1 です。このコマンドでは、パイプラインのサンプルの ARN も指定されます。パイプラインは、リポジトリ内に変更が加えられると開始します。

```
aws events put-targets --rule EnabledS3SourceRule --targets Id=codepipeline-
AppPipeline,Arn=arn:aws:codepipeline:us-west-2:80398EXAMPLE:TestPipeline
```

パイプラインの PollForSourceChanges パラメータを編集するには

> **Important**
>
> このメソッドを使用してパイプラインを作成すると、PollForSourceChanges パラメータはデフォルトで true になります (ただし、明示的に false に設定した場合は除きます)。

イベントベースの変更検出を追加する場合は、このパラメータを出力に追加する必要があります。ポーリングを無効にするには、このパラメータを false に設定します。そうしないと、1 つのソース変更に対してパイプラインが 2 回起動されます。詳細については、「PollForSourceChanges パラメータの有効な設定」を参照してください

1. get-pipeline コマンドを実行して、パイプライン構造を JSON ファイルにコピーします。例えば、MyFirstPipeline という名前のパイプラインに対して、以下のコマンドを実行します。

```
aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
```

このコマンドは何も返しませんが、作成したファイルは、コマンドを実行したディレクトリにあります。

2. この例に示すように、プレーンテキストエディタで JSON ファイルを開き、PollForSourceChanges という名前のバケットの amzn-s3-demo-source-bucket パラメータを false に変更してソースステージを編集します。

この変更を行う理由 このパラメータを false に設定すると、定期的チェックがオフになるため、イベントベースの変更検出のみ使用することができます。

```
"configuration": {
    "S3Bucket": "amzn-s3-demo-source-bucket",
    "PollForSourceChanges": "false",
    "S3ObjectKey": "index.zip"
},
```

3. get-pipeline コマンドを使用して取得したパイプライン構造を使用している場合、JSON ファイルから metadata 行を削除する必要があります。それ以外の場合は、update-pipeline コマンドで使用することはできません。"metadata": { } 行と、"created"、"pipelineARN"、"updated" フィールドを削除します。

例えば、構造から以下の行を削除します。

```
"metadata": {
    "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
    "created": "date",
    "updated": "date"
},
```

ファイルを保存します。

4. 変更を適用するには、パイプライン JSON ファイルを指定して、update-pipeline コマンドを実行します。

> **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 codepipeline update-pipeline --cli-input-json file://pipeline.json
```

このコマンドは、編集したパイプラインの構造全体を返します。

> **Note**
>
> update-pipeline コマンドは、パイプラインを停止します。update-pipeline コマンドを実行したときにパイプラインによりリビジョンが実行されている場合、その実行は停止します。更新されたパイプラインによりそのリビジョンを実行するには、パイプラインを手動で開始する必要があります。パイプラインを手動で開始するには start-pipeline-execution コマンドを使用します。

イベントに対して S3 ソースを有効にしたポーリングパイプラインを移行する (CloudFormation テンプレート)

この手順は、ソースバケットでイベントが有効になっているパイプライン用です。

以下の手順を使用して、Amazon S3 ソースを含むパイプラインを、ポーリングからイベントベースの変更検出に編集します。

Amazon S3 でイベント駆動型パイプラインを構築するには、パイプラインの `PollForSourceChanges` パラメータを編集してから、以下のリソースをテンプレートに追加します。

- このイベントによるパイプラインの開始を許可する EventBridge ルールと IAM ロール。

CloudFormation を使用してパイプラインを作成および管理する場合、テンプレートには次のようなコンテンツが含まれます。

> **Note**
>
> PollForSourceChanges と呼ばれるソースステージの Configuration プロパティ。テンプレートにプロパティが含まれていない場合、PollForSourceChanges はデフォルトで true に設定されます。

YAML

```
 AppPipeline:
   Type: AWS::CodePipeline::Pipeline
   Properties:
     RoleArn: !GetAtt CodePipelineServiceRole.Arn
     Stages:
       -
         Name: Source
         Actions:
           -
             Name: SourceAction
             ActionTypeId:
               Category: Source
               Owner: AWS
               Version: 1
               Provider: S3
             OutputArtifacts:
               -
                 Name: SourceOutput
             Configuration:
               S3Bucket: !Ref SourceBucket
               S3ObjectKey: !Ref S3SourceObjectKey
               PollForSourceChanges: true
             RunOrder: 1


...
```

JSON

```
        "AppPipeline": {
```

```
            "Type": "AWS::CodePipeline::Pipeline",
            "Properties": {
                "RoleArn": {
                    "Fn::GetAtt": ["CodePipelineServiceRole", "Arn"]
                },
                "Stages": [
                    {
                        "Name": "Source",
                        "Actions": [
                            {
                                "Name": "SourceAction",
                                "ActionTypeId": {
                                    "Category": "Source",
                                    "Owner": "AWS",
                                    "Version": 1,
                                    "Provider": "S3"
                                },
                                "OutputArtifacts": [
                                    {
                                        "Name": "SourceOutput"
                                    }
                                ],
                                "Configuration": {
                                    "S3Bucket": {
                                        "Ref": "SourceBucket"
                                    },
                                    "S3ObjectKey": {
                                        "Ref": "SourceObjectKey"
                                    },
                                    "PollForSourceChanges": true
                                },
                                "RunOrder": 1
                            }
                        ]
                    },


...
```

Amazon S3 をイベントソース、CodePipeline をターゲットとする EventBridge ルールを作成し、アクセス許可ポリシーを適用するには

1. テンプレートの でResources、 AWS::IAM::Role CloudFormation リソースを使用して、イベントがパイプラインを開始できるようにする IAM ロールを設定します。このエントリによって、2 つのポリシーを使用するロールが作成されます。

   - 最初のポリシーでは、ロールを引き受けることを許可します。
   - 2 つめのポリシーでは、パイプラインを開始するアクセス権限が付与されます。

   この変更を行う理由 AWS::IAM::Role リソースを追加する CloudFormation と、 は EventBridge のアクセス許可を作成できます。このリソースは CloudFormation スタックに追加されます。

   YAML

```
EventRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: 2012-10-17
      Statement:
        -
          Effect: Allow
          Principal:
            Service:
              - events.amazonaws.com
          Action: sts:AssumeRole
    Path: /
    Policies:
      -
        PolicyName: eb-pipeline-execution
        PolicyDocument:
          Version: 2012-10-17
          Statement:
            -
              Effect: Allow
              Action: codepipeline:StartPipelineExecution
              Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref
'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]
```

```
...
```

JSON

```
"EventRole": {
  "Type": "AWS::IAM::Role",
  "Properties": {
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": [
              "events.amazonaws.com"
            ]
          },
          "Action": "sts:AssumeRole"
        }
      ]
    },
    "Path": "/",
    "Policies": [
      {
        "PolicyName": "eb-pipeline-execution",
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Effect": "Allow",
              "Action": "codepipeline:StartPipelineExecution",
              "Resource": {
                "Fn::Join": [
                  "",
                  [
                    "arn:aws:codepipeline:",
                    {
                      "Ref": "AWS::Region"
                    },
                    ":",
                    {
                      "Ref": "AWS::AccountId"
```

```
                    },
                    ":",
                    {
                      "Ref": "AppPipeline"
                    }
                  ]
                ]

...
```

2. AWS::Events::Rule CloudFormation リソースを使用して EventBridge ルールを追加します。このイベントパターンは、Amazon S3 ソースバケット内のオブジェクトの作成または削除をモニタリングするイベントを作成します。さらに、パイプラインのターゲットも含めます。オブジェクトが作成されると、このルールによりターゲットパイプラインで StartPipelineExecution が呼び出されます。

   この変更を行う理由 AWS::Events::Rule リソースを追加すると CloudFormation 、 は イベントを作成できます。このリソースは CloudFormation スタックに追加されます。

   YAML

```
  EventRule:
    Type: AWS::Events::Rule
    Properties:
      EventBusName: default
      EventPattern:
        source:
          - aws.s3
        detail-type:
          - Object Created
        detail:
          bucket:
            name:
              - !Ref SourceBucket
      Name: EnabledS3SourceRule
      State: ENABLED
      Targets:
        -
          Arn:
            !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
'AWS::AccountId', ':', !Ref AppPipeline ] ]
          RoleArn: !GetAtt EventRole.Arn
          Id: codepipeline-AppPipeline
```

```
...
```

JSON

```
"EventRule": {
  "Type": "AWS::Events::Rule",
  "Properties": {
"EventBusName": "default",
"EventPattern": {
    "source": [
 "aws.s3"
    ],
    "detail-type": [
  "Object Created"
    ],
    "detail": {
  "bucket": {
      "name": [
    "s3-pipeline-source-fra-bucket"
      ]
      }
          }
},
"Name": "EnabledS3SourceRule",
      "State": "ENABLED",
      "Targets": [
      {
        "Arn": {
          "Fn::Join": [
            "",
            [
              "arn:aws:codepipeline:",
              {
                "Ref": "AWS::Region"
              },
              ":",
              {
                "Ref": "AWS::AccountId"
              },
              ":",
```

```
                {
                  "Ref": "AppPipeline"
                }
              ]
            ]
          },
          "RoleArn": {
            "Fn::GetAtt": [
              "EventRole",
              "Arn"
            ]
          },
          "Id": "codepipeline-AppPipeline"
        }
      ]
    }
  }
},

...
```

3. 更新したテンプレートをローカルコンピュータに保存し、 CloudFormation コンソールを開きます。
4. スタックを選択し、[既存スタックの変更セットの作成] を選択します。
5. 更新されたテンプレートをアップロードし、 CloudFormationに示された変更を表示します。これらがスタックに加えられる変更です。新しいリソースがリストに表示されています。
6. [実行] を選択してください。

パイプラインの PollForSourceChanges パラメータを編集するには

> **Important**
>
> このメソッドを使用してパイプラインを作成すると、PollForSourceChanges パラメータはデフォルトで true になります (ただし、明示的に false に設定した場合は除きます)。イベントベースの変更検出を追加する場合は、このパラメータを出力に追加する必要があります。ポーリングを無効にするには、このパラメータを false に設定します。そうしないと、1 つのソース変更に対してパイプラインが 2 回起動されます。詳細については、「PollForSourceChanges パラメータの有効な設定」を参照してください

- テンプレートで、PollForSourceChanges を false に変更します。パイプライン定義に PollForSourceChanges が含まれていなかった場合は、追加して false に設定します。

  この変更を行う理由 PollForSourceChanges パラメータを false に変更すると、定期的チェックがオフになるため、イベントベースの変更検出のみ使用することができます。

  YAML

```
        Name: Source
        Actions:
          -
            Name: SourceAction
            ActionTypeId:
              Category: Source
              Owner: AWS
              Version: 1
              Provider: S3
            OutputArtifacts:
              - Name: SourceOutput
            Configuration:
              S3Bucket: !Ref SourceBucket
              S3ObjectKey: !Ref SourceObjectKey
              PollForSourceChanges: false
            RunOrder: 1
```

  JSON

```
{
   "Name": "SourceAction",
   "ActionTypeId": {
     "Category": "Source",
     "Owner": "AWS",
     "Version": 1,
     "Provider": "S3"
   },
   "OutputArtifacts": [
     {
       "Name": "SourceOutput"
     }
   ],
   "Configuration": {
     "S3Bucket": {
```

```
        "Ref": "SourceBucket"
      },
      "S3ObjectKey": {
        "Ref": "SourceObjectKey"
      },
      "PollForSourceChanges": false
    },
    "RunOrder": 1
  }
```

Example

CloudFormation を使用してこれらのリソースを作成すると、リポジトリ内のファイルが作成または更新されたときにパイプラインがトリガーされます。

> **Note**
>
> ここで手順は終わりではありません。パイプラインは作成されますが、Amazon S3 パイプライン用の 2 番目の CloudFormation テンプレートを作成する必要があります。2 番目のテンプレートを作成しない場合、パイプラインに変更検出機能はありません。

YAML

```
Parameters:
  SourceObjectKey:
    Description: 'S3 source artifact'
    Type: String
    Default: SampleApp_Linux.zip
  ApplicationName:
    Description: 'CodeDeploy application name'
    Type: String
    Default: DemoApplication
  BetaFleet:
    Description: 'Fleet configured in CodeDeploy'
    Type: String
    Default: DemoFleet

Resources:
  SourceBucket:
```

```
    Type: AWS::S3::Bucket
    Properties:
      NotificationConfiguration:
        EventBridgeConfiguration:
          EventBridgeEnabled: true
      VersioningConfiguration:
        Status: Enabled
  CodePipelineArtifactStoreBucket:
    Type: AWS::S3::Bucket
  CodePipelineArtifactStoreBucketPolicy:
    Type: AWS::S3::BucketPolicy
    Properties:
      Bucket: !Ref CodePipelineArtifactStoreBucket
      PolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Sid: DenyUnEncryptedObjectUploads
            Effect: Deny
            Principal: '*'
            Action: s3:PutObject
            Resource: !Join [ '', [ !GetAtt CodePipelineArtifactStoreBucket.Arn, '/
*' ] ]
            Condition:
              StringNotEquals:
                s3:x-amz-server-side-encryption: aws:kms
          -
            Sid: DenyInsecureConnections
            Effect: Deny
            Principal: '*'
            Action: s3:*
            Resource: !Sub ${CodePipelineArtifactStoreBucket.Arn}/*
            Condition:
              Bool:
                aws:SecureTransport: false
  CodePipelineServiceRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Effect: Allow
            Principal:
```

```
            Service:
              - codepipeline.amazonaws.com
          Action: sts:AssumeRole
    Path: /
    Policies:
      -
        PolicyName: AWS-CodePipeline-Service-3
        PolicyDocument:
          Version: 2012-10-17
          Statement:
            -
              Effect: Allow
              Action:
                - codecommit:CancelUploadArchive
                - codecommit:GetBranch
                - codecommit:GetCommit
                - codecommit:GetUploadArchiveStatus
                - codecommit:UploadArchive
              Resource: 'resource_ARN'
            -
              Effect: Allow
              Action:
                - codedeploy:CreateDeployment
                - codedeploy:GetApplicationRevision
                - codedeploy:GetDeployment
                - codedeploy:GetDeploymentConfig
                - codedeploy:RegisterApplicationRevision
              Resource: 'resource_ARN'
            -
              Effect: Allow
              Action:
                - codebuild:BatchGetBuilds
                - codebuild:StartBuild
              Resource: 'resource_ARN'
            -
              Effect: Allow
              Action:
                - devicefarm:ListProjects
                - devicefarm:ListDevicePools
                - devicefarm:GetRun
                - devicefarm:GetUpload
                - devicefarm:CreateUpload
                - devicefarm:ScheduleRun
              Resource: 'resource_ARN'
```

```
            -
              Effect: Allow
              Action:
                - lambda:InvokeFunction
                - lambda:ListFunctions
              Resource: 'resource_ARN'
            -
              Effect: Allow
              Action:
                - iam:PassRole
              Resource: 'resource_ARN'
            -
              Effect: Allow
              Action:
                - elasticbeanstalk:*
                - ec2:*
                - elasticloadbalancing:*
                - autoscaling:*
                - cloudwatch:*
                - s3:*
                - sns:*
                - cloudformation:*
                - rds:*
                - sqs:*
                - ecs:*
              Resource: 'resource_ARN'
AppPipeline:
  Type: AWS::CodePipeline::Pipeline
  Properties:
    Name: s3-events-pipeline
    RoleArn:
      !GetAtt CodePipelineServiceRole.Arn
    Stages:
      -
        Name: Source
        Actions:
          -
            Name: SourceAction
            ActionTypeId:
              Category: Source
              Owner: AWS
              Version: 1
              Provider: S3
            OutputArtifacts:
```

```
              - Name: SourceOutput
            Configuration:
              S3Bucket: !Ref SourceBucket
              S3ObjectKey: !Ref SourceObjectKey
              PollForSourceChanges: false
            RunOrder: 1
      -
        Name: Beta
        Actions:
          -
            Name: BetaAction
            InputArtifacts:
              - Name: SourceOutput
            ActionTypeId:
              Category: Deploy
              Owner: AWS
              Version: 1
              Provider: CodeDeploy
            Configuration:
              ApplicationName: !Ref ApplicationName
              DeploymentGroupName: !Ref BetaFleet
            RunOrder: 1
    ArtifactStore:
      Type: S3
      Location: !Ref CodePipelineArtifactStoreBucket
EventRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: 2012-10-17
      Statement:
        -
          Effect: Allow
          Principal:
            Service:
              - events.amazonaws.com
          Action: sts:AssumeRole
    Path: /
    Policies:
      -
        PolicyName: eb-pipeline-execution
        PolicyDocument:
          Version: 2012-10-17
          Statement:
```

```
            -
              Effect: Allow
              Action: codepipeline:StartPipelineExecution
              Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region',
':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]
 EventRule:
   Type: AWS::Events::Rule
   Properties:
     EventBusName: default
     EventPattern:
       source:
         - aws.s3
       detail-type:
         - Object Created
       detail:
         bucket:
           name:
             - !Ref SourceBucket
     Name: EnabledS3SourceRule
     State: ENABLED
     Targets:
       -
         Arn:
           !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
'AWS::AccountId', ':', !Ref AppPipeline ] ]
         RoleArn: !GetAtt EventRole.Arn
         Id: codepipeline-AppPipeline
```

JSON

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "appconfig:StartDeployment",
                "appconfig:StopDeployment",
                "appconfig:GetDeployment"
            ],
            "Resource": [
```

```
                "arn:aws:appconfig:*:111122223333:application/
[[Application]]",
                "arn:aws:appconfig:*:111122223333:application/[[Application]]/
*",
                "arn:aws:appconfig:*:111122223333:deploymentstrategy/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

S3 ソースと CloudTrail 証跡を使用してポーリングパイプラインを移行する

Amazon S3 ソースを含むパイプラインでは、変更検出が EventBridge を通じて自動化されるように、パイプラインを修正します。以下のいずれかの方法で移行を実装します。

- コンソール: ポーリングパイプラインを移行する (CodeCommit または Amazon S3 ソース) (コンソール)
- CLI: S3 ソースと CloudTrail 証跡を使用してポーリングパイプラインを移行する (CLI)
- CloudFormation: S3 ソースと CloudTrail 証跡 (CloudFormation テンプレート) を使用してポーリングパイプラインを移行する

S3 ソースと CloudTrail 証跡を使用してポーリングパイプラインを移行する (CLI)

ポーリング (定期的なチェック) を使用しているパイプラインを、EventBridge のイベントを使用するように編集するには、次の手順に従います。パイプラインを作成する場合は、「パイプライン、ステージ、アクションを作成する」を参照してください。

Amazon S3 でイベント駆動型パイプラインを構築するには、パイプラインの `PollForSourceChanges` パラメータを編集してから、以下のリソースを作成します。

- AWS CloudTrail Amazon S3 がイベントのログ記録に使用できる証跡、バケット、バケットポリシー。
- EventBridge イベント
- EventBridge イベントによるパイプラインの開始を許可する IAM ロール

AWS CloudTrail 証跡を作成し、ログ記録を有効にするには

を使用して証跡 AWS CLI を作成するには、 create-trail コマンドを呼び出し、以下を指定します。

- 証跡名。
- AWS CloudTrailにバケットポリシーをすでに適用しているバケットです。

詳細については、AWS 「コマンドラインインターフェイスを使用した証跡の作成」を参照してください。

1. create-trail コマンドを呼び出し、--name および --s3-bucket-name パラメータを含めます。

   この変更を行う理由 これにより、S3 ソースバケットに必要な CloudTrail 証跡が作成されます。

   次のコマンドでは、--name および --s3-bucket-name を使用して、my-trail という名前の証跡と、amzn-s3-demo-source-bucket という名前のバケットを作成します。

   ```
   aws cloudtrail create-trail --name my-trail --s3-bucket-name amzn-s3-demo-source-bucket
   ```

2. start-logging コマンドを呼び出し、--name パラメータを含めます。

   この変更を行う理由 これにより、ソースバケットの CloudTrail ロギングが開始され、EventBridge にイベントが送信されます。

   例:

   次のコマンドでは、--name を使用して、my-trail という名前の証跡のログ記録を開始します。

   ```
   aws cloudtrail start-logging --name my-trail
   ```

3. put-event-selectors コマンドを呼び出し、--trail-name および --event-selectors パラメータを含めます。イベントセレクタを使用してソースバケットに記録するデータイベントを指定し、それらのイベントを EventBridge ルールに送信します。

   この変更を行う理由 このコマンドはイベントをフィルタ処理します。

   例:

次のサンプルコマンドでは、--trail-name および --event-selectors を使用してソースバケットと amzn-s3-demo-source-bucket/myFolder という名前のプレフィックスにデータイベントの管理を指定します。

```
aws cloudtrail put-event-selectors --trail-name my-trail --event-selectors
 '[{ "ReadWriteType": "WriteOnly", "IncludeManagementEvents":false,
 "DataResources": [{ "Type": "AWS::S3::Object", "Values": ["arn:aws:s3:::amzn-s3-
demo-source-bucket/myFolder/file.zip"] }] }]'
```

Amazon S3 をイベントソース、CodePipeline をターゲットとする EventBridge ルールを作成し、アクセス許可ポリシーを適用するには

1. EventBridge が CodePipeline を使用してルールを呼び出すためのアクセス許可を付与します。詳細については、 デベロッパーガイドの [Amazon EventBridge のリソースベースのポリシーを使用する] を参照してください。

   a. 次のサンプルを使用して、EventBridge にサービスロールの引き受けを許可する信頼ポリシーを作成します。このスクリプトに trustpolicyforEB.json という名前を付けます。

      JSON

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

   b. 次のコマンドを使用して、Role-for-MyRule ロールを作成し、信頼ポリシーをアタッチします。

この変更を行う理由 ロールにこの信頼ポリシーを追加すると、EventBridge に対するアクセス許可が作成されます。

```
aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document
 file://trustpolicyforEB.json
```

c. 次に示すように、MyFirstPipeline という名前のパイプラインに対してアクセス許可ポリシー JSON を作成します。アクセス権限ポリシーに permissionspolicyforEB.json と名前を付けます。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:StartPipelineExecution"
            ],
            "Resource": [
                "arn:aws:codepipeline:us-
west-2:111122223333:MyFirstPipeline"
            ]
        }
    ]
}
```

d. 次のコマンドを実行して、作成した Role-for-MyRule ロールに新しい CodePipeline-Permissions-Policy-for-EB アクセス権限ポリシーをアタッチします。

```
aws iam put-role-policy --role-name Role-for-MyRule --policy-name CodePipeline-
Permissions-Policy-For-EB --policy-document file://permissionspolicyforEB.json
```

2. put-rule コマンドを呼び出し、--name、--event-pattern、--role-arn パラメータを含めます。

次のサンプルコマンドでは、MyS3SourceRule という名前のルールが作成されます。

```
aws events put-rule --name "MyS3SourceRule" --event-pattern "{\"source\":
[\"aws.s3\"],\"detail-type\":[\"AWS API Call via CloudTrail\"],\"detail\":
{\"eventSource\":[\"s3.amazonaws.com\"],\"eventName\":[\"CopyObject\",\"PutObject
\",\"CompleteMultipartUpload\"],\"requestParameters\":{\"bucketName\":[\"amzn-s3-
demo-source-bucket\"],\"key\":[\"my-key\"]}}}
 --role-arn "arn:aws:iam::ACCOUNT_ID:role/Role-for-MyRule"
```

3. CodePipeline をターゲットとして追加するには、put-targets コマンドを呼び出し、--rule および --targets パラメータを含めます。

   次のコマンドでは、MyS3SourceRule という名前のルールに対して指定し、ターゲット Id は 1 番で構成されています。これは、ルールのターゲットのリストが何であるかを示し、この場合は ターゲット 1 です。このコマンドでは、パイプラインのサンプルの ARN も指定されます。パイプラインは、リポジトリ内に変更が加えられると開始します。

   ```
   aws events put-targets --rule MyS3SourceRule --targets
    Id=1,Arn=arn:aws:codepipeline:us-west-2:80398EXAMPLE:TestPipeline
   ```

4. (オプション) 特定のイメージ ID のソースオーバーライドを使用して入力トランスフォーマーを設定するには、CLI コマンドで次の JSON を使用します。次の例では、以下のオーバーライドを設定します。

   - actionName は (この例では Source)、ソースイベントから派生したものではなく、パイプラインの作成時に定義された動的な値です。
   - revisionType は (この例では S3_OBJECT_VERSION_ID)、ソースイベントから派生したものではなく、パイプラインの作成時に定義された動的な値です。
   - revisionValue は (この例では <*revisionValue*>)、ソースイベント変数から派生しています。

   ```
   {
       "Rule": "my-rule",
       "Targets": [
           {
               "Id": "MyTargetId",
               "Arn": "ARN",
               "InputTransformer": {
                   "InputPathsMap": {
                       "revisionValue": "$.detail.object.version-id"
                   },
   ```

```
                "InputTemplate": {
                    "sourceRevisions": {
                        "actionName": "Source",
                        "revisionType": "S3_OBJECT_VERSION_ID",
                        "revisionValue": "<revisionValue>"
                    }
                }
            }
        }
    ]
}
```

パイプラインの PollForSourceChanges パラメータを編集するには

> **Important**
>
> このメソッドを使用してパイプラインを作成すると、PollForSourceChanges パラメータはデフォルトで true になります (ただし、明示的に false に設定した場合は除きます)。イベントベースの変更検出を追加する場合は、このパラメータを出力に追加する必要があります。ポーリングを無効にするには、このパラメータを false に設定します。そうしないと、1 つのソース変更に対してパイプラインが 2 回起動されます。詳細については、「PollForSourceChanges パラメータの有効な設定」を参照してください

1. get-pipeline コマンドを実行して、パイプライン構造を JSON ファイルにコピーします。例えば、MyFirstPipeline という名前のパイプラインに対して、以下のコマンドを実行します。

   ```
   aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
   ```

   このコマンドは何も返しませんが、作成したファイルは、コマンドを実行したディレクトリにあります。

2. この例に示すように、プレーンテキストエディタで JSON ファイルを開き、PollForSourceChanges という名前のバケットの amzn-s3-demo-source-bucket パラメータを false に変更してソースステージを編集します。

   この変更を行う理由 このパラメータを false に設定すると、定期的チェックがオフになるため、イベントベースの変更検出のみ使用することができます。

```
"configuration": {
    "S3Bucket": "amzn-s3-demo-source-bucket",
    "PollForSourceChanges": "false",
    "S3ObjectKey": "index.zip"
},
```

3. get-pipeline コマンドを使用して取得したパイプライン構造を使用している場合、JSON ファイルから metadata 行を削除する必要があります。それ以外の場合は、update-pipeline コマンドで使用することはできません。"metadata": { } 行と、"created"、"pipelineARN"、"updated" フィールドを削除します。

   例えば、構造から以下の行を削除します。

```
"metadata": {
    "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
    "created": "date",
    "updated": "date"
},
```

   ファイルを保存します。

4. 変更を適用するには、パイプライン JSON ファイルを指定して、update-pipeline コマンドを実行します。

> **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 codepipeline update-pipeline --cli-input-json file://pipeline.json
```

このコマンドは、編集したパイプラインの構造全体を返します。

> **Note**
>
> update-pipeline コマンドは、パイプラインを停止します。update-pipeline コマンドを実行したときにパイプラインによりリビジョンが実行されている場合、その実行は停止します。更新されたパイプラインによりそのリビジョンを実行するには、パイプライン

を手動で開始する必要があります。パイプラインを手動で開始するには start-pipeline-execution コマンドを使用します。

S3 ソースと CloudTrail 証跡 (CloudFormation テンプレート) を使用してポーリングパイプラインを移行する

以下の手順を使用して、Amazon S3 ソースを含むパイプラインを、ポーリングからイベントベースの変更検出に編集します。

Amazon S3 でイベント駆動型パイプラインを構築するには、パイプラインの `PollForSourceChanges` パラメータを編集してから、以下のリソースをテンプレートに追加します。

- EventBridge では、すべての Amazon S3 イベントをログに記録する必要があります。発生するイベントのロク記録に Amazon S3 が使用できる AWS CloudTrail 証跡、バケット、バケットポリシーを作成する必要があります。詳細については、「 証跡のデータイベント 」と「 管理イベントの口グ記録 」を参照してください。
- このイベントによるパイプラインの開始を許可する EventBridge ルールと IAM ロール。

CloudFormation を使用してパイプラインを作成および管理する場合、テンプレートには次のようなコンテンツが含まれます。

> **Note**
>
> `PollForSourceChanges` と呼ばれるソースステージの `Configuration` プロパティ。テンプレートにプロパティが含まれていない場合、`PollForSourceChanges` はデフォルトで `true` に設定されます。

YAML

```
AppPipeline:
  Type: AWS::CodePipeline::Pipeline
  Properties:
    RoleArn: !GetAtt CodePipelineServiceRole.Arn
    Stages:
      -
```

```
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: AWS
                Version: 1
                Provider: S3
              OutputArtifacts:
                -
                  Name: SourceOutput
              Configuration:
                S3Bucket: !Ref SourceBucket
                S3ObjectKey: !Ref S3SourceObjectKey
                PollForSourceChanges: true
              RunOrder: 1


...
```

JSON

```
        "AppPipeline": {
            "Type": "AWS::CodePipeline::Pipeline",
            "Properties": {
                "RoleArn": {
                    "Fn::GetAtt": ["CodePipelineServiceRole", "Arn"]
                },
                "Stages": [
                    {
                        "Name": "Source",
                        "Actions": [
                            {
                                "Name": "SourceAction",
                                "ActionTypeId": {
                                    "Category": "Source",
                                    "Owner": "AWS",
                                    "Version": 1,
                                    "Provider": "S3"
                                },
                                "OutputArtifacts": [
                                    {
```

```
                                "Name": "SourceOutput"
                            }
                        ],
                        "Configuration": {
                            "S3Bucket": {
                                "Ref": "SourceBucket"
                            },
                            "S3ObjectKey": {
                                "Ref": "SourceObjectKey"
                            },
                            "PollForSourceChanges": true
                        },
                        "RunOrder": 1
                    }
                ]
            },


...
```

Amazon S3 をイベントソース、CodePipeline をターゲットとする EventBridge ルールを作成し、アクセス許可ポリシーを適用するには

1. テンプレートの でResources、 AWS::IAM::Role CloudFormation リソースを使用して、イベントがパイプラインを開始できるようにする IAM ロールを設定します。このエントリによって、2 つのポリシーを使用するロールが作成されます。

   - 最初のポリシーでは、ロールを引き受けることを許可します。
   - 2 つめのポリシーでは、パイプラインを開始するアクセス権限が付与されます。

   この変更を行う理由 AWS::IAM::Role リソースを追加する CloudFormation と、 は EventBridge のアクセス許可を作成できます。このリソースは CloudFormation スタックに追加されます。

   YAML

   ```
   EventRole:
     Type: AWS::IAM::Role
     Properties:
       AssumeRolePolicyDocument:
   ```

```
        Version: 2012-10-17
        Statement:
          -
            Effect: Allow
            Principal:
              Service:
                - events.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref
 'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]


...
```

JSON

```
 "EventRole": {
   "Type": "AWS::IAM::Role",
   "Properties": {
     "AssumeRolePolicyDocument": {
       "Version": "2012-10-17",
       "Statement": [
         {
           "Effect": "Allow",
           "Principal": {
             "Service": [
               "events.amazonaws.com"
             ]
           },
           "Action": "sts:AssumeRole"
         }
       ]
     },
```

```
      "Path": "/",
      "Policies": [
        {
          "PolicyName": "eb-pipeline-execution",
          "PolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
              {
                "Effect": "Allow",
                "Action": "codepipeline:StartPipelineExecution",
                "Resource": {
                  "Fn::Join": [
                    "",
                    [
                      "arn:aws:codepipeline:",
                      {
                        "Ref": "AWS::Region"
                      },
                      ":",
                      {
                        "Ref": "AWS::AccountId"
                      },
                      ":",
                      {
                        "Ref": "AppPipeline"
                      }
                    ]
                  ]

...
```

2. AWS::Events::Rule CloudFormation リソースを使用して EventBridge ルールを追加します。このイベントパターンは、Amazon S3 ソースバケットでの CopyObject、PutObject、および CompleteMultipartUpload をモニタリングするイベントを作成します。さらに、パイプラインのターゲットも含めます。CopyObject、PutObject、または CompleteMultipartUpload が発生すると、このルールは、ターゲットパイプラインで StartPipelineExecution を呼び出します。

   この変更を行う理由 AWS::Events::Rule リソースを追加すると CloudFormation 、 は イベントを作成できます。このリソースは CloudFormation スタックに追加されます。

YAML

```
  EventRule:
    Type: AWS::Events::Rule
    Properties:
      EventPattern:
        source:
          - aws.s3
        detail-type:
          - 'AWS API Call via CloudTrail'
        detail:
          eventSource:
            - s3.amazonaws.com
          eventName:
            - CopyObject
            - PutObject
            - CompleteMultipartUpload
          requestParameters:
            bucketName:
              - !Ref SourceBucket
            key:
              - !Ref SourceObjectKey
      Targets:
        -
          Arn:
            !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref AppPipeline ] ]
          RoleArn: !GetAtt EventRole.Arn
          Id: codepipeline-AppPipeline


...
```

JSON

```
  "EventRule": {
    "Type": "AWS::Events::Rule",
    "Properties": {
      "EventPattern": {
        "source": [
          "aws.s3"
```

```
        ],
        "detail-type": [
          "AWS API Call via CloudTrail"
        ],
        "detail": {
          "eventSource": [
            "s3.amazonaws.com"
          ],
          "eventName": [
            "CopyObject",
            "PutObject",
            "CompleteMultipartUpload"
          ],
          "requestParameters": {
            "bucketName": [
              {
                "Ref": "SourceBucket"
              }
            ],
            "key": [
              {
                "Ref": "SourceObjectKey"
              }
            ]
          }
        }
      },
      "Targets": [
        {
          "Arn": {
            "Fn::Join": [
              "",
              [
                "arn:aws:codepipeline:",
                {
                  "Ref": "AWS::Region"
                },
                ":",
                {
                  "Ref": "AWS::AccountId"
                },
                ":",
                {
                  "Ref": "AppPipeline"
```

```
                }
              ]
            ]
          },
          "RoleArn": {
            "Fn::GetAtt": [
              "EventRole",
              "Arn"
            ]
          },
          "Id": "codepipeline-AppPipeline"
        }
      ]
    }
  }
},

...
```

3. このスニペットを最初のテンプレートに追加して、クロススタック機能を有効にします。

YAML

```
Outputs:
  SourceBucketARN:
    Description: "S3 bucket ARN that Cloudtrail will use"
    Value: !GetAtt SourceBucket.Arn
    Export:
      Name: SourceBucketARN
```

JSON

```
  "Outputs" : {
    "SourceBucketARN" : {
      "Description" : "S3 bucket ARN that Cloudtrail will use",
      "Value" : { "Fn::GetAtt": ["SourceBucket", "Arn"] },
      "Export" : {
        "Name" : "SourceBucketARN"
      }
    }

...
```

4. (オプション) 特定のイメージ ID のソースオーバーライドを使用して入力トランスフォーマーを設定するには、次の YAML スニペットを使用します。次の例では、以下のオーバーライドを設定します。

   - actionName は (この例では Source)、ソースイベントから派生したものではなく、パイプラインの作成時に定義された動的な値です。
   - revisionType は (この例では S3_OBJECT_VERSION_ID)、ソースイベントから派生したものではなく、パイプラインの作成時に定義された動的な値です。
   - revisionValue は (この例では <*revisionValue*>)、ソースイベント変数から派生しています。

```
---
Rule: my-rule
Targets:
- Id: MyTargetId
  Arn: pipeline-ARN
  InputTransformer:
    InputPathsMap:
      revisionValue: "$.detail.object.version-id"
    InputTemplate:
      sourceRevisions:
        actionName: Source
        revisionType: S3_OBJECT_VERSION_ID
        revisionValue: '<revisionValue>'
```

5. 更新されたテンプレートをローカルコンピュータに保存し、 CloudFormation コンソールを開きます。
6. スタックを選択し、[既存スタックの変更セットの作成] を選択します。
7. 更新されたテンプレートをアップロードし、 CloudFormationに示された変更を表示します。これらがスタックに加えられる変更です。新しいリソースがリストに表示されています。
8. [実行] を選択してください。

パイプラインの PollForSourceChanges パラメータを編集するには

> **Important**
>
> このメソッドを使用してパイプラインを作成すると、PollForSourceChanges パラメータはデフォルトで true になります (ただし、明示的に false に設定した場合は除きます)。

イベントベースの変更検出を追加する場合は、このパラメータを出力に追加する必要があります。ポーリングを無効にするには、このパラメータを false に設定します。そうしないと、1 つのソース変更に対してパイプラインが 2 回起動されます。詳細については、「PollForSourceChanges パラメータの有効な設定」を参照してください

- テンプレートで、PollForSourceChanges を false に変更します。パイプライン定義に PollForSourceChanges が含まれていなかった場合は、追加して false に設定します。

  この変更を行う理由 PollForSourceChanges パラメータを false に変更すると、定期的チェックがオフになるため、イベントベースの変更検出のみ使用することができます。

  YAML

```
        Name: Source
        Actions:
          -
            Name: SourceAction
            ActionTypeId:
              Category: Source
              Owner: AWS
              Version: 1
              Provider: S3
            OutputArtifacts:
              - Name: SourceOutput
            Configuration:
              S3Bucket: !Ref SourceBucket
              S3ObjectKey: !Ref SourceObjectKey
              PollForSourceChanges: false
            RunOrder: 1
```

  JSON

```
{
   "Name": "SourceAction",
   "ActionTypeId": {
     "Category": "Source",
     "Owner": "AWS",
     "Version": 1,
     "Provider": "S3"
   },
```

```
    "OutputArtifacts": [
      {
        "Name": "SourceOutput"
      }
    ],
    "Configuration": {
      "S3Bucket": {
        "Ref": "SourceBucket"
      },
      "S3ObjectKey": {
        "Ref": "SourceObjectKey"
      },
      "PollForSourceChanges": false
    },
    "RunOrder": 1
  }
```

Amazon S3 パイプラインの CloudTrail リソース用に 2 番目のテンプレートを作成するには

- 別のテンプレートの でResources、、AWS::S3::BucketPolicy、および AWS::S3::BucketAWS::CloudTrail::Trail CloudFormation リソースを使用して、CloudTrail のシンプルなバケット定義と証跡を提供します。

  この変更を行う理由 CloudTrail 証跡は、アカウントあたり 5 証跡を現在の制限として、個別に作成して管理する必要があります。(「 の制限 AWS CloudTrail」を参照してください。) ただし、1 つの証跡に複数の Amazon S3 バケットを含めることができるため、いったん証跡を作成してから、必要に応じて他のパイプライン用に Amazon S3 バケットを追加できます。2 番目のサンプルテンプレートファイルに以下のコードを貼り付けます。

  YAML

```
###############################################################################
# Prerequisites:
#   - S3 SourceBucket and SourceObjectKey must exist
###############################################################################

Parameters:
  SourceObjectKey:
    Description: 'S3 source artifact'
    Type: String
```

```
    Default: SampleApp_Linux.zip

Resources:
  AWSCloudTrailBucketPolicy:
    Type: AWS::S3::BucketPolicy
    Properties:
      Bucket: !Ref AWSCloudTrailBucket
      PolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Sid: AWSCloudTrailAclCheck
            Effect: Allow
            Principal:
              Service:
                - cloudtrail.amazonaws.com
            Action: s3:GetBucketAcl
            Resource: !GetAtt AWSCloudTrailBucket.Arn
          -
            Sid: AWSCloudTrailWrite
            Effect: Allow
            Principal:
              Service:
                - cloudtrail.amazonaws.com
            Action: s3:PutObject
            Resource: !Join [ '', [ !GetAtt AWSCloudTrailBucket.Arn, '/
AWSLogs/', !Ref 'AWS::AccountId', '/*' ] ]
            Condition:
              StringEquals:
                s3:x-amz-acl: bucket-owner-full-control
  AWSCloudTrailBucket:
    Type: AWS::S3::Bucket
    DeletionPolicy: Retain
  AwsCloudTrail:
    DependsOn:
      - AWSCloudTrailBucketPolicy
    Type: AWS::CloudTrail::Trail
    Properties:
      S3BucketName: !Ref AWSCloudTrailBucket
      EventSelectors:
        -
          DataResources:
            -
              Type: AWS::S3::Object
```

```
              Values:
                - !Join [ '', [ !ImportValue SourceBucketARN, '/', !Ref
 SourceObjectKey ] ]
          ReadWriteType: WriteOnly
          IncludeManagementEvents: false
      IncludeGlobalServiceEvents: true
      IsLogging: true
      IsMultiRegionTrail: true


...
```

JSON

```
{
  "Parameters": {
    "SourceObjectKey": {
      "Description": "S3 source artifact",
      "Type": "String",
      "Default": "SampleApp_Linux.zip"
    }
  },
  "Resources": {
    "AWSCloudTrailBucket": {
      "Type": "AWS::S3::Bucket",
        "DeletionPolicy": "Retain"
    },
    "AWSCloudTrailBucketPolicy": {
      "Type": "AWS::S3::BucketPolicy",
      "Properties": {
        "Bucket": {
          "Ref": "AWSCloudTrailBucket"
        },
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Sid": "AWSCloudTrailAclCheck",
              "Effect": "Allow",
              "Principal": {
                "Service": [
                  "cloudtrail.amazonaws.com"
                ]
```

```
            },
            "Action": "s3:GetBucketAcl",
            "Resource": {
              "Fn::GetAtt": [
                "AWSCloudTrailBucket",
                "Arn"
              ]
            }
          },
          {
            "Sid": "AWSCloudTrailWrite",
            "Effect": "Allow",
            "Principal": {
              "Service": [
                "cloudtrail.amazonaws.com"
              ]
            },
            "Action": "s3:PutObject",
            "Resource": {
              "Fn::Join": [
                "",
                [
                  {
                    "Fn::GetAtt": [
                      "AWSCloudTrailBucket",
                      "Arn"
                    ]
                  },
                  "/AWSLogs/",
                  {
                    "Ref": "AWS::AccountId"
                  },
                  "/*"
                ]
              ]
            },
            "Condition": {
              "StringEquals": {
                "s3:x-amz-acl": "bucket-owner-full-control"
              }
            }
          }
        ]
      }
```

```
      }
    },
    "AwsCloudTrail": {
      "DependsOn": [
        "AWSCloudTrailBucketPolicy"
      ],
      "Type": "AWS::CloudTrail::Trail",
      "Properties": {
        "S3BucketName": {
          "Ref": "AWSCloudTrailBucket"
        },
        "EventSelectors": [
          {
            "DataResources": [
              {
                "Type": "AWS::S3::Object",
                "Values": [
                  {
                    "Fn::Join": [
                      "",
                      [
                        {
                          "Fn::ImportValue": "SourceBucketARN"
                        },
                        "/",
                        {
                          "Ref": "SourceObjectKey"
                        }
                      ]
                    ]
                  }
                ]
              }
            ],
            "ReadWriteType": "WriteOnly",
            "IncludeManagementEvents": false
          }
        ],
        "IncludeGlobalServiceEvents": true,
        "IsLogging": true,
        "IsMultiRegionTrail": true
      }
    }
  }
```

```
}

...
```

Example

CloudFormation を使用してこれらのリソースを作成すると、リポジトリ内のファイルが作成または更新されたときにパイプラインがトリガーされます。

> **Note**
>
> ここで手順は終わりではありません。パイプラインは作成されますが、Amazon S3 パイプライン用の 2 番目の CloudFormation テンプレートを作成する必要があります。2 番目のテンプレートを作成しない場合、パイプラインに変更検出機能はありません。

YAML

```
Resources:
  SourceBucket:
    Type: AWS::S3::Bucket
    Properties:
      VersioningConfiguration:
        Status: Enabled
  CodePipelineArtifactStoreBucket:
    Type: AWS::S3::Bucket
  CodePipelineArtifactStoreBucketPolicy:
    Type: AWS::S3::BucketPolicy
    Properties:
      Bucket: !Ref CodePipelineArtifactStoreBucket
      PolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Sid: DenyUnEncryptedObjectUploads
            Effect: Deny
            Principal: '*'
            Action: s3:PutObject
            Resource: !Join [ '', [ !GetAtt CodePipelineArtifactStoreBucket.Arn, '/
*' ] ]
            Condition:
```

```
              StringNotEquals:
                s3:x-amz-server-side-encryption: aws:kms
          -
            Sid: DenyInsecureConnections
            Effect: Deny
            Principal: '*'
            Action: s3:*
            Resource: !Join [ '', [ !GetAtt CodePipelineArtifactStoreBucket.Arn, '/
*' ] ]
            Condition:
              Bool:
                aws:SecureTransport: false
  CodePipelineServiceRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Effect: Allow
            Principal:
              Service:
                - codepipeline.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: AWS-CodePipeline-Service-3
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action:
                  - codecommit:CancelUploadArchive
                  - codecommit:GetBranch
                  - codecommit:GetCommit
                  - codecommit:GetUploadArchiveStatus
                  - codecommit:UploadArchive
                Resource: 'resource_ARN'
              -
                Effect: Allow
                Action:
                  - codedeploy:CreateDeployment
```

```
        - codedeploy:GetApplicationRevision
        - codedeploy:GetDeployment
        - codedeploy:GetDeploymentConfig
        - codedeploy:RegisterApplicationRevision
      Resource: 'resource_ARN'
    -
      Effect: Allow
      Action:
        - codebuild:BatchGetBuilds
        - codebuild:StartBuild
      Resource: 'resource_ARN'
    -
      Effect: Allow
      Action:
        - devicefarm:ListProjects
        - devicefarm:ListDevicePools
        - devicefarm:GetRun
        - devicefarm:GetUpload
        - devicefarm:CreateUpload
        - devicefarm:ScheduleRun
      Resource: 'resource_ARN'
    -
      Effect: Allow
      Action:
        - lambda:InvokeFunction
        - lambda:ListFunctions
      Resource: 'resource_ARN'
    -
      Effect: Allow
      Action:
        - iam:PassRole
      Resource: 'resource_ARN'
    -
      Effect: Allow
      Action:
        - elasticbeanstalk:*
        - ec2:*
        - elasticloadbalancing:*
        - autoscaling:*
        - cloudwatch:*
        - s3:*
        - sns:*
        - cloudformation:*
        - rds:*
```

```
                - sqs:*
                - ecs:*
              Resource: 'resource_ARN'
  AppPipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      Name: s3-events-pipeline
      RoleArn:
        !GetAtt CodePipelineServiceRole.Arn
      Stages:
        -
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: AWS
                Version: 1
                Provider: S3
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                S3Bucket: !Ref SourceBucket
                S3ObjectKey: !Ref SourceObjectKey
                PollForSourceChanges: false
              RunOrder: 1
        -
          Name: Beta
          Actions:
            -
              Name: BetaAction
              InputArtifacts:
                - Name: SourceOutput
              ActionTypeId:
                Category: Deploy
                Owner: AWS
                Version: 1
                Provider: CodeDeploy
              Configuration:
                ApplicationName: !Ref ApplicationName
                DeploymentGroupName: !Ref BetaFleet
              RunOrder: 1
      ArtifactStore:
```

```
        Type: S3
        Location: !Ref CodePipelineArtifactStoreBucket
  EventRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Effect: Allow
            Principal:
              Service:
                - events.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region',
':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]
  EventRule:
    Type: AWS::Events::Rule
    Properties:
      EventPattern:
        source:
          - aws.s3
        detail-type:
          - 'AWS API Call via CloudTrail'
        detail:
          eventSource:
            - s3.amazonaws.com
          eventName:
            - PutObject
            - CompleteMultipartUpload
          resources:
            ARN:
              - !Join [ '', [ !GetAtt SourceBucket.Arn, '/', !Ref
SourceObjectKey ] ]
```

```
      Targets:
        -
          Arn:
            !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref AppPipeline ] ]
          RoleArn: !GetAtt EventRole.Arn
          Id: codepipeline-AppPipeline

Outputs:
  SourceBucketARN:
    Description: "S3 bucket ARN that Cloudtrail will use"
    Value: !GetAtt SourceBucket.Arn
    Export:
      Name: SourceBucketARN
```

JSON

```
    "Resources": {
        "SourceBucket": {
            "Type": "AWS::S3::Bucket",
            "Properties": {
                "VersioningConfiguration": {
                    "Status": "Enabled"
                }
            }
        },
        "CodePipelineArtifactStoreBucket": {
            "Type": "AWS::S3::Bucket"
        },
        "CodePipelineArtifactStoreBucketPolicy": {
            "Type": "AWS::S3::BucketPolicy",
            "Properties": {
                "Bucket": {
                    "Ref": "CodePipelineArtifactStoreBucket"
                },
                "PolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Sid": "DenyUnEncryptedObjectUploads",
                            "Effect": "Deny",
                            "Principal": "*",
                            "Action": "s3:PutObject",
```

```
                "Resource": {
                    "Fn::Join": [
                        "",
                        [
                            {
                                "Fn::GetAtt": [
                                    "CodePipelineArtifactStoreBucket",
                                    "Arn"
                                ]
                            },
                            "/*"
                        ]
                    ]
                },
                "Condition": {
                    "StringNotEquals": {
                        "s3:x-amz-server-side-encryption": "aws:kms"
                    }
                }
            },
            {
                "Sid": "DenyInsecureConnections",
                "Effect": "Deny",
                "Principal": "*",
                "Action": "s3:*",
                "Resource": {
                    "Fn::Join": [
                        "",
                        [
                            {
                                "Fn::GetAtt": [
                                    "CodePipelineArtifactStoreBucket",
                                    "Arn"
                                ]
                            },
                            "/*"
                        ]
                    ]
                },
                "Condition": {
                    "Bool": {
                        "aws:SecureTransport": false
                    }
                }
```

```
                    }
                ]
            }
        }
    },
    "CodePipelineServiceRole": {
        "Type": "AWS::IAM::Role",
        "Properties": {
            "AssumeRolePolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Principal": {
                            "Service": [
                                "codepipeline.amazonaws.com"
                            ]
                        },
                        "Action": "sts:AssumeRole"
                    }
                ]
            },
            "Path": "/",
            "Policies": [
                {
                    "PolicyName": "AWS-CodePipeline-Service-3",
                    "PolicyDocument": {
                        "Version": "2012-10-17",
                        "Statement": [
                            {
                                "Effect": "Allow",
                                "Action": [
                                    "codecommit:CancelUploadArchive",
                                    "codecommit:GetBranch",
                                    "codecommit:GetCommit",
                                    "codecommit:GetUploadArchiveStatus",
                                    "codecommit:UploadArchive"
                                ],
                                "Resource": "resource_ARN"
                            },
                            {
                                "Effect": "Allow",
                                "Action": [
                                    "codedeploy:CreateDeployment",
```

```
                "codedeploy:GetApplicationRevision",
                "codedeploy:GetDeployment",
                "codedeploy:GetDeploymentConfig",
                "codedeploy:RegisterApplicationRevision"
            ],
            "Resource": "resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
                "codebuild:BatchGetBuilds",
                "codebuild:StartBuild"
            ],
            "Resource": "resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
                "devicefarm:ListProjects",
                "devicefarm:ListDevicePools",
                "devicefarm:GetRun",
                "devicefarm:GetUpload",
                "devicefarm:CreateUpload",
                "devicefarm:ScheduleRun"
            ],
            "Resource": "resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction",
                "lambda:ListFunctions"
            ],
            "Resource": "resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "resource_ARN"
        },
        {
            "Effect": "Allow",
```

```
                                "Action": [
                                    "elasticbeanstalk:*",
                                    "ec2:*",
                                    "elasticloadbalancing:*",
                                    "autoscaling:*",
                                    "cloudwatch:*",
                                    "s3:*",
                                    "sns:*",
                                    "cloudformation:*",
                                    "rds:*",
                                    "sqs:*",
                                    "ecs:*"
                                ],
                                "Resource": "resource_ARN"
                            }
                        ]
                    }
                }
            ]
        }
    },
    "AppPipeline": {
        "Type": "AWS::CodePipeline::Pipeline",
        "Properties": {
            "Name": "s3-events-pipeline",
            "RoleArn": {
                "Fn::GetAtt": [
                    "CodePipelineServiceRole",
                    "Arn"
                ]
            },
            "Stages": [
                {
                    "Name": "Source",
                    "Actions": [
                        {
                            "Name": "SourceAction",
                            "ActionTypeId": {
                                "Category": "Source",
                                "Owner": "AWS",
                                "Version": 1,
                                "Provider": "S3"
                            },
                            "OutputArtifacts": [
```

```
                        {
                            "Name": "SourceOutput"
                        }
                    ],
                    "Configuration": {
                        "S3Bucket": {
                            "Ref": "SourceBucket"
                        },
                        "S3ObjectKey": {
                            "Ref": "SourceObjectKey"
                        },
                        "PollForSourceChanges": false
                    },
                    "RunOrder": 1
                }
            ]
        },
        {
            "Name": "Beta",
            "Actions": [
                {
                    "Name": "BetaAction",
                    "InputArtifacts": [
                        {
                            "Name": "SourceOutput"
                        }
                    ],
                    "ActionTypeId": {
                        "Category": "Deploy",
                        "Owner": "AWS",
                        "Version": 1,
                        "Provider": "CodeDeploy"
                    },
                    "Configuration": {
                        "ApplicationName": {
                            "Ref": "ApplicationName"
                        },
                        "DeploymentGroupName": {
                            "Ref": "BetaFleet"
                        }
                    },
                    "RunOrder": 1
                }
            ]
```

```
                    }
                ],
                "ArtifactStore": {
                    "Type": "S3",
                    "Location": {
                        "Ref": "CodePipelineArtifactStoreBucket"
                    }
                }
            }
        },
        "EventRole": {
            "Type": "AWS::IAM::Role",
            "Properties": {
                "AssumeRolePolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {
                                "Service": [
                                    "events.amazonaws.com"
                                ]
                            },
                            "Action": "sts:AssumeRole"
                        }
                    ]
                },
                "Path": "/",
                "Policies": [
                    {
                        "PolicyName": "eb-pipeline-execution",
                        "PolicyDocument": {
                            "Version": "2012-10-17",
                            "Statement": [
                                {
                                    "Effect": "Allow",
                                    "Action": "codepipeline:StartPipelineExecution",
                                    "Resource": {
                                        "Fn::Join": [
                                            "",
                                            [
                                                "arn:aws:codepipeline:",
                                                {
                                                    "Ref": "AWS::Region"
```

```
                                        },
                                        ":",
                                        {
                                            "Ref": "AWS::AccountId"
                                        },
                                        ":",
                                        {
                                            "Ref": "AppPipeline"
                                        }
                                    ]
                                ]
                            }
                        }
                    ]
                }
            }
        ]
    }
},
"EventRule": {
    "Type": "AWS::Events::Rule",
    "Properties": {
        "EventPattern": {
            "source": [
                "aws.s3"
            ],
            "detail-type": [
                "AWS API Call via CloudTrail"
            ],
            "detail": {
                "eventSource": [
                    "s3.amazonaws.com"
                ],
                "eventName": [
                    "PutObject",
                    "CompleteMultipartUpload"
                ],
                "resources": {
                    "ARN": [
                        {
                            "Fn::Join": [
                                "",
                                [
                                    {
```

```
                                        "Fn::GetAtt": [
                                            "SourceBucket",
                                            "Arn"
                                        ]
                                    },
                                    "/",
                                    {
                                        "Ref": "SourceObjectKey"
                                    }
                                ]
                            ]
                        }
                    ]
                }
            }
        },
        "Targets": [
            {
                "Arn": {
                    "Fn::Join": [
                        "",
                        [
                            "arn:aws:codepipeline:",
                            {
                                "Ref": "AWS::Region"
                            },
                            ":",
                            {
                                "Ref": "AWS::AccountId"
                            },
                            ":",
                            {
                                "Ref": "AppPipeline"
                            }
                        ]
                    ]
                },
                "RoleArn": {
                    "Fn::GetAtt": [
                        "EventRole",
                        "Arn"
                    ]
                },
                "Id": "codepipeline-AppPipeline"
```

```
                    }
                ]
            }
        }
    },
    "Outputs" : {
        "SourceBucketARN" : {
            "Description" : "S3 bucket ARN that Cloudtrail will use",
            "Value" : { "Fn::GetAtt": ["SourceBucket", "Arn"] },
            "Export" : {
                "Name" : "SourceBucketARN"
            }
        }
    }
}


...
```

GitHub (OAuth アプリ経由) ソースアクションのポーリングパイプラインを接続に移行する

GitHub (OAuth アプリ経由) ソースアクションを、外部リポジトリの接続を使用するように移行できます。これは、GitHub (OAuth アプリ経由) ソースアクションを含むパイプラインで推奨される変更検出方法です。

GitHub (OAuth アプリ経由) のソースアクションを含むパイプラインでは、GitHub (GitHub アプリ経由) のアクションを使用するようにパイプラインを修正し、 AWS CodeConnectionsを通じて変更検出を自動化することをお勧めします。接続の使用の詳細については、「GitHub コネクション」を参照してください。

GitHub (コンソール) への接続を作成する

コンソールを使用して、GitHub への接続を作成できます。

ステップ 1: GitHub (OAuth アプリ経由) アクションを置き換える

パイプライン編集ページを使用して、GitHub (OAuth アプリ経由) アクションを GitHub (GitHub アプリ経由) アクションに置き換えます。

GitHub (OAuth アプリ経由) アクションを置き換えるには

1. CodePipeline コンソールにサインインします。
2. パイプラインを選択し、[編集] を選択します。ソースステージで、[ステージを編集] を選択します。アクションを更新することを推奨するメッセージが表示されます。
3. [アクションプロバイダー] で、[GitHub (GitHub アプリ経由)] を選択します。
4. 次のいずれかを行います。
   - [接続] でプロバイダへの接続をまだ作成していない場合は、[GitHub への接続] を選択します。ステップ 2: GitHub への接続を作成するに進みます。
   - [接続] でプロバイダへの接続を既に作成している場合は、その接続を選択します。ステップ 3： 接続のソースアクションを保存するに進みます。

ステップ 2 ：GitHub への接続を作成する

接続の作成を選択した後、[Connect to GitHub] ページが表示されます。

GitHub への接続を作成するには

1. [GitHub connection settings] で、[Connection name] に接続名が表示されます。

   [GitHub Apps] で、アプリケーションのインストールを選択するか、[Install a new app] (新しいアプリケーションをインストールする) を選択してアプリケーションを作成します。

   > **Note**
   >
   > 特定のプロバイダーへのすべての接続に対してアプリを 1 つインストールします。GitHub アプリをすでにをインストールしている場合は、これを選択してこのステップをスキップしてください。

2. GitHub の認可ページが表示されたら、認証情報を使用してログインし、続行を選択します。
3. アプリのインストールページで、 AWS CodeStar アプリが GitHub アカウントに接続しようとしていることを示すメッセージが表示されます。

> **Note**
>
> アプリは、GitHub アカウントごとに 1 回だけインストールします。アプリをインストール済みである場合は、Configure (設定) をクリックしてアプリのインストールの変更ページに進むか、戻るボタンでコンソールに戻ることができます。

4. [AWS CodeStarのインストール] ページで、[インストール] を選択します。
5. [Connect to GitHub] ページで、新規インストールの接続 ID が GitHub Apps に表示されます。接続 を選択します。

ステップ 3: GitHub のソースアクションを保存する

[アクションを編集] というページで更新を実行し、新しいソースアクションを保存します。

GitHub のソースアクションを保存するには

1. [リポジトリ] で、サードパーティーのリポジトリの名前を入力します。[ブランチ] で、パイプラインでソースの変更を検出するブランチを入力します。

> **Note**
>
> [Repository] で、例に示すように `owner-name/repository-name` を入力します。
>
> ```
> my-account/my-repository
> ```

2. [Output artifact format (出力アーティファクトのフォーマット)] で、アーティファクトのフォーマットを選択します。

   - デフォルトのメソッドを使用して GitHub アクションからの出力アーティファクトを保存するには、CodePipeline default を選択します。アクションは、Bitbucket リポジトリからファイルにアクセスし、パイプラインアーティファクトストアの ZIP ファイルにアーティファクトを保存します。
   - リポジトリへの URL 参照を含む JSON ファイルを保存して、ダウンストリームのアクションで Git コマンドを直接実行できるようにするには、[Full clone (フルクローン)] を選択します。このオプションは、CodeBuild ダウンストリームアクションでのみ使用できます。

このオプションを選択した場合は、Bitbucket、GitHub、GitHub Enterprise Server、または GitLab.com に接続するための CodeBuild GitClone アクセス許可を追加します。 で示されるように CodeBuild プロジェクトサービスロールの権限を更新する必要があります。フルクローン オプションの使い方を紹介したチュートリアルは、チュートリアル: CodeCommit パイプラインソースで完全なクローンを使用する をご覧ください。

3. 出力アーティファクト の場合、SourceArtifact のようにこのアクションの出力アーティファクトの名前を保持できます。[Done] を選択して、[アクションを編集] ページを閉じます。
4. [Done] を選択して、ステージの編集ページを閉じます。[Save] を選択して、パイプラインの編集ページを閉じます。

GitHub (CLI) への接続を作成する

AWS Command Line Interface (AWS CLI) を使用して GitHub への接続を作成できます。

これを行うには、create-connection コマンドを使用します。

> **Important**
>
> AWS CLI または を通じて作成された接続 AWS CloudFormation は、デフォルトで PENDINGステータスです。CLI または との接続を作成したら CloudFormation、コンソールを使用して接続を編集し、ステータスを にしますAVAILABLE。

GitHub への接続を作成するには

1. ターミナル (Linux/macOS/Unix) またはコマンドプロンプト (Windows) を開きます。を使用して create-connection コマンド AWS CLI を実行し、接続--connection-nameの --provider-typeと を指定します。この例では、サードパーティープロバイダー名は GitHub で、指定された接続名は MyConnection です。

```
aws codeconnections create-connection --provider-type GitHub --connection-name
 MyConnection
```

成功した場合、このコマンドは次のような接続 ARN 情報を返します。

```
{
```

```
    "ConnectionArn": "arn:aws:codeconnections:us-west-2:account_id:connection/
aEXAMPLE-8aad-4d5d-8878-dfcab0bc441f"
}
```

2. コンソールを使用して接続を完了します。

GitHub (OAuth アプリ経由) ソースアクションのポーリングパイプラインを Webhook に移行する

パイプラインを移行して、Webhook を使用して GitHub ソースリポジトリ内の変更を検出するようにできます。この Webhook への移行は GitHub (OAuth アプリ経由) アクションのみが対象です。

- コンソール: ポーリングパイプラインを Webhook に移行する (GitHub (OAuth アプリ経由) ソースアクション) (コンソール)
- CLI: ポーリングパイプラインを Webhook に移行する (GitHub (OAuth アプリ経由) ソースアクション) (CLI)
- CloudFormation: プッシュイベントのパイプラインを更新する (GitHub (OAuth アプリ経由) ソースアクション) (CloudFormation テンプレート)

> **Important**
>
> CodePipeline ウェブフックを作成するときは、独自の認証情報を使用したり、複数のウェブフック間で同じシークレットトークンを再利用したりしないでください。セキュリティを最適化するには、作成するウェブフックごとに一意のシークレットトークンを生成します。シークレットトークンは、ユーザーが指定する任意の文字列で、ウェブフックペイロードの整合性と信頼性を保護するために、CodePipeline に送信するウェブフックペイロードを GitHub で計算して署名するために使用します。独自の認証情報を使用したり、複数のウェブフック間で同じトークンを再利用したりすると、セキュリティの脆弱性につながる可能性があります。

ポーリングパイプラインを Webhook に移行する (GitHub (OAuth アプリ経由) ソースアクション) (コンソール)

GitHub (OAuth アプリ経由) のソースアクションでは、CodePipeline コンソールを使用してパイプラインを更新し、ウェブフックを使用して GitHub ソースリポジトリの変更を検出できます。

ポーリング (定期的なチェック) を使用しているパイプラインを EventBridge のイベントを使用するように編集するには、次の手順に従います。パイプラインを作成する場合は、「パイプライン、ステージ、アクションを作成する」を参照してください。

コンソールを使用すると、パイプラインの `PollForSourceChanges` パラメータが変更されます。GitHub ウェブフックが作成され、登録されます。

パイプラインソースステージを編集するには

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

   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。
2. [名前] で、編集するパイプラインの名前を選択します。これにより、パイプラインの詳細ビューが開いて、パイプラインの各ステージの各アクションの状態などがわかります。
3. パイプライン詳細ページで、[編集] を選択します。
4. [Edit (編集)] ステージで、ソースアクションの編集アイコンを選択します。
5. [Change detection options (変更検出オプション)] を展開し、[Use Amazon CloudWatch Events to automatically start my pipeline when a change occurs (recommended)] を選択します。

   CodePipeline が GitHub でソースの変更を検出するためのウェブフックを作成することを知らせるメッセージが表示されます。 AWS CodePipeline がウェブフックを作成します。以下のオプションでオプトアウトできます。[更新] を選択します。CodePipeline では、ウェブフックのほかに以下が作成されます。

   - シークレット。ランダムに生成され、GitHub への接続を承認するために使用されます。
   - ウェブフック URL。リージョンのパブリックエンドポイントを使用して生成されます。

   CodePipeline は、ウェブフック を GitHub に登録します。これにより、レポジトリのイベントを受信するための URL がサブスクライブされます。
6. パイプラインの編集が終わったら、[パイプラインの変更を保存] を選択して概要ページに戻ります。

   パイプラインに対して作成されるウェブフックの名前を示すメッセージが表示されます。[保存して続行] を選択します。
7. アクションをテストするには、 を使用して変更をリリース AWS CLI し、パイプラインのソースステージで指定されたソースに変更をコミットします。

ポーリングパイプラインを Webhook に移行する (GitHub (OAuth アプリ経由) ソースアクション) (CLI)

ウェブフックを使用するために定期的なチェックを使用しているパイプラインを編集するには、次の手順を使用します。パイプラインを作成する場合は、「パイプライン、ステージ、アクションを作成する」を参照してください。

イベント駆動型パイプラインを構築するには、パイプラインの PollForSourceChanges パラメータを編集してから、以下のリソースを手動で作成します。

- GitHub のウェブフックと承認パラメータ

ウェブフックを作成して登録するには

> **Note**
>
> CLI または を使用してパイプライン CloudFormation を作成し、ウェブフックを追加する場合は、定期的なチェックを無効にする必要があります。定期的なチェックを無効にするには、以下の最終的な手順に詳述するとおり、PollForSourceChanges パラメータを明示的に追加して false に設定する必要があります。それ以外の場合、CLI または CloudFormation パイプラインのデフォルトはPollForSourceChangesデフォルトで true になり、パイプライン構造の出力には表示されません。PollForSourceChanges のデフォルトの詳細については、「PollForSourceChanges パラメータの有効な設定」を参照してください。

1. テキストエディタで、作成するウェブフックの JSON ファイルを作成して保存します。「my-webhook」という名前のウェブフックには、このサンプルを使用します。

```
{
    "webhook": {
        "name": "my-webhook",
  "targetPipeline": "pipeline_name",
  "targetAction": "source_action_name",
  "filters": [{
      "jsonPath": "$.ref",
      "matchEquals": "refs/heads/{Branch}"
  }],
  "authentication": "GITHUB_HMAC",
  "authenticationConfiguration": {
```

```
      "SecretToken": "secret"
  }
    }
}
```

2. put-webhook コマンドを呼び出し、--cli-input および --region パラメータを含めます。

   次のサンプルコマンドは、webhook_json JSON ファイルで ウェブフックを作成します。

```
aws codepipeline put-webhook --cli-input-json file://webhook_json.json --region
 "eu-central-1"
```

3. この例に示す出力では、my-webhook という名前のウェブフックに対して URL と ARN が返されます。

```
{
    "webhook": {
        "url": "https://webhooks.domain.com/
trigger111111111EXAMPLE1111111111111111",
        "definition": {
            "authenticationConfiguration": {
                "SecretToken": "secret"
            },
            "name": "my-webhook",
            "authentication": "GITHUB_HMAC",
            "targetPipeline": "pipeline_name",
            "targetAction": "Source",
            "filters": [
                {
                    "jsonPath": "$.ref",
                    "matchEquals": "refs/heads/{Branch}"
                }
            ]
        },
        "arn": "arn:aws:codepipeline:eu-central-1:ACCOUNT_ID:webhook:my-webhook"
    },
    "tags": [{
      "key": "Project",
      "value": "ProjectA"
    }]
}
```

この例では、ウェブフックにProjectタグキーとProjectA値を含めることで、ウェブフックにタグ付けを追加します。CodePipeline の リソースのタグ付けの詳細については、リソースのタグ付け を参照してください。

4. register-webhook-with-third-party コマンドを呼び出し、--webhook-name パラメータを含めます。

   次のサンプルコマンドは、「my-webhook」という名前のウェブフックを登録します。

   ```
   aws codepipeline register-webhook-with-third-party --webhook-name my-webhook
   ```

パイプラインの PollForSourceChanges パラメータを編集するには

> **Important**
>
> このメソッドを使用してパイプラインを作成すると、PollForSourceChanges パラメータはデフォルトで true になります (ただし、明示的に false に設定した場合は除きます)。イベントベースの変更検出を追加する場合は、このパラメータを出力に追加する必要があります。ポーリングを無効にするには、このパラメータを false に設定します。そうしないと、1 つのソース変更に対してパイプラインが 2 回起動されます。詳細については、「PollForSourceChanges パラメータの有効な設定」を参照してください

1. get-pipeline コマンドを実行して、パイプライン構造を JSON ファイルにコピーします。例えば、MyFirstPipeline という名前のパイプラインの場合は、以下のコマンドを入力します。

   ```
   aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
   ```

   このコマンドは何も返しませんが、作成したファイルは、コマンドを実行したディレクトリにあります。

2. 任意のプレーンテキストエディタで JSON ファイルを開き、以下に示しているように、PollForSourceChanges パラメータを変更または追加してソースステージを編集します。この例では、UserGitHubRepo という名前のリポジトリで、パラメータを false に設定します。

   この変更を行う理由 このパラメータを変更すると、定期的なチェックがオフになるため、イベントベースの変更検出のみ使用することができます。

```
"configuration": {
    "Owner": "name",
    "Repo": "UserGitHubRepo",
    "PollForSourceChanges": "false",
    "Branch": "main",
    "OAuthToken": "****"
},
```

3. get-pipeline コマンドを使用して取得されたパイプライン構造を操作している場合、ファイルから metadata 行を削除して JSON ファイルの構造を編集する必要があります。それ以外の場合は、update-pipeline コマンドで使用することはできません。JSON ファイルのパイプライン構造から "metadata" セクションを削除します ({ } 行と、"created"、"pipelineARN"、および "updated" フィールド)。

   例えば、構造から以下の行を削除します。

```
"metadata": {
    "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
    "created": "date",
    "updated": "date"
},
```

   ファイルを保存します。

4. 変更を適用するには、以下のように、パイプライン JSON ファイルを指定して、 update-pipeline コマンドを実行します。

> **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 codepipeline update-pipeline --cli-input-json file://pipeline.json
```

   このコマンドは、編集したパイプラインの構造全体を返します。

> **Note**
>
> update-pipeline コマンドは、パイプラインを停止します。update-pipeline コマンドを実行したときにパイプラインによりリビジョンが実行されている場合、その実行は停止します。更新されたパイプラインによりそのリビジョンを実行するには、パイプラインを手動で開始する必要があります。パイプラインを手動で開始するには start-pipeline-execution コマンドを使用します。

プッシュイベントのパイプラインを更新する (GitHub (OAuth アプリ経由) ソースアクション) (CloudFormation テンプレート)

以下の手順に従って、GitHub ソースを含むパイプラインを、定期的なチェック (ポーリング) から、ウェブフックを使用したイベントベースの変更検出に更新します。

を使用してイベント駆動型パイプラインを構築するには AWS CodeCommit、パイプラインの `PollForSourceChanges`パラメータを編集し、GitHub ウェブフックリソースをテンプレートに追加します。

CloudFormation を使用してパイプラインを作成および管理する場合、テンプレートには次のようなコンテンツがあります。

> **Note**
>
> ソースステージ内の `PollForSourceChanges` 設定プロパティを書き留めます。テンプレートにプロパティが含まれていない場合、`PollForSourceChanges` はデフォルトで `true` に設定されます。

YAML

```
Resources:
  AppPipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      Name: github-polling-pipeline
      RoleArn:
        !GetAtt CodePipelineServiceRole.Arn
```

```
      Stages:
        -
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: ThirdParty
                Version: 1
                Provider: GitHub
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                Owner: !Ref GitHubOwner
                Repo: !Ref RepositoryName
                Branch: !Ref BranchName
                OAuthToken:
 {{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}
 PollForSourceChanges: true
              RunOrder: 1


...
```

JSON

```
        "AppPipeline": {
            "Type": "AWS::CodePipeline::Pipeline",
            "Properties": {
                "Name": "github-polling-pipeline",
                "RoleArn": {
                    "Fn::GetAtt": [
                        "CodePipelineServiceRole",
                        "Arn"
                    ]
                },
                "Stages": [
                    {
                        "Name": "Source",
                        "Actions": [
                            {
                                "Name": "SourceAction",
```

```
                                "ActionTypeId": {
                                    "Category": "Source",
                                    "Owner": "ThirdParty",
                                    "Version": 1,
                                    "Provider": "GitHub"
                                },
                                "OutputArtifacts": [
                                    {
                                        "Name": "SourceOutput"
                                    }
                                ],
                                "Configuration": {
                                    "Owner": {
                                        "Ref": "GitHubOwner"
                                    },
                                    "Repo": {
                                        "Ref": "RepositoryName"
                                    },
                                    "Branch": {
                                        "Ref": "BranchName"
                                    },
                                    "OAuthToken":
 "{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}",
                                    "PollForSourceChanges": true
                                },
                                "RunOrder": 1
                            }
                        ]
                    },


...
```

テンプレートにパラメータを追加してウェブフックを作成するには

を使用して認証情報 AWS Secrets Manager を保存することを強くお勧めします。Secrets Manager を使用する場合は、Secrets Manager でシークレットパラメータをすでに設定して保存しておく必要があります。この例では、ウェブフックの GitHub 認証情報に Secrets Manager への動的な参照を使用します。詳細については、「動的な参照を使用してテンプレート値を指定する」を参照してください。

> **Important**
>
> シークレットパラメータを渡すときは、値をテンプレートに直接入力しないでください。値はプレーンテキストとしてレンダリングされるため、読み取り可能です。セキュリティ上の理由から、 CloudFormation テンプレートにプレーンテキストを使用して認証情報を保存しないでください。

CLI または を使用してパイプライン CloudFormation を作成し、ウェブフックを追加する場合は、定期的なチェックを無効にする必要があります。

> **Note**
>
> 定期的なチェックを無効にするには、以下の最終的な手順に詳述するとおり、PollForSourceChanges パラメータを明示的に追加して false に設定する必要があります。それ以外の場合、CLI または CloudFormation パイプラインのデフォルトはPollForSourceChangesデフォルトで true になり、パイプライン構造の出力には表示されません。PollForSourceChanges のデフォルトの詳細については、「PollForSourceChanges パラメータの有効な設定」を参照してください。

1. テンプレートの Resources に、パラメータを追加します。

   YAML

   ```
   Parameters:
           GitHubOwner:
             Type: String

   ...
   ```

   JSON

   ```
   {
    "Parameters": {
     "BranchName": {
      "Description": "GitHub branch name",
      "Type": "String",
      "Default": "main"
   ```

```
  },
  "GitHubOwner": {
   "Type": "String"
  },

...
```

2. AWS::CodePipeline::Webhook CloudFormation リソースを使用してウェブフックを追加します。

> **Note**
>
> 指定した TargetAction は、パイプラインで定義したソースアクションの Name プロパティと一致する必要があります。

RegisterWithThirdParty が true に設定されている場合は、OAuthToken に関連付けられたユーザーが GitHub で必要なスコープを設定できることを確認してください。トークンとウェブフックには、以下の GitHub スコープが必要となります。

- repo - パブリックおよびプライベートリポジトリからパイプラインにアーティファクトを読み込んでプルする完全制御に使用されます。
- admin:repo_hook - リポジトリフックの完全制御に使用されます。

それ以外の場合、GitHub から 404 が返されます。返される 404 の詳細については、「https://help.github.com/articles/about-webhooks」を参照してください

YAML

```
  AppPipelineWebhook:
    Type: AWS::CodePipeline::Webhook
    Properties:
      Authentication: GITHUB_HMAC
      AuthenticationConfiguration:
        SecretToken:
{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}
      Filters:
        -
          JsonPath: "$.ref"
          MatchEquals: refs/heads/{Branch}
```

```
      TargetPipeline: !Ref AppPipeline
      TargetAction: SourceAction
      Name: AppPipelineWebhook
      TargetPipelineVersion: !GetAtt AppPipeline.Version
      RegisterWithThirdParty: true


...
```

JSON

```
"AppPipelineWebhook": {
    "Type": "AWS::CodePipeline::Webhook",
    "Properties": {
        "Authentication": "GITHUB_HMAC",
        "AuthenticationConfiguration": {
            "SecretToken":
 "{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}"
        },
        "Filters": [{
            "JsonPath": "$.ref",
            "MatchEquals": "refs/heads/{Branch}"
        }],
        "TargetPipeline": {
            "Ref": "AppPipeline"
        },
        "TargetAction": "SourceAction",
        "Name": "AppPipelineWebhook",
        "TargetPipelineVersion": {
            "Fn::GetAtt": [
              "AppPipeline",
              "Version"
            ]
        },
        "RegisterWithThirdParty": true
    }
},

...
```

3. 更新されたテンプレートをローカルコンピュータに保存し、 CloudFormation コンソールを開きます。
4. スタックを選択し、[既存スタックの変更セットの作成] を選択します。

5. テンプレートをアップロードし、 CloudFormationに示された変更を表示します。これらがスタックに加えられる変更です。新しいリソースがリストに表示されています。
6. [実行] を選択してください。

パイプラインの PollForSourceChanges パラメータを編集するには

> **Important**
>
> このメソッドを使用してパイプラインを作成すると、PollForSourceChanges パラメータはデフォルトで true になります (ただし、明示的に false に設定した場合は除きます)。イベントベースの変更検出を追加する場合は、このパラメータを出力に追加する必要があります。ポーリングを無効にするには、このパラメータを false に設定します。そうしないと、1 つのソース変更に対してパイプラインが 2 回起動されます。詳細については、「PollForSourceChanges パラメータの有効な設定」を参照してください

- テンプレートで、PollForSourceChanges を false に変更します。パイプライン定義に PollForSourceChanges が含まれていなかった場合は、追加して false に設定します。

  この変更を行う理由 このパラメータを false に変更すると、定期的チェックがオフになるため、イベントベースの変更検出のみ使用することができます。

  YAML

```
        Name: Source
        Actions:
          -
            Name: SourceAction
            ActionTypeId:
              Category: Source
              Owner: ThirdParty
              Version: 1
              Provider: GitHub
            OutputArtifacts:
              - Name: SourceOutput
            Configuration:
              Owner: !Ref GitHubOwner
              Repo: !Ref RepositoryName
              Branch: !Ref BranchName
```

```
                OAuthToken:
{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}
                PollForSourceChanges: false
              RunOrder: 1
```

JSON

```
{
   "Name": "Source",
   "Actions": [{
 "Name": "SourceAction",
 "ActionTypeId": {
     "Category": "Source",
     "Owner": "ThirdParty",
     "Version": 1,
     "Provider": "GitHub"
},
"OutputArtifacts": [{
    "Name": "SourceOutput"
}],
"Configuration": {
    "Owner": {
  "Ref": "GitHubOwner"
    },
    "Repo": {
  "Ref": "RepositoryName"
    },
    "Branch": {
        "Ref": "BranchName"
    },
    "OAuthToken":
"{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}",
    PollForSourceChanges: false
},
"RunOrder": 1
   }]
```

Example

これらのリソースを で作成すると CloudFormation、定義されたウェブフックが指定された GitHub リポジトリに作成されます。パイプラインはコミット時にトリガーされます。

YAML

```
Parameters:
  GitHubOwner:
    Type: String

Resources:
  AppPipelineWebhook:
    Type: AWS::CodePipeline::Webhook
    Properties:
      Authentication: GITHUB_HMAC
      AuthenticationConfiguration:
        SecretToken: {{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}
      Filters:
        -
          JsonPath: "$.ref"
          MatchEquals: refs/heads/{Branch}
      TargetPipeline: !Ref AppPipeline
      TargetAction: SourceAction
      Name: AppPipelineWebhook
      TargetPipelineVersion: !GetAtt AppPipeline.Version
      RegisterWithThirdParty: true
  AppPipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      Name: github-events-pipeline
      RoleArn:
        !GetAtt CodePipelineServiceRole.Arn
      Stages:
        -
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: ThirdParty
                Version: 1
                Provider: GitHub
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                Owner: !Ref GitHubOwner
                Repo: !Ref RepositoryName
```

```
                Branch: !Ref BranchName
                OAuthToken:
 {{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}
                PollForSourceChanges: false
              RunOrder: 1

...
```

JSON

```
{
    "Parameters": {
        "BranchName": {
            "Description": "GitHub branch name",
            "Type": "String",
            "Default": "main"
        },
        "RepositoryName": {
            "Description": "GitHub repository name",
            "Type": "String",
            "Default": "test"
        },
        "GitHubOwner": {
            "Type": "String"
        },
        "ApplicationName": {
            "Description": "CodeDeploy application name",
            "Type": "String",
            "Default": "DemoApplication"
        },
        "BetaFleet": {
            "Description": "Fleet configured in CodeDeploy",
            "Type": "String",
            "Default": "DemoFleet"
        }
    },
    "Resources": {

...

        },
        "AppPipelineWebhook": {
            "Type": "AWS::CodePipeline::Webhook",
```

```
            "Properties": {
                "Authentication": "GITHUB_HMAC",
                "AuthenticationConfiguration": {
                    "SecretToken": {

"{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}"
                    }
                },
                "Filters": [
                    {
                        "JsonPath": "$.ref",
                        "MatchEquals": "refs/heads/{Branch}"
                    }
                ],
                "TargetPipeline": {
                    "Ref": "AppPipeline"
                },
                "TargetAction": "SourceAction",
                "Name": "AppPipelineWebhook",
                "TargetPipelineVersion": {
                    "Fn::GetAtt": [
                        "AppPipeline",
                        "Version"
                    ]
                },
                "RegisterWithThirdParty": true
            }
        },
        "AppPipeline": {
            "Type": "AWS::CodePipeline::Pipeline",
            "Properties": {
                "Name": "github-events-pipeline",
                "RoleArn": {
                    "Fn::GetAtt": [
                        "CodePipelineServiceRole",
                        "Arn"
                    ]
                },
                "Stages": [
                    {
                        "Name": "Source",
                        "Actions": [
                            {
                                "Name": "SourceAction",
```

```
                                "ActionTypeId": {
                                    "Category": "Source",
                                    "Owner": "ThirdParty",
                                    "Version": 1,
                                    "Provider": "GitHub"
                                },
                                "OutputArtifacts": [
                                    {
                                        "Name": "SourceOutput"
                                    }
                                ],
                                "Configuration": {
                                    "Owner": {
                                        "Ref": "GitHubOwner"
                                    },
                                    "Repo": {
                                        "Ref": "RepositoryName"
                                    },
                                    "Branch": {
                                        "Ref": "BranchName"
                                    },
                                    "OAuthToken":
 "{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}",
                                    "PollForSourceChanges": false
                                },
                                "RunOrder": 1

...
```

CodePipeline サービスロールを作成する

パイプラインの作成時に、サービスロールを作成するか、既存のサービスロールを使用します。

CodePipeline コンソールまたは を使用して AWS CLI 、CodePipeline サービスロールを作成できます。サービスロールは、パイプラインを作成するために必要であり、パイプラインは必ず作成元のサービスロールに関連付けられます。

CLI AWS を使用してパイプラインを作成する前に、パイプラインの CodePipeline サービスロールを作成する必要があります。サービスロールとポリシーが指定されている CloudFormation テンプレートの例については、「」のチュートリアルを参照してくださいチュートリアル: AWS CloudFormation デプロイアクションの変数を使用するパイプラインを作成する。

サービスロールは AWS マネージドロールではありませんが、最初にパイプライン作成用に作成され、新しいアクセス許可がサービスロールポリシーに追加されると、パイプラインのサービスロールを更新する必要がある場合があります。サービスロールを使用してパイプラインを作成すると、このパイプラインに別のサービスロールを適用することはできません。推奨されるポリシーをサービスロールにアタッチします。

サービスロールの詳細については、「CodePipeline サービスロールを管理する」を参照してください。

CodePipeline サービスロールを作成する (コンソール)

コンソールを使用してパイプラインを作成すると、パイプライン作成ウィザードで CodePipeline サービスロールを作成します。

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

   [パイプラインの作成] を選択し、パイプライン作成ウィザードの [ステップ 1: パイプラインの設定を選択する] を完了します。

   > **Note**
   >
   > パイプラインを作成したら、その名前を変更することはできません。その他の制限についての詳細については、「AWS CodePipeline のクォータ」を参照してください。

2. [サービスロール] で、[新しいサービスロール] をクリックして、CodePipeline の IAM での新しいサービスロールの作成を許可します。
3. パイプラインの作成を完了します。パイプラインのサービスロールを使用して IAM ロールを一覧表示できます。また、`get-pipeline` CLI で AWS コマンドを実行して、パイプラインに関連付けられているサービスロールの ARN を表示できます。

CodePipeline サービスロールを作成する (CLI)

CLI または AWS を使用してパイプラインを作成する前に CloudFormation、パイプラインの CodePipeline サービスロールを作成し、サービスロールポリシーと信頼ポリシーをアタッチする必要があります。CLI を使用してサービスロールを作成するには、以下の手順を使用して、まず CLI コマンドを実行するディレクトリに信頼ポリシー JSON とロールポリシー JSON を別々のファイルとして作成します。

> **Note**
>
> 管理者ユーザーのみにサービスロールの作成を許可することをお勧めします。ロールの作成とポリシーのアタッチを許可されたユーザーは、自分のアクセス許可をエスカレートできます。代わりに、彼らが必要とするロールの作成のみを許可したポリシーを作成するか、彼らの代わりに、管理者にサービスロールを作成させます。

1. ターミナルウィンドウで以下のコマンドを入力して、TrustPolicy.json という名前のファイルを作成し、そのファイルにロールポリシー JSON を貼り付けます。この例では VIM を使用します。

```
vim TrustPolicy.json
```

2. そのファイルに次の JSON を貼り付けます。

   JSON

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

   ファイルを保存して終了するには、次の VIM コマンドを入力します。

```
:wq
```

3. ターミナルウィンドウで以下のコマンドを入力して、RolePolicy.json という名前のファイルを作成し、そのファイルにロールポリシー JSON を貼り付けます。この例では VIM を使用します。

```
vim RolePolicy.json
```

4. JSON ポリシーをファイルに貼り付けます。「CodePipeline のサービスロールポリシー」に記載されている最小サービスロールポリシーを使用します。さらに、使用する予定のアクションに基づいて、適切なアクセス許可をサービスロールに追加します。アクションのリスト、および各アクションに必要なサービスロールのアクセス許可へのリンクについては、「CodePipeline サービスロールにアクセス許可を追加する」を参照してください。

   Resource フィールドのリソースレベルまで、アクセス許可を可能な限り絞り込みます。

   ファイルを保存して終了するには、次の VIM コマンドを入力します。

```
:wq
```

5. 次のコマンドを入力して、そのロールを作成し、ロールポリシーをアタッチします。ポリシー名の形式は、通常、ロール名の形式と同じです。この例では、ロール名MyRoleと、別のファイルとしてTrustPolicy作成されたポリシーを使用します。

```
aws iam create-role --role-name MyRole --assume-role-policy-document file://
TrustPolicy.json
```

6. 次のコマンドを入力してロールポリシーを作成し、ロールにアタッチします。ポリシー名の形式は、通常、ロール名の形式と同じです。この例では、ロール名MyRoleと、別のファイルとしてMyRole作成されたポリシーを使用します。

```
aws iam put-role-policy --role-name MyRole --policy-name RolePolicy --policy-
document file://RolePolicy.json
```

7. 作成されたロール名と信頼ポリシーを表示するには、MyRole という名前のロールに対して次のコマンドを入力します。

```
aws iam get-role --role-name MyRole
```

8. CLI または でパイプラインを作成するときは、サービスロール ARN AWS を使用します CloudFormation。

リソースのタグ付け

タグは、 AWS リソース AWS に割り当てるカスタム属性ラベルです。各 AWS タグには 2 つの部分があります。

- タグキー (CostCenter、Environment、Project、Secret など)。タグキーでは、大文字と小文字が区別されます。
- タグ値と呼ばれるオプションのフィールド (111122223333、Production、チーム名など)。タグ値を省略すると、空の文字列を使用した場合と同じになります。タグキーと同様に、タグ値では大文字と小文字が区別されます。

これらを合わせて、キーと値のペアと呼ばれます。

タグは、 AWS リソースの識別と整理に役立ちます。多くの はタグ付け AWS のサービス をサポートしているため、異なるサービスのリソースに同じタグを割り当てて、リソースが関連していることを示すことができます。例えば、Amazon S3 ソースバケットに割り当てたのと同じタグをパイプラインに割り当てることができます。

タグの使用方法のヒントについては、AWS の回答ブログのAWS タグ付け戦略記事を参照してください。

CodePipeline では、以下のリソースタイプにタグを付けることができます。

- CodePipeline でパイプラインにタグ付けする
- CodePipeline でカスタムアクションにタグ付けする

AWS CLI、CodePipeline APIs、または AWS SDKsを使用して、次のことができます。

- パイプライン、カスタムアクション、ウェブフックを作成するときに、タグを追加します。
- パイプライン、カスタムアクション、ウェブフックのタグを追加、管理、削除します。

また、コンソールを使用してパイプラインのタグを追加、管理、削除することもできます。

タグを使用してリソースを識別、整理、追跡するだけでなく、IAM ポリシーのタグを使ってリソースを表示および操作できるユーザーを制御することもできます。タグベースのアクセスポリシーの例については、「タグを使用した CodePipeline リソースへのアクセスのコントロール」を参照してください。

CodePipeline でパイプラインにタグ付けする

タグは、 AWS リソースに関連付けられたキーと値のペアです。CodePipeline で作成したパイプラインにタグを適用することができます。CodePipeline リソースのタグ付け、使用例、タグのキーと値の制約、サポートされているリソースの種類については、リソースのタグ付け を参照してください。

パイプラインを作成するときに CLI を使用してタグを指定できます。コンソールまたは CLI を使用してタグを追加または削除し、パイプラインのタグの値を更新できます。各パイプラインに最大 50 個のタグを追加できます。

トピック

- パイプラインにタグ付けする (コンソール)
- パイプラインにタグ付けする (CLI)

パイプラインにタグ付けする (コンソール)

コンソールまたは CLI を使用して、リソースのタグ付けをします。パイプラインは、コンソールまたは CLI のどちらでも管理できる唯一の CodePipeline リソースです。

トピック

- パイプラインにタグを追加する (コンソール)
- パイプラインのタグを表示する (コンソール)
- パイプラインのタグを編集する (コンソール)
- パイプラインからタグを削除する (コンソール)

パイプラインにタグを追加する (コンソール)

コンソールを使用して既存のパイプラインにタグを追加します。

1. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. Pipelines ページで、タグを追加するパイプラインを選択します。
3. ナビゲーションパネルから [Settings (設定)] を選択します。
4. [Pipeline tags] で、[Edit] を選択します。

5. [Key] フィールドと [Value] フィールドに、追加するタグのセットごとにキーペアを入力します。([値] フィールドはオプションです。) 例えば、[キー] では、「**Project**」と入力します。[値] には「**ProjectA**」と入力します。
6. (オプション) [タグを追加] をクリックして行を追加し、さらにタグを入力します。
7. [Submit] を選択してください。タグは、パイプラインの設定の下に表示されます。

パイプラインのタグを表示する (コンソール)

コンソールを使用して既存のパイプラインのタグを一覧表示します。

1. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [Pipelines] ページで、タグを表示するパイプラインを選択します。
3. ナビゲーションパネルから [Settings (設定)] を選択します。
4. [Pipeline tags] で、[Key] 列と [Value] 列下のパイプラインのタグを表示します。

パイプラインのタグを編集する (コンソール)

コンソールを使用してパイプラインに追加されたタグを編集します。

1. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [Pipelines] ページで、タグを更新するパイプラインを選択します。
3. ナビゲーションパネルから [Settings (設定)] を選択します。
4. [Pipeline tags] で、[Edit] を選択します。
5. [キー] フィールドと [値] フィールドに、必要に応じて各フィールドの値を更新します。例えば、**Project** キーの場合は、[Value] で、**ProjectA** を **ProjectB** に変更します。
6. [Submit] を選択してください。

パイプラインからタグを削除する (コンソール)

コンソールを使用してパイプラインからタグを削除できます。関連付けられているリソースからタグを削除すると、そのタグが削除されます。

1. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [Pipelines] ページで、タグを削除するパイプラインを選択します。
3. ナビゲーションパネルから [Settings (設定)] を選択します。
4. [Pipeline tags] で、[Edit] を選択します。
5. 削除する各タグのキーと値の横にある [Remove tag] を選択します。
6. [Submit] を選択してください。

パイプラインにタグ付けする (CLI)

CLI を使用してリソースにタグを付けることができます。パイプラインのタグを管理するには、コンソールを使用する必要があります。

トピック

- パイプラインにタグを追加する (CLI)
- パイプラインのタグを表示する (CLI)
- パイプラインのタグを編集する (CLI)
- パイプラインからタグを削除する (CLI)

パイプラインにタグを追加する (CLI)

コンソールまたは を使用してパイプライン AWS CLI にタグを付けることができます。

作成時にタグをパイプラインに追加するには、「パイプライン、ステージ、アクションを作成する」を参照してください。

以下のステップでは、 AWS CLI の最新版を既にインストールしているか、最新版に更新しているものと想定します。詳細については、「AWS Command Line Interfaceのインストール」を参照してください。

ターミナルまたはコマンドラインで、タグを追加するパイプラインのAmazon リソースネーム (ARN)、および追加するタグのキーと値を指定して tag-resource コマンドを実行します。パイプラインに複数のタグを追加できます。例えば、*MyPipeline* というパイプラインに、*Test* のタグ値を持った *DeploymentEnvironment* というタグキーと、*true* のタグ値を持った *IscontainerBased* というタグキーの 2 つのタグを付けるとします。

```
aws codepipeline tag-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:MyPipeline --tags key=Project,value=ProjectA key=IscontainerBased,value=true
```

成功した場合、このコマンドは何も返しません。

パイプラインのタグを表示する (CLI)

を使用してパイプラインの AWS タグ AWS CLI を表示するには、次の手順に従います。タグが追加されていない場合、返されるリストは空になります。

ターミナルまたはコマンドラインで、list-tags-for-resource コマンドを実行します。例えば、arn:aws:codepipeline:*us-west-2*:*account-id*:*MyPipeline* ARN の値を持った *MyPipeline* という名前のパイプラインのタグキーとタグ値のリストを表示するとします。

```
aws codepipeline list-tags-for-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:MyPipeline
```

成功した場合、このコマンドは次のような情報を返します。

```
{
    "tags": {
        "Project": "ProjectA",
        "IscontainerBased": "true"
    }
}
```

パイプラインのタグを編集する (CLI)

を使用してパイプラインのタグ AWS CLI を編集するには、次の手順に従います。既存のキーの値を変更したり、別のキーを追加できます。次のセクションに示すように、パイプラインからタグを削除することもできます。

ターミナルまたはコマンドラインで、タグを更新するパイプラインの ARN を指定して、tag-resource コマンドを実行し、タグキーとタグ値を指定します。

```
aws codepipeline tag-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:MyPipeline --tags key=Project,value=ProjectA
```

成功した場合、このコマンドは何も返しません。

パイプラインからタグを削除する (CLI)

を使用してパイプラインからタグ AWS CLI を削除するには、次の手順に従います。関連付けられているリソースからタグを削除すると、そのタグが削除されます。

> **Note**
>
> パイプラインを削除すると、削除されたパイプラインからすべてのタグの関連付けが削除されます。パイプラインを削除する前にタグを削除する必要はありません。

ターミナルまたはコマンド行で、タグを削除するパイプラインのARNと削除するタグのタグキーを指定して、untag-resource コマンドを実行します。例えば、*MyPipeline* という名前のパイプライン上のタグキーが *Project* および *IscontainerBased* である複数のタグを削除するには、次のようにします。

```
aws codepipeline untag-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:MyPipeline --tag-keys Project IscontainerBased
```

成功した場合、このコマンドは何も返しません。パイプラインに関連付けられているタグを確認するには、list-tags-for-resource コマンドを実行します。

通知ルールの作成

通知ルールを使用すると、パイプラインの実行開始時など、重要な変更を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 CodePipeline。

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/codepipeline/ で CodePipeline コンソールを開きます。
2. [Pipelines (パイプライン)] を選択し、通知を追加するパイプラインを選択します。
3. パイプラインページで、[Notify (通知)] を選択し、次に、[Create notification rule (通知ルールを作成)] を選択します。パイプラインの [Settings (設定)] ページに移動し、[Create notification rule (通知ルールを作成)] 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、CodePipeline 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。詳細については、「パイプラインでの通知ルールのイベント」を参照してください。
7. [Targets (ターゲット)] で、次のいずれかの操作を行います。

   - 通知で使用するリソースを設定済みである場合は、[ターゲットタイプを選択] で、[チャットアプリケーションの Amazon Q Developer (Slack)] または [SNS トピック] を選択します。[ターゲットを選択] で、クライアントの名前 (チャットアプリケーションの Amazon Q Developer で設定した Slack クライアントの場合) または Amazon SNS トピックの Amazon リソースネーム (ARN) (通知に必要なポリシーが設定済みである Amazon SNS トピックの場合) を選択します。
   - 通知で使用するリソースを設定していない場合は、[Create target]、[SNS topic] の順に選択します。codestar-notifications- の後にトピックの名前を指定し、[Create] を選択します。

> **Note**
>
> - 通知ルールの作成の一環として Amazon SNS トピックを作成すると、トピックへのイベント発行を通知機能に許可するポリシーが適用されます。通知ルール用に作成したトピックを使用すると、このリソースに関する通知を受信するユーザーのみをサブスクライブできます。
> - 通知ルールの作成の一環としてチャットアプリケーションの Amazon Q Developer クライアントを作成することはできません。チャットアプリケーションの Amazon Q Developer (Slack) を選択すると、チャットアプリケーションで Amazon Q Developer でクライアントを設定するように指示するボタンが表示されます。このオプションを選択すると、チャットアプリケーションの Amazon Q Developer コンソールが開きます。詳細については、「通知とチャットアプリケーションの Amazon Q Developer との統合の設定」を参照してください。

- 既存の Amazon SNS トピックをターゲットとして使用する場合は、このトピック用の他のすべてのポリシーに加えて、AWS CodeStar Notifications に必要なポリシーを追加する必要があります。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」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、「通知とチャットアプリケーションの Amazon Q Developer との統合の設定」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、ターゲットを含めます。次の例は、ID *123456789012* の AWS アカウントで *MyDemoPipeline* という名前のパイプライン**MyNotificationRule**に という名前の通知ルールを示しています。パイプラインの実行がスタートすると、通知は完全な詳細タイプで Amazon SNS トピック *codestar-notifications-MyNotificationTopic* に送信されます：

```
{
    "Name": "MyNotificationRule",
    "EventTypeIds": [
        "codepipeline-pipeline-pipeline-execution-started"
    ],
    "Resource": "arn:aws:codebuild:us-east-2:123456789012:MyDemoPipeline",
    "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"
}
```

ソースアクションと変更検出方法

ソースアクションをパイプラインに追加すると、アクションは表で説明されている追加のリソースで動作します。

> **Note**
>
> CodeCommit および S3 ソースアクションには、設定済みの変更検出リソース (EventBridge ルール)、またはオプションを使用してソースの変更をリポジトリにポーリングする必要があります。Bitbucket、GitHub、または GitHub Enterprise Server のソースアクションを持つパイプラインの場合、ウェブフックを設定したり、デフォルトでポーリングを行う必要はありません。接続アクションは、変更検出を管理します。

ソース	その他のリソースを使用しますか？	Steps	
CloudTrail リソースを使用する Amazon S3	このソースアクションは、イベントルールと追加の CloudTrail リソースを使用します。CLI または CloudFormation を使用してこのアクションを作成する場合は、これらのリソースも作成および管理します。	パイプライン、ステージ、アクションを作成する および EventBridge と を使用する Amazon S3 ソースアクションへの接続 AWS CloudTrail を参照してください。	
CloudTrail リソースを使用しない Amazon S3	このソースアクションは、追加の CloudTrail リソースを必要とせずに、イベントルールを持つイベントに対して有効になっているバケットを使用します。CLI または CloudFormation を使用してこのアクションを作成	パイプライン、ステージ、アクションを作成する および イベントに対して有効にしたソースを使用して Amazon S3 ソースアクションに接続する を参照してください。	

ソース	その他のリソースを使用しますか？	Steps	
	する場合は、これらのリソースも作成および管理します。		
Bitbucket Cloud	このソースアクションは、コネクションリソースを使用します。	「Bitbucket Cloud への接続」を参照してください。	
AWS CodeCommit	Amazon EventBridge (推奨)。これは、コンソールで作成または編集した CodeCommit ソースを含むパイプラインのデフォルトです。	パイプライン、ステージ、アクションを作成する および CodeCommit ソースアクションと EventBridge を参照してください。	
Amazon ECR	Amazon EventBridge これは、コンソールで作成または編集した Amazon ECR ソースを含むパイプライン用にウィザードで作成されます。	「パイプライン、ステージ、アクションを作成する」および「Amazon ECR ソースアクションと EventBridge リソース」を参照してください。	
GitHub または GitHub Enterprise Cloud	このソースアクションは、コネクションリソースを使用します。	GitHub コネクション を参照してください。	
GitHub Enterprise Server	このソースアクションは、コネクションリソースとホストリソースを使用します。	GitHub Enterprise Server 接続 を参照してください。	
GitLab.com	このソースアクションは、コネクションリソースを使用します。	「GitLab.com への接続」を参照してください。	

ソース	その他のリソースを使用しますか？	Steps	
GitLab セルフマネージド	このソースアクションは、コネクションリソースとホストリソースを使用します。	GitLab セルフマネージドの接続 を参照してください。	

ポーリングを使用するパイプラインがある場合は、推奨される検出方法を使用するように更新できます。詳細については、「ポーリングパイプラインを推奨される変更検出方法に更新する」を参照してください。

接続を使用するソースアクションの変更検出をオフにするには、CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアクションの場合) を参照してください。

ソースアクションを使用してファーストパーティーソースプロバイダーに接続する

AWS CodePipeline コンソールまたは を使用して、CodeCommit や S3 などのソースアクションプロバイダー AWS CLI に接続できます。

> **Note**
>
> コンソールを使用してパイプラインを作成または編集すると、変更検出リソースが作成されます。 AWS CLI を使用してパイプラインを作成する場合は、追加のリソースを自分で作成する必要があります。詳細については、「CodeCommit ソースアクションと EventBridge」を参照してください。

トピック

- Amazon ECR ソースアクションと EventBridge リソース
- イベントに対して有効にしたソースを使用して Amazon S3 ソースアクションに接続する
- EventBridge と を使用する Amazon S3 ソースアクションへの接続 AWS CloudTrail
- CodeCommit ソースアクションと EventBridge

Amazon ECR ソースアクションと EventBridge リソース

CodePipeline で Amazon ECR のソースアクションを追加するには、次のいずれかを選択できます。

- CodePipeline コンソールの [パイプラインの作成] ウィザード (カスタムパイプラインを作成する (コンソール))、または [アクションを編集する] ページを使用して、[Amazon ECR] プロバイダオプションを選択します。このコンソールはソースが変更されたときにパイプラインを開始する EventBridge ルールを作成します。
- CLI を使用して、ECR アクションのアクション設定を追加し、次のように追加のリソースを作成します。
  - ECR でのアクション設定の例を Amazon ECR ソースアクションリファレンス で使用し、パイプラインを作成する (CLI) で表示されるようにアクションを作成します。
  - 変更検出方法のデフォルトは、ソースをポーリングすることによってパイプラインを開始します。定期的なチェックを無効にして、手動で変更検出ルールを作成することをお勧めしま

す。Amazon ECR ソースに対する EventBridge ルールを作成する (コンソール)、Amazon ECR ソースに対する EventBridge ルールを作成する (CLI)、Amazon ECR ソースの EventBridge ルールを作成する (CloudFormation テンプレート) の内、いずれかの方法を使用します。

トピック

- Amazon ECR ソースに対する EventBridge ルールを作成する (コンソール)
- Amazon ECR ソースに対する EventBridge ルールを作成する (CLI)
- Amazon ECR ソースの EventBridge ルールを作成する (CloudFormation テンプレート)

Amazon ECR ソースに対する EventBridge ルールを作成する (コンソール)

CodePipeline オペレーションで使用する EventBridge ルールを作成するには (Amazon ECR ソース)

1. Amazon EventBridge コンソールの https://console.aws.amazon.com/events/ を開いてください。
2. ナビゲーションペインの [Events] を選択してください。
3. [ルールの作成] を選択し、[イベントソース] の [サービス名] から [Elastic Container Registry (ECR)] を選択します。
4. [イベントソース] で、[イベントパターン] を選択します。

   [編集] をクリックし、次のイベントパターン例を [イベントソース] のウィンドウに貼り付ける事で、 `eb-test` のリポジトリに `cli-testing` イメージタグが追加されます。

```
{
    "detail-type": [
        "ECR Image Action"
    ],
    "source": [
        "aws.ecr"
    ],
    "detail": {
        "action-type": [
            "PUSH"
        ],
        "image-tag": [
            "latest"
        ],
        "repository-name": [
```

```
            "eb-test"
        ],
        "result": [
            "SUCCESS"
        ]
    }
}
```

> **Note**
>
> Amazon ECR イベントでサポートされているイベントパターン全体を表示するには、[Amazon ECR Events と EventBridge] または [Amazon Elastic Container Registry Events] を参照してください。

5. [保存] を選択します。

   [イベントパターンのプレビュー] ペインで、ルールを表示します。

6. [ターゲット] で、[CodePipeline] を選択します。

7. このルールによって開始するパイプラインの、パイプライン ARN を入力します。

> **Note**
>
> get-pipeline コマンドを実行した後、メタデータ出力でパイプライン ARN を見つけることができます。パイプライン ARN はこの形式で作成されます。
> arn:aws:codepipeline:*region*:*account*:pipeline-name
> パイプライン ARN の例:
> `arn:aws:codepipeline:us-east-2:80398EXAMPLE:MyFirstPipeline`

8. EventBridge ルールに関連付けられたターゲットを呼び出すためのアクセス許可を EventBridge に与える IAM サービスロールを作成または指定します (この場合、ターゲットは CodePipeline)。

   - この特定のリソースの新しいロールを作成する を選択して、パイプラインの実行を開始するアクセス許可を EventBridge に付与するサービスロールを作成します。
   - 既存のロールを使用して、パイプラインの実行を開始するアクセス許可を EventBridge に付与するサービスロールを入力します。

9. (オプション) 特定のイメージ ID でソースオーバーライドを指定するには、入力トランスフォーマーを使用してデータを JSON パラメータとして渡します。入力トランスフォーマーを使用してパイプライン変数を渡すこともできます。

   - [追加の設定] を展開します。

     [ターゲット入力を設定] で、[入力トランスフォーマーを設定] を選択します。

     ダイアログウィンドウで、[独自のサンプルイベントを入力] を選択します。[入力パス] ボックスで、次のキーと値のペアを入力します。

     ```
     {"revisionValue": "$.detail.image-digest"}
     ```

   - [テンプレート] ボックスで、次のキーと値のペアを入力します。

     ```
     {
         "sourceRevisions": [
             {
                 "actionName": "Source",
                 "revisionType": "IMAGE_DIGEST",
                 "revisionValue": "<revisionValue>"
             }
         ],
          "variables": [
             {
                 "name": "Variable_Name",
                 "value": "Variable_Value"
             }
         ]
     }
     ```

   - [確認] を選択します。

10. ルール設定を確認して、要件を満たしていることを確認します。
11. [詳細の設定] を選択します。
12. [Configure rule details] ページでルールの名前と説明を入力してから、[State] を選択してルールを有効化します。
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。

Amazon ECR ソースに対する EventBridge ルールを作成する (CLI)

put-rule コマンドを呼び出して、以下を指定します。

- 作成中のルールを一意に識別する名前。この名前は、 AWS アカウントに関連付けられた CodePipeline で作成するすべてのパイプラインで一意である必要があります。
- ルールで使用するソースと詳細フィールドのイベントパターン。詳細については、「Amazon EventBridge とイベントパターン」を参照してください。

Amazon ECR をイベントソース、CodePipeline をターゲットとして EventBridge ルールを作成するには

1. EventBridge が CodePipeline を使用してルールを呼び出すためのアクセス許可を追加します。詳細については、 デベロッパーガイドの [Amazon EventBridge のリソースベースのポリシーを使用する] を参照してください。

   a. 次のサンプルを使用して、EventBridge にサービスロールの引き受けを許可する信頼ポリシーを作成します。信頼ポリシーに `trustpolicyforEB.json` と名前を付けます。

      JSON

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

   b. 次のコマンドを使用して、`Role-for-MyRule` ロールを作成し、信頼ポリシーをアタッチします。

      ```
      aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document
       file://trustpolicyforEB.json
      ```

c. 次のサンプルに示すように、MyFirstPipeline というパイプラインに対して、アクセス権限ポリシー JSON を作成します。アクセス権限ポリシーに permissionspolicyforEB.json と名前を付けます。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:StartPipelineExecution"
            ],
            "Resource": [
                "arn:aws:codepipeline:us-
west-2:111122223333:MyFirstPipeline"
            ]
        }
    ]
}
```

d. 次のコマンドを使用して、Role-for-MyRule ロールに CodePipeline-Permissions-Policy-for-EB アクセス権限ポリシーをアタッチします。

この変更を行う理由 ロールにこのポリシーを追加すると、EventBridge に対するアクセス許可が作成されます。

```
aws iam put-role-policy --role-name Role-for-MyRule --policy-name CodePipeline-
Permissions-Policy-For-EB --policy-document file://permissionspolicyforEB.json
```

2. put-rule コマンドを呼び出し、--name、--event-pattern、--role-arn パラメータを含めます。

この変更を行う理由 イメージプッシュを行う方法を指定するルールと、そのイベントによって開始されるパイプラインを指定するターゲットを持つイベントを作成する必要があります。

次のサンプルコマンドは、MyECRRepoRule というルールを作成します。

```
aws events put-rule --name "MyECRRepoRule" --event-pattern "{\"detail-type\":[\"ECR
 Image Action\"],\"source\":[\"aws.ecr\"],\"detail\":{\"action-type\":[\"PUSH\"],
```

```
\"image-tag\":[\"latest\"],\"repository-name\":[\"eb-test\"],\"result\":[\"SUCCESS
\"]}}}" --role-arn "arn:aws:iam::ACCOUNT_ID:role/Role-for-MyRule"
```

> **Note**
>
> Amazon ECR イベントでサポートされているイベントパターン全体を表示するには、[Amazon ECR Events と EventBridge] または [Amazon Elastic Container Registry Events] を参照してください。

3. CodePipeline をターゲットとして追加するには、put-targets コマンドを呼び出し、次のパラメータを含めます。

   - `--rule` パラメータは、put-rule を使用して作成した `rule_name` で使用されます。
   - `--targets` パラメータは、ターゲットリストのリスト `Id` とターゲットパイプラインの `ARN` で使用されます。

   次のサンプルコマンドでは、`MyECRRepoRule` と呼ばれるルールに対して指定し、ターゲット `Id` は 1 番で構成されています。これは、ルールのターゲットのリストが何であるかを示し、この場合は ターゲット 1 です。サンプルコマンドでは、パイプラインの例 `Arn` とルールの例 `RoleArn` も指定します。パイプラインは、リポジトリ内に変更が加えられると開始します。

```
aws events put-targets --rule MyECRRepoRule --targets
 Id=1,Arn=arn:aws:codepipeline:us-
west-2:80398EXAMPLE:TestPipeline,RoleArn=arn:aws:iam::80398EXAMPLE:role/Role-for-
MyRule
```

4. (オプション) 特定のイメージ ID のソースオーバーライドを使用して入力トランスフォーマーを設定するには、CLI コマンドで次の JSON を使用します。次の例では、以下のオーバーライドを設定します。

   - `actionName` は (この例では `Source`)、ソースイベントから派生したものではなく、パイプラインの作成時に定義された動的な値です。
   - `revisionType` は (この例では `IMAGE_DIGEST`)、ソースイベントから派生したものではなく、パイプラインの作成時に定義された動的な値です。
   - `revisionValue` は (この例では <*revisionValue*>)、ソースイベント変数から派生しています。

```
{
    "Rule": "my-rule",
    "Targets": [
        {
            "Id": "MyTargetId",
            "Arn": "ARN",
            "InputTransformer": {
                "InputPathsMap": {
                    "revisionValue": "$.detail.image-digest"
                },
                "InputTemplate": {
                    "sourceRevisions": [
                        {
                            "actionName": "Source",
                            "revisionType": "IMAGE_DIGEST",
                            "revisionValue": "<revisionValue>"
                        }
                    ]
                }
            }
        }
    ]
}
```

Amazon ECR ソースの EventBridge ルールを作成する (CloudFormation テンプレート)

CloudFormation を使用してルールを作成するには、次に示すようにテンプレートスニペットを使用します。

パイプライン CloudFormation テンプレートを更新して EventBridge ルールを作成するには

1. テンプレートの でResources、 AWS::IAM::Role CloudFormation リソースを使用して、イベントがパイプラインを開始できるようにする IAM ロールを設定します。このエントリによって、2 つのポリシーを使用するロールが作成されます。

   - 最初のポリシーでは、ロールを引き受けることを許可します。
   - 2 つめのポリシーでは、パイプラインを開始するアクセス権限が付与されます。

この変更を行う理由 パイプラインで実行を開始するには、EventBridge によって引き受けることができるロールを作成する必要があります。

YAML

```
  EventRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Effect: Allow
            Principal:
              Service:
                - events.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Sub arn:aws:codepipeline:${AWS::Region}:
${AWS::AccountId}:${AppPipeline}
```

JSON

```
{
    "EventRole": {
        "Type": "AWS::IAM::Role",
        "Properties": {
            "AssumeRolePolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
```

```
                        "Principal": {
                            "Service": [
                                "events.amazonaws.com"
                            ]
                        },
                        "Action": "sts:AssumeRole"
                    }
                ]
            },
            "Path": "/",
            "Policies": [
                {
                    "PolicyName": "eb-pipeline-execution",
                    "PolicyDocument": {
                        "Version": "2012-10-17",
                        "Statement": [
                            {
                                "Effect": "Allow",
                                "Action": "codepipeline:StartPipelineExecution",
                                "Resource": {
                                    "Fn::Sub": "arn:aws:codepipeline:
${AWS::Region}:${AWS::AccountId}:${AppPipeline}"
                                }
                            }
                        ]
                    }
                }
            ]
        }
    }
}
...
```

2. テンプレートの でResources、 AWS::Events::Rule CloudFormation リソースを使用して Amazon ECR ソースの EventBridge ルールを追加します。このイベントパターンは、リポジトリへのコミットを監視するイベントを作成します。EventBridge でリポジトリの状態の変更が検出されると、ルールはターゲットパイプラインで StartPipelineExecution を呼び出します。

   この変更を行う理由 イメージプッシュを行う方法を指定するルールと、そのイベントによって開始されるパイプラインを指定するターゲットを持つイベントを作成する必要があります。

   このスニペットでは、latest のタグが付いた eb-test というイメージを使用しています。

YAML

```
EventRule:
  Type: 'AWS::Events::Rule'
  Properties:
    EventPattern:
      detail:
        action-type: [PUSH]
        image-tag: [latest]
        repository-name: [eb-test]
        result: [SUCCESS]
      detail-type: [ECR Image Action]
      source: [aws.ecr]
    Targets:
      - Arn: !Sub arn:aws:codepipeline:${AWS::Region}:${AWS::AccountId}:
${AppPipeline}
        RoleArn: !GetAtt
          - EventRole
          - Arn
        Id: codepipeline-AppPipeline
```

JSON

```
{
    "EventRule": {
        "Type": "AWS::Events::Rule",
        "Properties": {
            "EventPattern": {
                "detail": {
                    "action-type": [
                        "PUSH"
                    ],
                    "image-tag": [
                        "latest"
                    ],
                    "repository-name": [
                        "eb-test"
                    ],
                    "result": [
                        "SUCCESS"
                    ]
```

```
                },
                "detail-type": [
                    "ECR Image Action"
                ],
                "source": [
                    "aws.ecr"
                ]
            },
            "Targets": [
                {
                    "Arn": {
                        "Fn::Sub": "arn:aws:codepipeline:${AWS::Region}:
${AWS::AccountId}:${AppPipeline}"
                    },
                    "RoleArn": {
                        "Fn::GetAtt": [
                            "EventRole",
                            "Arn"
                        ]
                    },
                    "Id": "codepipeline-AppPipeline"
                }
            ]
        }
    }
},
```

> **Note**
>
> Amazon ECR イベントでサポートされているイベントパターン全体を表示するには、[Amazon ECR Events と EventBridge] または [Amazon Elastic Container Registry Events] を参照してください。

3. (オプション) 特定のイメージ ID のソースオーバーライドを使用して入力トランスフォーマーを設定するには、次の YAML スニペットを使用します。次の例では、以下のオーバーライドを設定します。

   - `actionName` は (この例では `Source`)、ソースイベントから派生したものではなく、パイプラインの作成時に定義された動的な値です。

- `revisionType` は (この例では `IMAGE_DIGEST`)、ソースイベントから派生したものではなく、パイプラインの作成時に定義された動的な値です。
- `revisionValue` は (この例では <*revisionValue*>)、ソースイベント変数から派生しています。

```
---
Rule: my-rule
Targets:
- Id: MyTargetId
  Arn: ARN
  InputTransformer:
    InputPathsMap:
      revisionValue: "$.detail.image-digest"
    InputTemplate:
      sourceRevisions:
        actionName: Source
        revisionType: IMAGE_DIGEST
        revisionValue: '<revisionValue>'
```

4. 更新したテンプレートをローカルコンピュータに保存し、 CloudFormation コンソールを開きます。
5. スタックを選択し、[既存スタックの変更セットの作成] を選択します。
6. テンプレートをアップロードし、 CloudFormationに示された変更を表示します。これらがスタックに加えられる変更です。新しいリソースがリストに表示されています。
7. [実行] を選択してください。

イベントに対して有効にしたソースを使用して Amazon S3 ソースアクションに接続する

このセクションの手順では、 AWS CloudTrail リソースを作成または管理する必要がない S3 ソースアクションを作成する手順について説明します。

> **Important**
>
> AWS CloudTrail リソースなしでこのアクションを作成する手順は、 コンソールでは使用できません。CLI を使用するには、このページの手順または「イベントに対応した S3 ソースを使用してポーリングパイプラインを移行する」を参照してください。

Amazon S3 ソースを含むパイプラインでは、変更検出が EventBridge を通じて自動化され、イベント通知が有効になっているソースバケットを使用するように、パイプラインを修正します。これは、CLI または を使用してパイプライン CloudFormation を移行する場合に推奨される方法です。

> **Note**
>
> この方法では、イベント通知が有効になっているバケットを使用するため、別途 CloudTrail 証跡を作成する必要はありません。コンソールを使用する場合は、イベントルールと CloudTrail 証跡が自動的に設定されます。これらの手順については、「S3 ソースと CloudTrail 証跡を使用してポーリングパイプラインを移行する」を参照してください。

- CLI: S3 ソースと CloudTrail 証跡を使用してポーリングパイプラインを移行する (CLI)
- CloudFormation: S3 ソースと CloudTrail 証跡 (CloudFormation テンプレート) を使用してポーリングパイプラインを移行する

イベントに対して有効な S3 ソースを使用してパイプラインを作成する (CLI)

EventBridge でイベントを使用して変更を検出する S3 ソースを使用してパイプラインを作成するには、次の手順に従います。CLI を使用してパイプラインを作成する詳細な手順については、「パイプライン、ステージ、アクションを作成する」を参照してください。

Amazon S3 でイベント駆動型パイプラインを構築するには、パイプラインの `PollForSourceChanges` パラメータを編集してから、以下のリソースを作成します。

- EventBridge イベントバス
- EventBridge イベントによるパイプラインの開始を許可する IAM ロール

Amazon S3 をイベントソース、CodePipeline をターゲットとする EventBridge ルールを作成し、アクセス許可ポリシーを適用するには

1. EventBridge が CodePipeline を使用してルールを呼び出すためのアクセス許可を付与します。詳細については、 デベロッパーガイドの [Amazon EventBridge のリソースベースのポリシーを使用する] を参照してください。

   a. 次のサンプルを使用して、EventBridge にサービスロールの引き受けを許可する信頼ポリシーを作成します。このスクリプトに `trustpolicyforEB.json` という名前を付けます。

      JSON

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

   b. 次のコマンドを使用して、`Role-for-MyRule` ロールを作成し、信頼ポリシーをアタッチします。

      この変更を行う理由 ロールにこの信頼ポリシーを追加すると、EventBridge に対するアクセス許可が作成されます。

      ```
      aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document
       file://trustpolicyforEB.json
      ```

   c. 次に示すように、`MyFirstPipeline` という名前のパイプラインに対してアクセス許可ポリシー JSON を作成します。アクセス権限ポリシーに `permissionspolicyforEB.json` と名前を付けます。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:StartPipelineExecution"
            ],
            "Resource": [
                "arn:aws:codepipeline:us-
west-2:111122223333:MyFirstPipeline"
            ]
        }
    ]
}
```

d. 次のコマンドを実行して、作成した Role-for-MyRule ロールに新しい CodePipeline-Permissions-Policy-for-EB アクセス権限ポリシーをアタッチします。

```
aws iam put-role-policy --role-name Role-for-MyRule --policy-name CodePipeline-
Permissions-Policy-For-EB --policy-document file://permissionspolicyforEB.json
```

2. put-rule コマンドを呼び出し、--name、--event-pattern、--role-arn パラメータを含めます。

   次のサンプルコマンドでは、EnabledS3SourceRule という名前のルールが作成されます。

```
aws events put-rule --name "EnabledS3SourceRule" --event-pattern "{\"source\":
[\"aws.s3\"],\"detail-type\":[\"Object Created\"],\"detail\":{\"bucket\":{\"name\":
[\"amzn-s3-demo-source-bucket\"]}}}" --role-arn "arn:aws:iam::ACCOUNT_ID:role/Role-
for-MyRule"
```

3. CodePipeline をターゲットとして追加するには、put-targets コマンドを呼び出し、--rule および --targets パラメータを含めます。

   次のコマンドでは、EnabledS3SourceRule という名前のルールに対して指定し、ターゲット Id は 1 番で構成されています。これは、ルールのターゲットのリストが何であるかを示し、こ

の場合は ターゲット 1 です。このコマンドでは、パイプラインのサンプルの ARN も指定されます。パイプラインは、リポジトリ内に変更が加えられると開始します。

```
aws events put-targets --rule EnabledS3SourceRule --targets Id=codepipeline-
AppPipeline,Arn=arn:aws:codepipeline:us-west-2:80398EXAMPLE:TestPipeline
```

パイプラインの PollForSourceChanges パラメータを編集するには

> **Important**
>
> このメソッドを使用してパイプラインを作成すると、PollForSourceChanges パラメータはデフォルトで true になります (ただし、明示的に false に設定した場合は除きます)。イベントベースの変更検出を追加する場合は、このパラメータを出力に追加する必要があります。ポーリングを無効にするには、このパラメータを false に設定します。そうしないと、1 つのソース変更に対してパイプラインが 2 回起動されます。詳細については、「PollForSourceChanges パラメータの有効な設定」を参照してください

1. get-pipeline コマンドを実行して、パイプライン構造を JSON ファイルにコピーします。例えば、MyFirstPipeline という名前のパイプラインに対して、以下のコマンドを実行します。

   ```
   aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
   ```

   このコマンドは何も返しませんが、作成したファイルは、コマンドを実行したディレクトリにあります。

2. この例に示すように、プレーンテキストエディタで JSON ファイルを開き、PollForSourceChanges という名前のバケットの amzn-s3-demo-source-bucket パラメータを false に変更してソースステージを編集します。

   この変更を行う理由 このパラメータを false に設定すると、定期的チェックがオフになるため、イベントベースの変更検出のみ使用することができます。

   ```
   "configuration": {
       "S3Bucket": "amzn-s3-demo-source-bucket",
       "PollForSourceChanges": "false",
       "S3ObjectKey": "index.zip"
   },
   ```

3. get-pipeline コマンドを使用して取得したパイプライン構造を使用している場合、JSON ファイルから metadata 行を削除する必要があります。それ以外の場合は、update-pipeline コマンドで使用することはできません。"metadata": { } 行と、"created"、"pipelineARN"、"updated" フィールドを削除します。

   例えば、構造から以下の行を削除します。

```
"metadata": {
    "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
    "created": "date",
    "updated": "date"
},
```

   ファイルを保存します。

4. 変更を適用するには、パイプライン JSON ファイルを指定して、update-pipeline コマンドを実行します。

> **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 codepipeline update-pipeline --cli-input-json file://pipeline.json
```

   このコマンドは、編集したパイプラインの構造全体を返します。

> **Note**
>
> update-pipeline コマンドは、パイプラインを停止します。update-pipeline コマンドを実行したときにパイプラインによりリビジョンが実行されている場合、その実行は停止します。更新されたパイプラインによりそのリビジョンを実行するには、パイプラインを手動で開始する必要があります。パイプラインを手動で開始するには start-pipeline-execution コマンドを使用します。

イベントに対して S3 ソースを有効にしてパイプラインを作成する (CloudFormation テンプレート)

この手順は、ソースバケットでイベントが有効になっているパイプライン用です。

イベントベースで変更を検出する Amazon S3 ソースを使用してパイプラインを作成するには、次の手順に従います。

Amazon S3 でイベント駆動型パイプラインを構築するには、パイプラインの `PollForSourceChanges` パラメータを編集してから、以下のリソースをテンプレートに追加します。

- このイベントによるパイプラインの開始を許可する EventBridge ルールと IAM ロール。

CloudFormation を使用してパイプラインを作成および管理する場合、テンプレートには次のようなコンテンツが含まれます。

> **Note**
>
> `PollForSourceChanges` と呼ばれるソースステージの `Configuration` プロパティ。テンプレートにプロパティが含まれていない場合、`PollForSourceChanges` はデフォルトで `true` に設定されます。

YAML

```
AppPipeline:
  Type: AWS::CodePipeline::Pipeline
  Properties:
    RoleArn: !GetAtt CodePipelineServiceRole.Arn
    Stages:
      -
        Name: Source
        Actions:
          -
            Name: SourceAction
            ActionTypeId:
              Category: Source
              Owner: AWS
              Version: 1
```

```
                Provider: S3
              OutputArtifacts:
                -
                  Name: SourceOutput
              Configuration:
                S3Bucket: !Ref SourceBucket
                S3ObjectKey: !Ref S3SourceObjectKey
                PollForSourceChanges: true
              RunOrder: 1


...
```

JSON

```
        "AppPipeline": {
            "Type": "AWS::CodePipeline::Pipeline",
            "Properties": {
                "RoleArn": {
                    "Fn::GetAtt": ["CodePipelineServiceRole", "Arn"]
                },
                "Stages": [
                    {
                        "Name": "Source",
                        "Actions": [
                            {
                                "Name": "SourceAction",
                                "ActionTypeId": {
                                    "Category": "Source",
                                    "Owner": "AWS",
                                    "Version": 1,
                                    "Provider": "S3"
                                },
                                "OutputArtifacts": [
                                    {
                                        "Name": "SourceOutput"
                                    }
                                ],
                                "Configuration": {
                                    "S3Bucket": {
                                        "Ref": "SourceBucket"
                                    },
                                    "S3ObjectKey": {
```

```
                                    "Ref": "SourceObjectKey"
                                },
                                "PollForSourceChanges": true
                            },
                            "RunOrder": 1
                        }
                    ]
                },


...
```

Amazon S3 をイベントソース、CodePipeline をターゲットとする EventBridge ルールを作成し、アクセス許可ポリシーを適用するには

1. テンプレートの でResources、 AWS::IAM::Role CloudFormation リソースを使用して、イベントがパイプラインを開始できるようにする IAM ロールを設定します。このエントリによって、2 つのポリシーを使用するロールが作成されます。

   - 最初のポリシーでは、ロールを引き受けることを許可します。
   - 2 つめのポリシーでは、パイプラインを開始するアクセス権限が付与されます。

   この変更を行う理由 AWS::IAM::Role リソースを追加する CloudFormation と、 は EventBridge のアクセス許可を作成できます。このリソースは CloudFormation スタックに追加されます。

   YAML

   ```
   EventRole:
     Type: AWS::IAM::Role
     Properties:
       AssumeRolePolicyDocument:
         Version: 2012-10-17
         Statement:
           -
             Effect: Allow
             Principal:
               Service:
                 - events.amazonaws.com
             Action: sts:AssumeRole
   ```

```
      Path: /
      Policies:
        -
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref
 'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]


...
```

JSON

```
  "EventRole": {
    "Type": "AWS::IAM::Role",
    "Properties": {
      "AssumeRolePolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Principal": {
              "Service": [
                "events.amazonaws.com"
              ]
            },
            "Action": "sts:AssumeRole"
          }
        ]
      },
      "Path": "/",
      "Policies": [
        {
          "PolicyName": "eb-pipeline-execution",
          "PolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
              {
```

```
                "Effect": "Allow",
                "Action": "codepipeline:StartPipelineExecution",
                "Resource": {
                  "Fn::Join": [
                    "",
                    [
                      "arn:aws:codepipeline:",
                      {
                        "Ref": "AWS::Region"
                      },
                      ":",
                      {
                        "Ref": "AWS::AccountId"
                      },
                      ":",
                      {
                        "Ref": "AppPipeline"
                      }
                    ]
                  ]

...
```

2. AWS::Events::Rule CloudFormation リソースを使用して EventBridge ルールを追加します。このイベントパターンは、Amazon S3 ソースバケット内のオブジェクトの作成または削除をモニタリングするイベントを作成します。さらに、パイプラインのターゲットも含めます。オブジェクトが作成されると、このルールによりターゲットパイプラインで StartPipelineExecution が呼び出されます。

   この変更を行う理由 AWS::Events::Rule リソースを追加すると、 CloudFormation は イベントを作成できます。このリソースは CloudFormation スタックに追加されます。

   YAML

```
  EventRule:
    Type: AWS::Events::Rule
    Properties:
      EventBusName: default
      EventPattern:
        source:
          - aws.s3
        detail-type:
          - Object Created
```

```
        detail:
          bucket:
            name:
              - !Ref SourceBucket
      Name: EnabledS3SourceRule
      State: ENABLED
      Targets:
        -
          Arn:
            !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref AppPipeline ] ]
          RoleArn: !GetAtt EventRole.Arn
          Id: codepipeline-AppPipeline


...
```

JSON

```
 "EventRule": {
   "Type": "AWS::Events::Rule",
   "Properties": {
 "EventBusName": "default",
 "EventPattern": {
     "source": [
  "aws.s3"
     ],
     "detail-type": [
   "Object Created"
     ],
     "detail": {
   "bucket": {
       "name": [
     "s3-pipeline-source-fra-bucket"
       ]
       }
           }
 },
 "Name": "EnabledS3SourceRule",
       "State": "ENABLED",
       "Targets": [
       {
```

```
          "Arn": {
            "Fn::Join": [
              "",
              [
                "arn:aws:codepipeline:",
                {
                  "Ref": "AWS::Region"
                },
                ":",
                {
                  "Ref": "AWS::AccountId"
                },
                ":",
                {
                  "Ref": "AppPipeline"
                }
              ]
            ]
          },
          "RoleArn": {
            "Fn::GetAtt": [
              "EventRole",
              "Arn"
            ]
          },
          "Id": "codepipeline-AppPipeline"
        }
      ]
    }
  }
},

...
```

3. 更新したテンプレートをローカルコンピュータに保存し、 CloudFormation コンソールを開きます。
4. スタックを選択し、[既存スタックの変更セットの作成] を選択します。
5. 更新されたテンプレートをアップロードし、 CloudFormationに示された変更を表示します。これらがスタックに加えられる変更です。新しいリソースがリストに表示されています。
6. [実行] を選択してください。

パイプラインの PollForSourceChanges パラメータを編集するには

> **Important**
>
> このメソッドを使用してパイプラインを作成すると、PollForSourceChanges パラメータはデフォルトで true になります (ただし、明示的に false に設定した場合は除きます)。イベントベースの変更検出を追加する場合は、このパラメータを出力に追加する必要があります。ポーリングを無効にするには、このパラメータを false に設定します。そうしないと、1 つのソース変更に対してパイプラインが 2 回起動されます。詳細については、「PollForSourceChanges パラメータの有効な設定」を参照してください

- テンプレートで、PollForSourceChanges を false に変更します。パイプライン定義に PollForSourceChanges が含まれていなかった場合は、追加して false に設定します。

  この変更を行う理由 PollForSourceChanges パラメータを false に変更すると、定期的チェックがオフになるため、イベントベースの変更検出のみ使用することができます。

  YAML

  ```
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: AWS
                Version: 1
                Provider: S3
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                S3Bucket: !Ref SourceBucket
                S3ObjectKey: !Ref SourceObjectKey
                PollForSourceChanges: false
              RunOrder: 1
  ```

  JSON

  ```
  {
     "Name": "SourceAction",
  ```

```
    "ActionTypeId": {
      "Category": "Source",
      "Owner": "AWS",
      "Version": 1,
      "Provider": "S3"
    },
    "OutputArtifacts": [
      {
        "Name": "SourceOutput"
      }
    ],
    "Configuration": {
      "S3Bucket": {
        "Ref": "SourceBucket"
      },
      "S3ObjectKey": {
        "Ref": "SourceObjectKey"
      },
      "PollForSourceChanges": false
    },
    "RunOrder": 1
  }
```

Example

CloudFormation を使用してこれらのリソースを作成すると、リポジトリ内のファイルが作成または更新されたときにパイプラインがトリガーされます。

> **Note**
>
> ここで手順は終わりではありません。パイプラインは作成されますが、Amazon S3 パイプライン用の 2 番目の CloudFormation テンプレートを作成する必要があります。2 番目のテンプレートを作成しない場合、パイプラインに変更検出機能はありません。

YAML

```
Parameters:
  SourceObjectKey:
    Description: 'S3 source artifact'
```

```
    Type: String
    Default: SampleApp_Linux.zip
  ApplicationName:
    Description: 'CodeDeploy application name'
    Type: String
    Default: DemoApplication
  BetaFleet:
    Description: 'Fleet configured in CodeDeploy'
    Type: String
    Default: DemoFleet

Resources:
  SourceBucket:
    Type: AWS::S3::Bucket
    Properties:
      NotificationConfiguration:
        EventBridgeConfiguration:
          EventBridgeEnabled: true
      VersioningConfiguration:
        Status: Enabled
  CodePipelineArtifactStoreBucket:
    Type: AWS::S3::Bucket
  CodePipelineArtifactStoreBucketPolicy:
    Type: AWS::S3::BucketPolicy
    Properties:
      Bucket: !Ref CodePipelineArtifactStoreBucket
      PolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Sid: DenyUnEncryptedObjectUploads
            Effect: Deny
            Principal: '*'
            Action: s3:PutObject
            Resource: !Join [ '', [ !GetAtt CodePipelineArtifactStoreBucket.Arn, '/
*' ] ]
            Condition:
              StringNotEquals:
                s3:x-amz-server-side-encryption: aws:kms
          -
            Sid: DenyInsecureConnections
            Effect: Deny
            Principal: '*'
            Action: s3:*
```

```
            Resource: !Sub ${CodePipelineArtifactStoreBucket.Arn}/*
            Condition:
              Bool:
                aws:SecureTransport: false
  CodePipelineServiceRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Effect: Allow
            Principal:
              Service:
                - codepipeline.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: AWS-CodePipeline-Service-3
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action:
                  - codecommit:CancelUploadArchive
                  - codecommit:GetBranch
                  - codecommit:GetCommit
                  - codecommit:GetUploadArchiveStatus
                  - codecommit:UploadArchive
                Resource: 'resource_ARN'
              -
                Effect: Allow
                Action:
                  - codedeploy:CreateDeployment
                  - codedeploy:GetApplicationRevision
                  - codedeploy:GetDeployment
                  - codedeploy:GetDeploymentConfig
                  - codedeploy:RegisterApplicationRevision
                Resource: 'resource_ARN'
              -
                Effect: Allow
                Action:
```

```
                - codebuild:BatchGetBuilds
                - codebuild:StartBuild
              Resource: 'resource_ARN'
            -
              Effect: Allow
              Action:
                - devicefarm:ListProjects
                - devicefarm:ListDevicePools
                - devicefarm:GetRun
                - devicefarm:GetUpload
                - devicefarm:CreateUpload
                - devicefarm:ScheduleRun
              Resource: 'resource_ARN'
            -
              Effect: Allow
              Action:
                - lambda:InvokeFunction
                - lambda:ListFunctions
              Resource: 'resource_ARN'
            -
              Effect: Allow
              Action:
                - iam:PassRole
              Resource: 'resource_ARN'
            -
              Effect: Allow
              Action:
                - elasticbeanstalk:*
                - ec2:*
                - elasticloadbalancing:*
                - autoscaling:*
                - cloudwatch:*
                - s3:*
                - sns:*
                - cloudformation:*
                - rds:*
                - sqs:*
                - ecs:*
              Resource: 'resource_ARN'
  AppPipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      Name: s3-events-pipeline
      RoleArn:
```

```
        !GetAtt CodePipelineServiceRole.Arn
      Stages:
        -
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: AWS
                Version: 1
                Provider: S3
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                S3Bucket: !Ref SourceBucket
                S3ObjectKey: !Ref SourceObjectKey
                PollForSourceChanges: false
              RunOrder: 1
        -
          Name: Beta
          Actions:
            -
              Name: BetaAction
              InputArtifacts:
                - Name: SourceOutput
              ActionTypeId:
                Category: Deploy
                Owner: AWS
                Version: 1
                Provider: CodeDeploy
              Configuration:
                ApplicationName: !Ref ApplicationName
                DeploymentGroupName: !Ref BetaFleet
              RunOrder: 1
      ArtifactStore:
        Type: S3
        Location: !Ref CodePipelineArtifactStoreBucket
  EventRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
```

```
          -
            Effect: Allow
            Principal:
              Service:
                - events.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region',
':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]
 EventRule:
   Type: AWS::Events::Rule
   Properties:
     EventBusName: default
     EventPattern:
       source:
         - aws.s3
       detail-type:
         - Object Created
       detail:
         bucket:
           name:
             - !Ref SourceBucket
     Name: EnabledS3SourceRule
     State: ENABLED
     Targets:
       -
         Arn:
           !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
'AWS::AccountId', ':', !Ref AppPipeline ] ]
         RoleArn: !GetAtt EventRule.Arn
         Id: codepipeline-AppPipeline
```

JSON

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "appconfig:StartDeployment",
                "appconfig:StopDeployment",
                "appconfig:GetDeployment"
            ],
            "Resource": [
                "arn:aws:appconfig:*:111122223333:application/
[[Application]]",
                "arn:aws:appconfig:*:111122223333:application/[[Application]]/
*",
                "arn:aws:appconfig:*:111122223333:deploymentstrategy/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

EventBridge と を使用する Amazon S3 ソースアクションへの接続 AWS CloudTrail

このセクションの手順では、作成および管理する必要がある AWS CloudTrail リソースを使用する S3 ソースアクションを作成する手順について説明します。追加の AWS CloudTrail リソースを必要としない EventBridge で S3 ソースアクションを使用するには、「」の CLI の手順を使用しますイベントに対応した S3 ソースを使用してポーリングパイプラインを移行する。

> **Important**
>
> この手順では、作成および管理する必要がある AWS CloudTrail リソースを使用する S3 ソースアクションを作成するステップを示します。 AWS CloudTrail リソースなしでこのアクションを作成する手順は、 コンソールでは使用できません。CLI を使用するには、「イベン

トに対応した S3 ソースを使用してポーリングパイプラインを移行する」を参照してください。

CodePipeline で Amazon S3 のソースアクションを追加するには、次のいずれかを選択できます。

- CodePipeline コンソールの [パイプラインの作成] ウィザード (カスタムパイプラインを作成する (コンソール))、または [アクションを編集する] ページを使用して、[S3] プロバイダオプションを選択します。コンソールは、ソースが変更されたときにパイプラインを開始する EventBridge ルールと CloudTrail 証跡を作成します。
- を使用して、アクションのS3アクション設定 AWS CLI を追加し、次のように追加のリソースを作成します。
  - S3 でのアクション設定の例を Amazon S3 ソースアクションリファレンス で使用し、パイプラインを作成する (CLI) で表示されるようにアクションを作成します。
  - 変更検出方法のデフォルトは、ソースをポーリングすることによってパイプラインを開始します。定期的なチェックを無効にして、手動で変更検出ルールと証跡を作成する必要があります。Amazon S3 ソースに対する EventBridge ルールを作成する (コンソール)、Amazon S3 ソースに対する EventBridge ルールを作成する (CLI)、Amazon S3 ソースの EventBridge ルールを作成する (CloudFormation テンプレート) Amazon S3 の内、いずれかののいずれかの方法を使用します。

AWS CloudTrail は、Amazon S3 ソースバケットのイベントをログに記録してフィルタリングするサービスです。この証跡によって、フィルタ処理されたソースの変更が EventBridge ルールに送信されます。EventBridge ルールはソースの変更を検出し、パイプラインを開始します。

要件:

- 証跡を作成しない場合は、既存の AWS CloudTrail 証跡を使用して Amazon S3 ソースバケットのイベントをログに記録し、フィルタリングされたイベントを EventBridge ルールに送信します。
- がログファイルを保存 AWS CloudTrail できる既存の S3 バケットを作成または使用します。 には、ログファイルを Amazon S3 バケットに配信するために必要なアクセス許可 AWS CloudTrail が必要です。このバケットをリクエスタ支払いバケットとして設定することはできません。コンソールで証跡を作成または更新する一環として Amazon S3 バケットを作成すると、 は必要なアクセス許可をバケットにア AWS CloudTrail タッチします。詳細については、「CloudTrail の Amazon S3 バケットポリシー」を参照して ください。

Amazon S3 ソースに対する EventBridge ルールを作成する (コンソール)

EventBridge でルールを設定する前に、 AWS CloudTrail 証跡を作成する必要があります。詳細については、「コンソールで証跡を作成する」を参照してください。

> **Important**
>
> コンソールを使用してパイプラインを作成または編集する場合は、EventBridge ルールと AWS CloudTrail 証跡が自動的に作成されます。

証跡を作成するには

1. AWS CloudTrail コンソールを開きます。
2. ナビゲーションペインで、[Trails] (追跡) を選択します。
3. [追跡の作成]を選択します。[Trail name] に、証跡の名前を入力します。
4. [保存場所] でログファイルを保存するために使用するバケットを作成あるいは指定します。デフォルトでは、Amazon S3 バケットとオブジェクトはプライベートです。リソース所有者 (バケットを作成した AWS アカウント) のみがバケットとそのオブジェクトにアクセスできます。バケットには、バケット内のオブジェクトへのアクセス AWS CloudTrail 許可を付与するリソースポリシーが必要です。
5. [証跡ログバケットとフォルダ] で、フォルダ内のすべてのオブジェクトのデータイベントを記録するための Amazon S3 バケットとオブジェクトプレフィックス (フォルダ名) を指定します。証跡ごとに、最大 250 個の Amazon S3 オブジェクトを追加できます。必要な暗号化キー情報を入力し、[次へ] を選択します。
6. [イベントタイプ] で [管理イベント] を選択します。
7. [管理イベント] で、[書き込み] を選択します。証跡では、指定したバケットとプレフィックスの Amazon S3 オブジェクトレベルの API アクティビティ (`GetObject` や `PutObject` など) が記録されます。
8. [Write (書き込み)] を選択します。
9. 証跡が適切であることを確認したら、[証跡の作成] を選択します。

Amazon S3 ソースを使用するパイプラインをターゲットとする EventBridge ルールを作成するには

1. Amazon EventBridge コンソール (https://console.aws.amazon.com/events/) を開きます。

2. ナビゲーションペインで [ルール] を選択します。デフォルトのバスを選択したままにするか、イベントバスを選択します。[ルールを作成] を選択します。
3. [名前] で、ルールの名前を入力します。
4. [ルールタイプ] で、[イベントパターンを持つルール] を選択します。[次へ] を選択します。
5. [イベントソース] で、[AWS イベントまたは EventBridge パートナーイベント] を選択します。
6. [サンプルイベントタイプ] で [AWS イベント] を選択します。
7. [サンプルイベント] に、フィルタ処理するキーワードとして「S3」と入力します。[CloudTrail 経由のAWS API コール] を選択します。
8. [作成方法] セクションで、[カスタムパターン (JSON エディタ)] を選択します。

   以下に示すイベントパターンを貼り付けます。バケット名と、バケット内のオブジェクトを `requestParameters` として一意に識別する S3 オブジェクトキー (またはキー名) を必ず追加してください。この例では、`amzn-s3-demo-source-bucket` というバケットと `my-files.zip` というオブジェクトキーのルールが作成されます。リソースを指定するために [編集] ウィンドウを使用する場合、ルールはカスタムイベントパターンを使用するように更新されます。

   以下に示しているのは、コピーして貼り付けるサンプルイベントパターンです。

```
{
    "source": [
        "aws.s3"
    ],
    "detail-type": [
        "AWS API Call via CloudTrail"
    ],
    "detail": {
        "eventSource": [
            "s3.amazonaws.com"
        ],
        "eventName": [
            "CopyObject",
            "CompleteMultipartUpload",
            "PutObject"
        ],
        "requestParameters": {
            "bucketName": [
                "amzn-s3-demo-source-bucket"
            ],
```

```
            "key": [
                "my-files.zip"
            ]
        }
    }
}
```

9. [次へ] を選択します。
10. [ターゲットタイプ] で、[AWS サービス] を選択します。
11. [ターゲットの選択] で、[CodePipeline] を選択します。[パイプライン ARN] に、このルールによって開始されるパイプラインの ARN を入力します。

> **Note**
>
> このパイプライン ARN を取得するには、get-pipeline コマンドを実行します。パイプライン ARN が出力に表示されます。以下の形式で作成されます。
> arn:aws:codepipeline:*region*:*account*:*pipeline-name*
> パイプライン ARN の例:
> arn:aws:codepipeline:us-east-2:80398EXAMPLE:MyFirstPipeline

12. EventBridge ルールに関連付けられたターゲットを呼び出すためのアクセス許可を EventBridge に与える IAM サービスロールを作成または指定するには (この場合、ターゲットは CodePipeline):

    - この特定のリソースの新しいロールを作成する を選択して、パイプラインの実行を開始するアクセス許可を EventBridge に付与するサービスロールを作成します。
    - 既存のロールを使用して、パイプラインの実行を開始するアクセス許可を EventBridge に付与するサービスロールを入力します。

13. (オプション) 特定のイメージ ID でソースオーバーライドを指定するには、入力トランスフォーマーを使用してデータを JSON パラメータとして渡します。入力トランスフォーマーを使用してパイプライン変数を渡すこともできます。

    - [追加の設定] を展開します。

      [ターゲット入力を設定] で、[入力トランスフォーマーを設定] を選択します。

      ダイアログウィンドウで、[独自のサンプルイベントを入力] を選択します。[入力パス] ボックスで、次のキーと値のペアを入力します。

```
{"revisionValue": "$.detail.object.version-id"}
```

- [テンプレート] ボックスで、次のキーと値のペアを入力します。

```
                                {
    "sourceRevisions": [
        {
            "actionName": "Source",
            "revisionType": "S3_OBJECT_VERSION_ID",
            "revisionValue": "<revisionValue>"
        }
    ],
     "variables": [
        {
            "name": "Variable_Name",
            "value": "Variable_Value"
        }
    ]
}
```

- [確認] を選択します。

14. [次へ] を選択します。
15. [タグ] ページで、[次へ] を選択します
16. [確認と作成] ページで、ルールの設定を確認します。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。

Amazon S3 ソースに対する EventBridge ルールを作成する (CLI)

AWS CloudTrail 証跡を作成し、ログ記録を有効にするには

を使用して証跡 AWS CLI を作成するには、 create-trail コマンドを呼び出し、以下を指定します。

- 証跡名。
- AWS CloudTrailにバケットポリシーをすでに適用しているバケットです。

詳細については、AWS「コマンドラインインターフェイスを使用した証跡の作成」を参照してください。

1. create-trail コマンドを呼び出し、--name および --s3-bucket-name パラメータを含めます。

   この変更を行う理由 これにより、S3 ソースバケットに必要な CloudTrail 証跡が作成されます。

   次のコマンドでは、--name および --s3-bucket-name を使用して、my-trail という名前の証跡と、amzn-s3-demo-source-bucket という名前のバケットを作成します。

   ```
   aws cloudtrail create-trail --name my-trail --s3-bucket-name amzn-s3-demo-source-bucket
   ```

2. start-logging コマンドを呼び出し、--name パラメータを含めます。

   この変更を行う理由 これにより、ソースバケットの CloudTrail ロギングが開始され、EventBridge にイベントが送信されます。

   例:

   次のコマンドでは、--name を使用して、my-trail という名前の証跡のログ記録を開始します。

   ```
   aws cloudtrail start-logging --name my-trail
   ```

3. put-event-selectors コマンドを呼び出し、--trail-name および --event-selectors パラメータを含めます。イベントセレクタを使用してソースバケットに記録するデータイベントを指定し、それらのイベントを EventBridge ルールに送信します。

   この変更を行う理由 このコマンドはイベントをフィルタ処理します。

   例:

   次のサンプルコマンドでは、--trail-name および --event-selectors を使用してソースバケットと amzn-s3-demo-source-bucket/myFolder という名前のプレフィックスにデータイベントの管理を指定します。

   ```
   aws cloudtrail put-event-selectors --trail-name my-trail --event-selectors
    '[{ "ReadWriteType": "WriteOnly", "IncludeManagementEvents":false,
   ```

```
 "DataResources": [{ "Type": "AWS::S3::Object", "Values": ["arn:aws:s3:::amzn-s3-
demo-source-bucket/myFolder/file.zip"] }] }]'
```

Amazon S3 をイベントソース、CodePipeline をターゲットとする EventBridge ルールを作成し、アクセス許可ポリシーを適用するには

1. EventBridge が CodePipeline を使用してルールを呼び出すためのアクセス許可を付与します。詳細については、 デベロッパーガイドの [Amazon EventBridge のリソースベースのポリシーを使用する] を参照してください。

   a. 次のサンプルを使用して、EventBridge にサービスロールの引き受けを許可する信頼ポリシーを作成します。このスクリプトに `trustpolicyforEB.json` という名前を付けます。

      JSON

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

   b. 次のコマンドを使用して、`Role-for-MyRule` ロールを作成し、信頼ポリシーをアタッチします。

      この変更を行う理由 ロールにこの信頼ポリシーを追加すると、EventBridge に対するアクセス許可が作成されます。

      ```
      aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document
       file://trustpolicyforEB.json
      ```

c. 次に示すように、MyFirstPipeline という名前のパイプラインに対してアクセス許可ポリシー JSON を作成します。アクセス権限ポリシーに permissionspolicyforEB.json と名前を付けます。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:StartPipelineExecution"
            ],
            "Resource": [
                "arn:aws:codepipeline:us-
west-2:111122223333:MyFirstPipeline"
            ]
        }
    ]
}
```

d. 次のコマンドを実行して、作成した Role-for-MyRule ロールに新しい CodePipeline-Permissions-Policy-for-EB アクセス権限ポリシーをアタッチします。

```
aws iam put-role-policy --role-name Role-for-MyRule --policy-name CodePipeline-
Permissions-Policy-For-EB --policy-document file://permissionspolicyforEB.json
```

2. put-rule コマンドを呼び出し、--name、--event-pattern、--role-arn パラメータを含めます。

次のサンプルコマンドでは、MyS3SourceRule という名前のルールが作成されます。

```
aws events put-rule --name "MyS3SourceRule" --event-pattern "{\"source\":
[\"aws.s3\"],\"detail-type\":[\"AWS API Call via CloudTrail\"],\"detail\":
{\"eventSource\":[\"s3.amazonaws.com\"],\"eventName\":[\"CopyObject\",\"PutObject
\",\"CompleteMultipartUpload\"],\"requestParameters\":{\"bucketName\":[\"amzn-s3-
demo-source-bucket\"],\"key\":[\"my-key\"]}}}
 --role-arn "arn:aws:iam::ACCOUNT_ID:role/Role-for-MyRule"
```

3. CodePipeline をターゲットとして追加するには、put-targets コマンドを呼び出し、`--rule` および `--targets` パラメータを含めます。

   次のコマンドでは、MyS3SourceRule という名前のルールに対して指定し、ターゲット Id は 1 番で構成されています。これは、ルールのターゲットのリストが何であるかを示し、この場合は ターゲット 1 です。このコマンドでは、パイプラインのサンプルの ARN も指定されます。パイプラインは、リポジトリ内に変更が加えられると開始します。

   ```
   aws events put-targets --rule MyS3SourceRule --targets
    Id=1,Arn=arn:aws:codepipeline:us-west-2:80398EXAMPLE:TestPipeline
   ```

4. (オプション) 特定のイメージ ID のソースオーバーライドを使用して入力トランスフォーマーを設定するには、CLI コマンドで次の JSON を使用します。次の例では、以下のオーバーライドを設定します。

   - `actionName` は (この例では `Source`)、ソースイベントから派生したものではなく、パイプラインの作成時に定義された動的な値です。
   - `revisionType` は (この例では `S3_OBJECT_VERSION_ID`)、ソースイベントから派生したものではなく、パイプラインの作成時に定義された動的な値です。
   - `revisionValue` は (この例では <*revisionValue*>)、ソースイベント変数から派生しています。

   ```
   {
       "Rule": "my-rule",
       "Targets": [
           {
               "Id": "MyTargetId",
               "Arn": "ARN",
               "InputTransformer": {
                   "InputPathsMap": {
                       "revisionValue": "$.detail.object.version-id"
                   },
                   "InputTemplate": {
                       "sourceRevisions": {
                           "actionName": "Source",
                           "revisionType": "S3_OBJECT_VERSION_ID",
                           "revisionValue": "<revisionValue>"
                       }
                   }
               }
   ```

```
        }
    ]
}
```

パイプラインの PollForSourceChanges パラメータを編集するには

> **Important**
>
> このメソッドを使用してパイプラインを作成すると、PollForSourceChanges パラメータはデフォルトで true になります (ただし、明示的に false に設定した場合は除きます)。イベントベースの変更検出を追加する場合は、このパラメータを出力に追加する必要があります。ポーリングを無効にするには、このパラメータを false に設定します。そうしないと、1 つのソース変更に対してパイプラインが 2 回起動されます。詳細については、「PollForSourceChanges パラメータの有効な設定」を参照してください

1. get-pipeline コマンドを実行して、パイプライン構造を JSON ファイルにコピーします。例えば、MyFirstPipeline という名前のパイプラインに対して、以下のコマンドを実行します。

   ```
   aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
   ```

   このコマンドは何も返しませんが、作成したファイルは、コマンドを実行したディレクトリにあります。

2. この例に示すように、プレーンテキストエディタで JSON ファイルを開き、PollForSourceChanges という名前のバケットの amzn-s3-demo-source-bucket パラメータを false に変更してソースステージを編集します。

   この変更を行う理由 このパラメータを false に設定すると、定期的チェックがオフになるため、イベントベースの変更検出のみ使用することができます。

   ```
   "configuration": {
       "S3Bucket": "amzn-s3-demo-source-bucket",
       "PollForSourceChanges": "false",
       "S3ObjectKey": "index.zip"
   },
   ```

3. get-pipeline コマンドを使用して取得したパイプライン構造を使用している場合、JSON ファイルから metadata 行を削除する必要があります。それ以外の場合

は、update-pipeline コマンドで使用することはできません。"metadata": { } 行と、"created"、"pipelineARN"、"updated" フィールドを削除します。

例えば、構造から以下の行を削除します。

```
"metadata": {
    "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
    "created": "date",
    "updated": "date"
},
```

ファイルを保存します。

4. 変更を適用するには、パイプライン JSON ファイルを指定して、update-pipeline コマンドを実行します。

> **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 codepipeline update-pipeline --cli-input-json file://pipeline.json
```

このコマンドは、編集したパイプラインの構造全体を返します。

> **Note**
>
> update-pipeline コマンドは、パイプラインを停止します。update-pipeline コマンドを実行したときにパイプラインによりリビジョンが実行されている場合、その実行は停止します。更新されたパイプラインによりそのリビジョンを実行するには、パイプラインを手動で開始する必要があります。パイプラインを手動で開始するには start-pipeline-execution コマンドを使用します。

Amazon S3 ソースの EventBridge ルールを作成する (CloudFormation テンプレート) Amazon S3

CloudFormation を使用してルールを作成するには、次に示すようにテンプレートを更新します。

Amazon S3 をイベントソース、CodePipeline をターゲットとする EventBridge ルールを作成し、アクセス許可ポリシーを適用するには

1. テンプレートの でResources、 AWS::IAM::Role CloudFormation リソースを使用して、イベントがパイプラインを開始できるようにする IAM ロールを設定します。このエントリによって、2 つのポリシーを使用するロールが作成されます。

   - 最初のポリシーでは、ロールを引き受けることを許可します。
   - 2 つめのポリシーでは、パイプラインを開始するアクセス権限が付与されます。

   この変更を行う理由 AWS::IAM::Role リソースを追加する CloudFormation と、 は EventBridge のアクセス許可を作成できます。このリソースは CloudFormation スタックに追加されます。

   YAML

```
EventRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: 2012-10-17
      Statement:
        -
          Effect: Allow
          Principal:
            Service:
              - events.amazonaws.com
          Action: sts:AssumeRole
    Path: /
    Policies:
      -
        PolicyName: eb-pipeline-execution
        PolicyDocument:
          Version: 2012-10-17
          Statement:
            -
              Effect: Allow
              Action: codepipeline:StartPipelineExecution
              Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref
'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]
```

```
...
```

JSON

```
"EventRole": {
  "Type": "AWS::IAM::Role",
  "Properties": {
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": [
              "events.amazonaws.com"
            ]
          },
          "Action": "sts:AssumeRole"
        }
      ]
    },
    "Path": "/",
    "Policies": [
      {
        "PolicyName": "eb-pipeline-execution",
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Effect": "Allow",
              "Action": "codepipeline:StartPipelineExecution",
              "Resource": {
                "Fn::Join": [
                  "",
                  [
                    "arn:aws:codepipeline:",
                    {
                      "Ref": "AWS::Region"
                    },
                    ":",
                    {
                      "Ref": "AWS::AccountId"
```

```
                    },
                    ":",
                    {
                      "Ref": "AppPipeline"
                    }
                  ]
                ]

...
```

2. AWS::Events::Rule CloudFormation リソースを使用して EventBridge ルールを追加します。このイベントパターンは、Amazon S3 ソースバケットでの CopyObject、PutObject、および CompleteMultipartUpload をモニタリングするイベントを作成します。さらに、パイプラインのターゲットも含めます。CopyObject、PutObject、または CompleteMultipartUpload が発生すると、このルールは、ターゲットパイプラインで StartPipelineExecution を呼び出します。

   この変更を行う理由 AWS::Events::Rule リソースを追加すると、 CloudFormation は イベントを作成できます。このリソースは CloudFormation スタックに追加されます。

   YAML

```
EventRule:
  Type: AWS::Events::Rule
  Properties:
    EventPattern:
      source:
        - aws.s3
      detail-type:
        - 'AWS API Call via CloudTrail'
      detail:
        eventSource:
          - s3.amazonaws.com
        eventName:
          - CopyObject
          - PutObject
          - CompleteMultipartUpload
        requestParameters:
          bucketName:
            - !Ref SourceBucket
          key:
            - !Ref SourceObjectKey
```

```
      Targets:
        -
          Arn:
            !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref AppPipeline ] ]
          RoleArn: !GetAtt EventRole.Arn
          Id: codepipeline-AppPipeline


...
```

JSON

```
 "EventRule": {
   "Type": "AWS::Events::Rule",
   "Properties": {
     "EventPattern": {
       "source": [
         "aws.s3"
       ],
       "detail-type": [
         "AWS API Call via CloudTrail"
       ],
       "detail": {
         "eventSource": [
           "s3.amazonaws.com"
         ],
         "eventName": [
           "CopyObject",
           "PutObject",
           "CompleteMultipartUpload"
         ],
         "requestParameters": {
           "bucketName": [
             {
               "Ref": "SourceBucket"
             }
           ],
           "key": [
             {
               "Ref": "SourceObjectKey"
             }
```

```
            ]
          }
        }
      },
      "Targets": [
        {
          "Arn": {
            "Fn::Join": [
              "",
              [
                "arn:aws:codepipeline:",
                {
                  "Ref": "AWS::Region"
                },
                ":",
                {
                  "Ref": "AWS::AccountId"
                },
                ":",
                {
                  "Ref": "AppPipeline"
                }
              ]
            ]
          },
          "RoleArn": {
            "Fn::GetAtt": [
              "EventRole",
              "Arn"
            ]
          },
          "Id": "codepipeline-AppPipeline"
        }
      ]
    }
  }
},

...
```

3. このスニペットを最初のテンプレートに追加して、クロススタック機能を有効にします。

YAML

```
Outputs:
  SourceBucketARN:
    Description: "S3 bucket ARN that Cloudtrail will use"
    Value: !GetAtt SourceBucket.Arn
    Export:
      Name: SourceBucketARN
```

JSON

```
  "Outputs" : {
    "SourceBucketARN" : {
      "Description" : "S3 bucket ARN that Cloudtrail will use",
      "Value" : { "Fn::GetAtt": ["SourceBucket", "Arn"] },
      "Export" : {
        "Name" : "SourceBucketARN"
      }
    }

...
```

4. (オプション) 特定のイメージ ID のソースオーバーライドを使用して入力トランスフォーマーを設定するには、次の YAML スニペットを使用します。次の例では、以下のオーバーライドを設定します。

   - `actionName` は (この例では `Source`)、ソースイベントから派生したものではなく、パイプラインの作成時に定義された動的な値です。
   - `revisionType` は (この例では `S3_OBJECT_VERSION_ID`)、ソースイベントから派生したものではなく、パイプラインの作成時に定義された動的な値です。
   - `revisionValue` は (この例では <*revisionValue*>)、ソースイベント変数から派生しています。

```
---
Rule: my-rule
Targets:
- Id: MyTargetId
  Arn: pipeline-ARN
  InputTransformer:
```

```
    InputPathsMap:
      revisionValue: "$.detail.object.version-id"
    InputTemplate:
      sourceRevisions:
        actionName: Source
        revisionType: S3_OBJECT_VERSION_ID
        revisionValue: '<revisionValue>'
```

5. 更新されたテンプレートをローカルコンピュータに保存し、 CloudFormation コンソールを開きます。
6. スタックを選択し、[既存スタックの変更セットの作成] を選択します。
7. 更新されたテンプレートをアップロードし、 CloudFormationに示された変更を表示します。これらがスタックに加えられる変更です。新しいリソースがリストに表示されています。
8. [実行] を選択してください。

パイプラインの PollForSourceChanges パラメータを編集するには

> **Important**
>
> このメソッドを使用してパイプラインを作成すると、PollForSourceChanges パラメータはデフォルトで true になります (ただし、明示的に false に設定した場合は除きます)。イベントベースの変更検出を追加する場合は、このパラメータを出力に追加する必要があります。ポーリングを無効にするには、このパラメータを false に設定します。そうしないと、1 つのソース変更に対してパイプラインが 2 回起動されます。詳細については、「PollForSourceChanges パラメータの有効な設定」を参照してください

- テンプレートで、PollForSourceChanges を false に変更します。パイプライン定義に PollForSourceChanges が含まれていなかった場合は、追加して false に設定します。

  この変更を行う理由 PollForSourceChanges パラメータを false に変更すると、定期的チェックがオフになるため、イベントベースの変更検出のみ使用することができます。

  YAML

```
        Name: Source
        Actions:
          -
            Name: SourceAction
```

```
            ActionTypeId:
              Category: Source
              Owner: AWS
              Version: 1
              Provider: S3
            OutputArtifacts:
              - Name: SourceOutput
            Configuration:
              S3Bucket: !Ref SourceBucket
              S3ObjectKey: !Ref SourceObjectKey
              PollForSourceChanges: false
            RunOrder: 1
```

JSON

```
{
   "Name": "SourceAction",
   "ActionTypeId": {
     "Category": "Source",
     "Owner": "AWS",
     "Version": 1,
     "Provider": "S3"
   },
   "OutputArtifacts": [
     {
       "Name": "SourceOutput"
     }
   ],
   "Configuration": {
     "S3Bucket": {
       "Ref": "SourceBucket"
     },
     "S3ObjectKey": {
       "Ref": "SourceObjectKey"
     },
     "PollForSourceChanges": false
   },
   "RunOrder": 1
 }
```

Amazon S3 パイプラインの CloudTrail リソース用に 2 番目のテンプレートを作成するには

- 別のテンプレートの でResources、、AWS::S3::BucketPolicy、および AWS::S3::BucketAWS::CloudTrail::Trail CloudFormation リソースを使用して、CloudTrail のシンプルなバケット定義と証跡を提供します。

  この変更を行う理由 CloudTrail 証跡は、アカウントあたり 5 証跡を現在の制限として、個別に作成して管理する必要があります。(「 の制限 AWS CloudTrail」を参照してください。) ただし、1 つの証跡に複数の Amazon S3 バケットを含めることができるため、いったん証跡を作成してから、必要に応じて他のパイプライン用に Amazon S3 バケットを追加できます。2 番目のサンプルテンプレートファイルに以下のコードを貼り付けます。

  YAML

```
################################################################################
# Prerequisites:
#   - S3 SourceBucket and SourceObjectKey must exist
################################################################################

Parameters:
  SourceObjectKey:
    Description: 'S3 source artifact'
    Type: String
    Default: SampleApp_Linux.zip

Resources:
  AWSCloudTrailBucketPolicy:
    Type: AWS::S3::BucketPolicy
    Properties:
      Bucket: !Ref AWSCloudTrailBucket
      PolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Sid: AWSCloudTrailAclCheck
            Effect: Allow
            Principal:
              Service:
                - cloudtrail.amazonaws.com
            Action: s3:GetBucketAcl
            Resource: !GetAtt AWSCloudTrailBucket.Arn
          -
```

```
            Sid: AWSCloudTrailWrite
            Effect: Allow
            Principal:
              Service:
                - cloudtrail.amazonaws.com
            Action: s3:PutObject
            Resource: !Join [ '', [ !GetAtt AWSCloudTrailBucket.Arn, '/
AWSLogs/', !Ref 'AWS::AccountId', '/*' ] ]
            Condition:
              StringEquals:
                s3:x-amz-acl: bucket-owner-full-control
  AWSCloudTrailBucket:
    Type: AWS::S3::Bucket
    DeletionPolicy: Retain
  AwsCloudTrail:
    DependsOn:
      - AWSCloudTrailBucketPolicy
    Type: AWS::CloudTrail::Trail
    Properties:
      S3BucketName: !Ref AWSCloudTrailBucket
      EventSelectors:
        -
          DataResources:
            -
              Type: AWS::S3::Object
              Values:
                - !Join [ '', [ !ImportValue SourceBucketARN, '/', !Ref
 SourceObjectKey ] ]
          ReadWriteType: WriteOnly
          IncludeManagementEvents: false
      IncludeGlobalServiceEvents: true
      IsLogging: true
      IsMultiRegionTrail: true


...
```

JSON

```
{
  "Parameters": {
    "SourceObjectKey": {
      "Description": "S3 source artifact",
```

```
      "Type": "String",
      "Default": "SampleApp_Linux.zip"
    }
  },
  "Resources": {
    "AWSCloudTrailBucket": {
      "Type": "AWS::S3::Bucket",
        "DeletionPolicy": "Retain"
    },
    "AWSCloudTrailBucketPolicy": {
      "Type": "AWS::S3::BucketPolicy",
      "Properties": {
        "Bucket": {
          "Ref": "AWSCloudTrailBucket"
        },
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Sid": "AWSCloudTrailAclCheck",
              "Effect": "Allow",
              "Principal": {
                "Service": [
                  "cloudtrail.amazonaws.com"
                ]
              },
              "Action": "s3:GetBucketAcl",
              "Resource": {
                "Fn::GetAtt": [
                  "AWSCloudTrailBucket",
                  "Arn"
                ]
              }
            },
            {
              "Sid": "AWSCloudTrailWrite",
              "Effect": "Allow",
              "Principal": {
                "Service": [
                  "cloudtrail.amazonaws.com"
                ]
              },
              "Action": "s3:PutObject",
              "Resource": {
```

```
              "Fn::Join": [
                "",
                [
                  {
                    "Fn::GetAtt": [
                      "AWSCloudTrailBucket",
                      "Arn"
                    ]
                  },
                  "/AWSLogs/",
                  {
                    "Ref": "AWS::AccountId"
                  },
                  "/*"
                ]
              ]
            },
            "Condition": {
              "StringEquals": {
                "s3:x-amz-acl": "bucket-owner-full-control"
              }
            }
          }
        ]
      }
    }
  },
  "AwsCloudTrail": {
    "DependsOn": [
      "AWSCloudTrailBucketPolicy"
    ],
    "Type": "AWS::CloudTrail::Trail",
    "Properties": {
      "S3BucketName": {
        "Ref": "AWSCloudTrailBucket"
      },
      "EventSelectors": [
        {
          "DataResources": [
            {
              "Type": "AWS::S3::Object",
              "Values": [
                {
                  "Fn::Join": [
```

```
                    "",
                    [
                      {
                        "Fn::ImportValue": "SourceBucketARN"
                      },
                      "/",
                      {
                        "Ref": "SourceObjectKey"
                      }
                    ]
                  ]
                }
              ]
            }
          ],
          "ReadWriteType": "WriteOnly",
          "IncludeManagementEvents": false
        }
      ],
      "IncludeGlobalServiceEvents": true,
      "IsLogging": true,
      "IsMultiRegionTrail": true
    }
  }
}

...
```

CodeCommit ソースアクションと EventBridge

CodePipeline で CodeCommit のソースアクションを追加するには、次のいずれかを選択できます。

- CodePipeline コンソールの [パイプラインの作成] ウィザード (カスタムパイプラインを作成する (コンソール))、または [アクションを編集する] ページを使用して、[CodeCommit] プロバイダオプションを選択します。このコンソールはソースが変更されたときにパイプラインを開始する EventBridge ルールを作成します。
- を使用して、アクションのCodeCommitアクション設定 AWS CLI を追加し、次のように追加のリソースを作成します。

- `CodeCommit` でのアクション設定の例を CodeCommit ソースアクションリファレンス で使用し、パイプラインを作成する (CLI) で表示されるようにアクションを作成します。
- 変更検出方法のデフォルトは、ソースをポーリングすることによってパイプラインを開始します。定期的なチェックを無効にして、手動で変更検出ルールを作成することをお勧めします。CodeCommit ソースに対する EventBridge ルールを作成する (コンソール)、CodeCommit ソースに対する EventBridge ルールを作成する (CLI)、CodeCommit ソースの EventBridge ルールを作成する (CloudFormation テンプレート) の内、いずれかの方法を使用します。

トピック

- CodeCommit ソースに対する EventBridge ルールを作成する (コンソール)
- CodeCommit ソースに対する EventBridge ルールを作成する (CLI)
- CodeCommit ソースの EventBridge ルールを作成する (CloudFormation テンプレート)

CodeCommit ソースに対する EventBridge ルールを作成する (コンソール)

> **Important**
>
> コンソールを使用してパイプラインを作成または編集する場合は、EventBridge ルールが自動的に作成されます。

CodePipeline オペレーションで使用する EventBridge ルールを作成するには

1. Amazon EventBridge コンソール (https://console.aws.amazon.com/events/) を開きます。
2. ナビゲーションペインで [ルール] を選択します。デフォルトのバスを選択したままにするか、イベントバスを選択します。[ルールを作成] を選択します。
3. [名前] で、ルールの名前を入力します。
4. [ルールタイプ] で、[イベントパターンを持つルール] を選択します。[次へ] を選択します。
5. [イベントソース] で、[AWS イベントまたは EventBridge パートナーイベント] を選択します。
6. [サンプルイベントタイプ] で [AWS イベント] を選択します。
7. [サンプルイベント] に、フィルタ処理するキーワードとして「CodeCommit」と入力します。[CodeCommit リポジトリの状態変更] を選択します。
8. [作成方法] セクションで、[カスタムパターン (JSON エディタ)] を選択します。

以下に示すイベントパターンを貼り付けます。以下は、[Event] ウィンドウに表示された、MyTestRepo リポジトリのサンプルがmain という名前のブランチを持った CodeCommit イベントパターンです。

```
{
  "source": [
    "aws.codecommit"
  ],
  "detail-type": [
    "CodeCommit Repository State Change"
  ],
  "resources": [
    "arn:aws:codecommit:us-west-2:80398EXAMPLE:MyTestRepo"
  ],
  "detail": {
    "referenceType": [
      "branch"
    ],
    "referenceName": [
      "main"
    ]
  }
}
```

9. [ターゲット] で、[CodePipeline] を選択します。
10. このルールによって開始するパイプラインの、パイプライン ARN を入力します。

> **Note**
>
> get-pipeline コマンドを実行した後、メタデータ出力でパイプライン ARN を見つけることができます。パイプライン ARN はこの形式で作成されます。
> arn:aws:codepipeline:*region*:*account*:pipeline-name
> パイプライン ARN の例:
> `arn:aws:codepipeline:us-east-2:80398EXAMPLE:MyFirstPipeline`

11. EventBridge ルールに関連付けられたターゲットを呼び出すためのアクセス許可を EventBridge に与える IAM サービスロールを作成または指定するには (この場合、ターゲットは CodePipeline):

- この特定のリソースの新しいロールを作成する を選択して、パイプラインの実行を開始するアクセス許可を EventBridge に付与するサービスロールを作成します。
- 既存のロールを使用して、パイプラインの実行を開始するアクセス許可を EventBridge に付与するサービスロールを入力します。

12. (オプション) 特定のイメージ ID でソースオーバーライドを指定するには、入力トランスフォーマーを使用してデータを JSON パラメータとして渡します。入力トランスフォーマーを使用してパイプライン変数を渡すこともできます。

    - [追加の設定] を展開します。

      [ターゲット入力を設定] で、[入力トランスフォーマーを設定] を選択します。

      ダイアログウィンドウで、[独自のサンプルイベントを入力] を選択します。[入力パス] ボックスで、次のキーと値のペアを入力します。

      ```
      {"revisionValue": "$.detail.image-digest"}
      ```

    - [テンプレート] ボックスで、次のキーと値のペアを入力します。

      ```
      {
          "sourceRevisions": [
              {
                  "actionName": "Source",
                  "revisionType": "COMMIT_ID",
                  "revisionValue": "<revisionValue>"
              }
          ],
          "variables": [
              {
                  "name": "Branch_Name",
                  "value": "value"
              }
          ]
      }
      ```

    - [確認] を選択します。

13. [次へ] を選択します。

14. [タグ] ページで、[次へ] を選択します

15. [確認と作成] ページで、ルールの設定を確認します。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。

CodeCommit ソースに対する EventBridge ルールを作成する (CLI)

put-rule コマンドを呼び出して、以下を指定します。

- 作成中のルールを一意に識別する名前。この名前は、 AWS アカウントに関連付けられた CodePipeline で作成するすべてのパイプラインで一意である必要があります。
- ルールで使用するソースと詳細フィールドのイベントパターン。詳細については、「Amazon EventBridge とイベントパターン」を参照してください。

CodeCommit をイベントソースとして、CodePipeline をターゲットとして EventBridge ルールを作成するには

1. EventBridge が CodePipeline を使用してルールを呼び出すためのアクセス許可を追加します。詳細については、 デベロッパーガイドの [Amazon EventBridge のリソースベースのポリシーを使用する] を参照してください。

   a. 次のサンプルを使用して、EventBridge にサービスロールの引き受けを許可する信頼ポリシーを作成します。信頼ポリシーに `trustpolicyforEB.json` と名前を付けます。

      JSON

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

   b. 次のコマンドを使用して、`Role-for-MyRule` ロールを作成し、信頼ポリシーをアタッチします。

```
aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document
 file://trustpolicyforEB.json
```

c. 次のサンプルに示すように、MyFirstPipeline というパイプラインに対して、アクセス権限ポリシー JSON を作成します。アクセス権限ポリシーに permissionspolicyforEB.json と名前を付けます。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:StartPipelineExecution"
            ],
            "Resource": [
                "arn:aws:codepipeline:us-
west-2:111122223333:MyFirstPipeline"
            ]
        }
    ]
}
```

d. 次のコマンドを使用して、Role-for-MyRule ロールに CodePipeline-Permissions-Policy-for-EB アクセス権限ポリシーをアタッチします。

この変更を行う理由 ロールにこのポリシーを追加すると、EventBridge に対するアクセス許可が作成されます。

```
aws iam put-role-policy --role-name Role-for-MyRule --policy-name CodePipeline-
Permissions-Policy-For-EB --policy-document file://permissionspolicyforEB.json
```

2. put-rule コマンドを呼び出し、--name、--event-pattern、--role-arn パラメータを含めます。

この変更を行う理由 このコマンドでは、 CloudFormation でイベントを作成することができます。

次のサンプルコマンドは、MyCodeCommitRepoRule というルールを作成します。

```
aws events put-rule --name "MyCodeCommitRepoRule" --event-pattern "{\"source\":
[\"aws.codecommit\"],\"detail-type\":[\"CodeCommit Repository State Change\"],
\"resources\":[\"repository-ARN\"],\"detail\":{\"referenceType\":[\"branch\"],
\"referenceName\":[\"main\"]}}" --role-arn "arn:aws:iam::ACCOUNT_ID:role/Role-for-
MyRule"
```

3. CodePipeline をターゲットとして追加するには、put-targets コマンドを呼び出し、次のパラメータを含めます。

   - --rule パラメータは、put-rule を使用して作成した rule_name で使用されます。
   - --targets パラメータは、ターゲットリストのリスト Id とターゲットパイプラインの ARN で使用されます。

   次のサンプルコマンドでは、MyCodeCommitRepoRule と呼ばれるルールに対して指定し、ターゲット Id は 1 番で構成されています。これは、ルールのターゲットのリストが何であるかを示し、この場合は ターゲット 1 です。このサンプルコマンドでは、パイプラインのサンプルの ARN も指定されます。パイプラインは、リポジトリ内に変更が加えられると開始します。

```
aws events put-targets --rule MyCodeCommitRepoRule --targets
 Id=1,Arn=arn:aws:codepipeline:us-west-2:80398EXAMPLE:TestPipeline
```

4. (オプション) 特定のイメージ ID のソースオーバーライドを使用して入力トランスフォーマーを設定するには、CLI コマンドで次の JSON を使用します。次の例では、以下のオーバーライドを設定します。

   - actionName は (この例では Source)、ソースイベントから派生したものではなく、パイプラインの作成時に定義された動的な値です。
   - revisionType は (この例では COMMIT_ID)、ソースイベントから派生したものではなく、パイプラインの作成時に定義された動的な値です。
   - revisionValue は (この例では <*revisionValue*>)、ソースイベント変数から派生しています。

```
{
    "Rule": "my-rule",
    "Targets": [
```

```
        {
            "Id": "MyTargetId",
            "Arn": "pipeline-ARN",
            "InputTransformer": {
                "sourceRevisions": {
                    "actionName": "Source",
                    "revisionType": "COMMIT_ID",
                    "revisionValue": "<revisionValue>"
                },
                "variables": [
                    {
                        "name": "Branch_Name",
                        "value": "value"
                    }
                ]
            }
        }
    ]
}
```

パイプラインの PollForSourceChanges パラメータを編集するには

> **Important**
>
> このメソッドを使用してパイプラインを作成すると、PollForSourceChanges パラメータはデフォルトで true になります (ただし、明示的に false に設定した場合は除きます)。イベントベースの変更検出を追加する場合は、このパラメータを出力に追加する必要があります。ポーリングを無効にするには、このパラメータを false に設定します。そうしないと、1 つのソース変更に対してパイプラインが 2 回起動されます。詳細については、「PollForSourceChanges パラメータの有効な設定」を参照してください

1. get-pipeline コマンドを実行して、パイプライン構造を JSON ファイルにコピーします。例えば、MyFirstPipeline という名前のパイプラインに対して、以下のコマンドを実行します。

   ```
   aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
   ```

   このコマンドは何も返しませんが、作成したファイルは、コマンドを実行したディレクトリにあります。

2. 任意のプレーンテキストエディタで JSON ファイルを開き、以下に示しているように、PollForSourceChanges パラメータを false に変更してソースステージを編集します。

   この変更を行う理由 このパラメータを false に変更すると、定期的チェックがオフになるため、イベントベースの変更検出のみ使用することができます。

   ```
   "configuration": {
       "PollForSourceChanges": "false",
       "BranchName": "main",
       "RepositoryName": "MyTestRepo"
   },
   ```

3. get-pipeline コマンドを使用して取得したパイプライン構造を使用している場合、JSON ファイルから metadata 行を削除します。それ以外の場合は、update-pipeline コマンドで使用することはできません。"metadata": { } 行と、"created"、"pipelineARN"、"updated" フィールドを削除します。

   例えば、構造から以下の行を削除します。

   ```
   "metadata": {
       "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
       "created": "date",
       "updated": "date"
   },
   ```

   ファイルを保存します。

4. 変更を適用するには、パイプライン JSON ファイルを指定して、update-pipeline コマンドを実行します。

   > **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 codepipeline update-pipeline --cli-input-json file://pipeline.json
   ```

   このコマンドは、編集したパイプラインの構造全体を返します。

> **Note**
>
> update-pipeline コマンドは、パイプラインを停止します。update-pipeline コマンドを実行したときにパイプラインによりリビジョンが実行されている場合、その実行は停止します。更新されたパイプラインによりそのリビジョンを実行するには、パイプラインを手動で開始する必要があります。パイプラインを手動で開始するには **start-pipeline-execution** コマンドを使用します。

CodeCommit ソースの EventBridge ルールを作成する (CloudFormation テンプレート)

CloudFormation を使用してルールを作成するには、次に示すようにテンプレートを更新します。

パイプライン CloudFormation テンプレートを更新して EventBridge ルールを作成するには

1. テンプレートの でResources、 AWS::IAM::Role CloudFormation リソースを使用して、イベントがパイプラインを開始できるようにする IAM ロールを設定します。このエントリによって、2 つのポリシーを使用するロールが作成されます。

   - 最初のポリシーでは、ロールを引き受けることを許可します。
   - 2 つめのポリシーでは、パイプラインを開始するアクセス権限が付与されます。

   この変更を行う理由 AWS::IAM::Role リソースを追加すると、 CloudFormation は EventBridge のアクセス許可を作成できます。このリソースは CloudFormation スタックに追加されます。

   YAML

```
EventRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: 2012-10-17
      Statement:
        -
          Effect: Allow
```

```
            Principal:
              Service:
                - events.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref
 'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]
```

JSON

```
"EventRole": {
  "Type": "AWS::IAM::Role",
  "Properties": {
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": [
              "events.amazonaws.com"
            ]
          },
          "Action": "sts:AssumeRole"
        }
      ]
    },
    "Path": "/",
    "Policies": [
      {
        "PolicyName": "eb-pipeline-execution",
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
```

```
            {
              "Effect": "Allow",
              "Action": "codepipeline:StartPipelineExecution",
              "Resource": {
                "Fn::Join": [
                  "",
                  [
                    "arn:aws:codepipeline:",
                    {
                      "Ref": "AWS::Region"
                    },
                    ":",
                    {
                      "Ref": "AWS::AccountId"
                    },
                    ":",
                    {
                      "Ref": "AppPipeline"
                    }
                  ]

...
```

2. テンプレートの でResources、 AWS::Events::Rule CloudFormation リソースを使用して EventBridge ルールを追加します。このイベントパターンは、リポジトリへの変更のプッシュをモニタリングするイベントを作成します。EventBridge でリポジトリの状態の変更が検出されると、ルールはターゲットパイプラインで StartPipelineExecution を呼び出します。

   この変更を行う理由 AWS::Events::Rule リソースを追加すると、 CloudFormation は イベントを作成できます。このリソースは CloudFormation スタックに追加されます。

   YAML

```
  EventRule:
    Type: AWS::Events::Rule
    Properties:
      EventPattern:
        source:
          - aws.codecommit
        detail-type:
          - 'CodeCommit Repository State Change'
        resources:
```

```
          - !Join [ '', [ 'arn:aws:codecommit:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref RepositoryName ] ]
        detail:
          event:
            - referenceCreated
            - referenceUpdated
          referenceType:
            - branch
          referenceName:
            - main
      Targets:
        -
          Arn:
            !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref AppPipeline ] ]
          RoleArn: !GetAtt EventRole.Arn
          Id: codepipeline-AppPipeline
```

JSON

```
"EventRule": {
  "Type": "AWS::Events::Rule",
  "Properties": {
    "EventPattern": {
      "source": [
        "aws.codecommit"
      ],
      "detail-type": [
        "CodeCommit Repository State Change"
      ],
      "resources": [
        {
          "Fn::Join": [
            "",
            [
              "arn:aws:codecommit:",
              {
                "Ref": "AWS::Region"
              },
              ":",
              {
                "Ref": "AWS::AccountId"
```

```
              },
              ":",
              {
                "Ref": "RepositoryName"
              }
            ]
          ]
        }
      ],
      "detail": {
        "event": [
          "referenceCreated",
          "referenceUpdated"
        ],
        "referenceType": [
          "branch"
        ],
        "referenceName": [
          "main"
        ]
      }
    },
    "Targets": [
      {
        "Arn": {
          "Fn::Join": [
            "",
            [
              "arn:aws:codepipeline:",
              {
                "Ref": "AWS::Region"
              },
              ":",
              {
                "Ref": "AWS::AccountId"
              },
              ":",
              {
                "Ref": "AppPipeline"
              }
            ]
          ]
        },
        "RoleArn": {
```

```
          "Fn::GetAtt": [
            "EventRole",
            "Arn"
          ]
        },
        "Id": "codepipeline-AppPipeline"
      }
    ]
  }
},
```

3. (オプション) 特定のイメージ ID のソースオーバーライドを使用して入力トランスフォーマーを設定するには、次の YAML スニペットを使用します。次の例では、以下のオーバーライドを設定します。

   - `actionName` は (この例では `Source`)、ソースイベントから派生したものではなく、パイプラインの作成時に定義された動的な値です。
   - `revisionType` は (この例では `COMMIT_ID`)、ソースイベントから派生したものではなく、パイプラインの作成時に定義された動的な値です。
   - `revisionValue` は (この例では *<revisionValue>*)、ソースイベント変数から派生しています。
   - `BranchName` および `Value` の出力変数が指定されます。

```
Rule: my-rule
Targets:
- Id: MyTargetId
  Arn: pipeline-ARN
  InputTransformer:
    sourceRevisions:
      actionName: Source
      revisionType: COMMIT_ID
      revisionValue: <revisionValue>
    variables:
    - name: BranchName
      value: value
```

4. 更新されたテンプレートをローカルコンピュータに保存し、 CloudFormation コンソールを開きます。
5. スタックを選択し、[既存スタックの変更セットの作成] を選択します。

6. テンプレートをアップロードし、 CloudFormationに示された変更を表示します。これらがスタックに加えられる変更です。新しいリソースがリストに表示されています。
7. [実行] を選択してください。

パイプラインの PollForSourceChanges パラメータを編集するには

> **Important**
>
> 多くの場合、パイプラインの作成時に PollForSourceChanges パラメータはデフォルトで true になります。イベントベースの変更検出を追加する場合は、このパラメータを出力に追加する必要があります。ポーリングを無効にするには、このパラメータを false に設定します。そうしないと、1 つのソース変更に対してパイプラインが 2 回起動されます。詳細については、「PollForSourceChanges パラメータの有効な設定」を参照してください

- テンプレートで、PollForSourceChanges を false に変更します。パイプライン定義に PollForSourceChanges が含まれていなかった場合は、追加して false に設定します。

  この変更を行う理由 このパラメータを false に変更すると、定期的チェックがオフになるため、イベントベースの変更検出のみ使用することができます。

  YAML

```
        Name: Source
        Actions:
          -
            Name: SourceAction
            ActionTypeId:
              Category: Source
              Owner: AWS
              Version: 1
              Provider: CodeCommit
            OutputArtifacts:
              - Name: SourceOutput
            Configuration:
              BranchName: !Ref BranchName
              RepositoryName: !Ref RepositoryName
              PollForSourceChanges: false
            RunOrder: 1
```

JSON

```
{
  "Name": "Source",
  "Actions": [
    {
      "Name": "SourceAction",
      "ActionTypeId": {
        "Category": "Source",
        "Owner": "AWS",
        "Version": 1,
        "Provider": "CodeCommit"
      },
      "OutputArtifacts": [
        {
          "Name": "SourceOutput"
        }
      ],
      "Configuration": {
        "BranchName": {
          "Ref": "BranchName"
        },
        "RepositoryName": {
          "Ref": "RepositoryName"
        },
        "PollForSourceChanges": false
      },
      "RunOrder": 1
    }
  ]
},
```

を使用してパイプラインにサードパーティーのソースプロバイダーを追加する AWS CodeConnections

AWS CodePipeline コンソールまたは を使用して AWS CLI 、パイプラインをサードパーティーのリポジトリに接続できます。

> **Note**
>
> コンソールを使用してパイプラインを作成または編集すると、変更検出リソースが作成されます。 AWS CLI を使用してパイプラインを作成する場合は、追加のリソースを自分で作成する必要があります。詳細については、「CodeCommit ソースアクションと EventBridge」を参照してください。

トピック

- Azure DevOps 接続
- Bitbucket Cloud への接続
- GitHub コネクション
- GitHub Enterprise Server 接続
- GitLab.com への接続
- GitLab セルフマネージドの接続
- 別のアカウントと共有されている接続を使用する

Azure DevOps 接続

接続を使用すると、サードパーティープロバイダーを AWS リソースに関連付ける設定を承認して確立できます。サードパーティーのリポジトリをパイプラインのソースとして関連付けるには、接続を使用します。

> **Note**
>
> アカウントで既存の接続を作成または使用する代わりに、別の AWS アカウント間の共有接続を使用できます。「別のアカウントと共有されている接続を使用する」を参照してください。

> **Note**
>
> この機能は、アジアパシフィック (香港) 、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 (ジャカルタ) 、アジアパシフィック (メルボルン) 、アジアパシフィック (大阪) 、アフリカ (ケープタウン) 、中東 (バーレーン) 、中東 (アラブ首長国連邦) 、欧州 (スペイン) 、欧州 (チューリッヒ) 、イスラエル (テルアビブ) 、または AWS GovCloud (米国西部) の各リージョンでは使用できません。利用可能なその他のアクションについては、「CodePipeline との製品とサービスの統合」を参照してください。欧州 (ミラノ) リージョンでのこのアクションに関する考慮事項については、「CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアクションの場合)」の注意を参照してください。

CodePipeline で Azure DevOps のソースアクションを追加するには、次のいずれかを選択します。

- CodePipeline コンソールの [パイプラインの作成] ウィザード、または [アクションを編集する] ページを使用して、[Azure DevOps] プロバイダーオプションを選択します。アクションを追加するには Bitbucket Cloud への接続を作成する (コンソール) を参照してください。このコンソールは、接続リソースの作成に役立ちます。
- CLI を使用して、次の通りに `CreateSourceConnection` でのアクションのアクション設定を `Azure DevOps` プロバイダに追加します。
  - 接続リソースを作成するには、Azure DevOps への接続を作成する (CLI) を参照してCLI で接続リソースを作成します。
  - `CreateSourceConnection` でのアクション設定の例を CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアクションの場合) で使用し、パイプラインを作成する (CLI) で表示されるようにアクションを追加します。

> **Note**
>
> [設定] からデベロッパーツール コンソールを使用して、接続を作成することもできます。[接続を作成する] を参照してください。

開始する前に:

- Azure DevOps など、サードパーティーのリポジトリプロバイダでアカウントを作成しておく必要があります。
- Azure DevOps リポジトリなどのサードパーティーのコードリポジトリを既に作成しておく必要があります。

> **Note**
>
> Azure DevOps 接続は、接続の作成に使用された Azure DevOps アカウントで所有するリポジトリへのアクセスだけを提供します。
> 接続を作成するには、Azure アカウントの管理者アクセス許可が必要です。

トピック

- Azure DevOps への接続を作成する (コンソール)
- Azure DevOps への接続を作成する (CLI)

Azure DevOps への接続を作成する (コンソール)

次の手順を使用して、CodePipeline コンソールを使用して Azure DevOps リポジトリに接続アクションを追加します。

ステップ 1 : パイプラインを作成または修正するには

パイプラインを作成または修正するには

1. CodePipeline コンソールにサインインします。
2. 次のいずれかを選択します 。
   - パイプラインの作成を選択します。[パイプラインを作成する] の手順に従い最初の画面を完了し、[次] を選択します。[ソース] ページの [ソースプロバイダー] で、[Azure DevOps] を選択します。
   - 既存のパイプラインを編集することを選択します。[Edit]、[Edit Stage] の順に選択します。ソースアクションを追加または編集するかを選択します。[アクションを編集する] ページで、[アクション名] に自分のアクション名を入力します。[アクションプロバイダー] で、[Azure DevOps] を選択します。
3. 次のいずれかを行います:

- プロバイダーへの接続をまだ作成していない場合は、[接続]で [Azure DevOps に接続] を選択します。「ステップ 2: Azure DevOps への接続を作成する」に進みます。
- [接続] でプロバイダへの接続を既に作成している場合は、その接続を選択します。ステップ 3： 接続のソースアクションを保存するに進みます。

ステップ 2: Azure DevOps への接続を作成する

Azure DevOps への接続を作成するには

1. Azure DevOps リポジトリへの接続を作成するには、[プロバイダーを選択する] で、[Azure DevOps] を選択します。[接続名] に、作成する接続の名前を入力します。 [Azure DevOps に接続] を選択して、ステップ 2 に進みます。

Add source stage Info

Step 3 of 7

Source

Source provider
This is where you stored your input artifacts for your pipeline. Choose the provider and then provide the connection details.

Azure DevOps

Connection
Choose an existing connection that you have already configured, or create a new one and then return to this task.

or Connect to AzureDevOps

Owner
Choose an owner in your Azure DevOps account.

Repository name
Choose a repository in your Azure DevOps account.

You can type or paste the group path to any project that the provided credentials can access. Use the format 'group/subgroup/project'.

Default branch
Default branch will be used only when pipeline execution starts from a different source or manually started.

Output artifact format
Choose the output artifact format.

- CodePipeline default
  AWS CodePipeline uses the default zip format for artifacts in the pipeline. Does not include Git metadata about the repository.
- Full clone
  AWS CodePipeline passes metadata about the repository that allows subsequent actions to do a full Git clone. Only supported for AWS CodeBuild actions. Learn more

Enable automatic retry on stage failure

Webhook events

Webhook - *optional*

Start your pipeline on push and pull request events.

Webhook event filters - *optional*
Starts your pipeline on a specific event.

Remove filters

Cancel Previous Next

2. [Azure DevOps に接続] 設定ページに、接続名が表示されます。

   [Azure DevOps アプリ] で、アプリケーションのインストールを選択するか、[新しいアプリをインストールする] を選択してアプリケーションを作成します。

   > **Note**
   >
   > アプリは、Azure DevOps アカウントごとに 1 回だけインストールします。コネクタアプリを既にインストールしている場合は、それを選択してこのセクションの最後のステップに移動します。

3. Microsoft のログインページが表示されたら、認証情報を使用してログインし、続行を選択します。
4. アプリのインストールページで、コネクタアプリが Azure DevOps アカウントに接続しようとしていることを示すメッセージが表示されます。

   [アクセス権の付与] を選択します。
5. 接続ページで、新規インストールの接続 ID が表示されます。接続 を選択します。作成された接続が接続リストに表示されます。

ステップ 3: Azure DevOps ソースアクションを保存する

ウィザードで次の手順を使用するか、[アクションを編集] ページで、ソースアクションを接続情報とともに保存します。

接続でソースアクションを完了して保存するには

1. [リポジトリ名] で、サードパーティーのリポジトリの名前を選択します。
2. パイプライントリガーでは、アクションが AWS CodeConnections アクションである場合にトリガーを追加できます。パイプライントリガー設定を構成し、必要に応じてトリガーでフィルタリングするには、「コードプッシュまたはプルリクエストイベントタイプを使用してトリガーを追加する」で詳細を参照してください。
3. [Output artifact format (出力アーティファクトのフォーマット)] で、アーティファクトのフォーマットを選択する必要があります。

   - デフォルトのメソッドを使用して Azure DevOps アクションからの出力アーティファクトを保存するには、[CodePipeline のデフォルト] を選択します。アクションは、リポジトリから

ファイルにアクセスして、パイプラインアーティファクトストアの ZIP ファイルにアーティファクトを保存します。

- リポジトリへの URL 参照を含む JSON ファイルを保存して、ダウンストリームのアクションで Git コマンドを直接実行できるようにするには、[Full clone (フルクローン)] を選択します。このオプションは、CodeBuild ダウンストリームアクションでのみ使用できます。

  このオプションを選択した場合は、Bitbucket、GitHub、GitHub Enterprise Server、または GitLab.com に接続するための CodeBuild GitClone アクセス許可を追加します。 で示されるように CodeBuild プロジェクトサービスロールの権限を更新する必要があります。

4. ウィザード上で [次へ] または [保存] を [アクションを編集] ページで選択します。

Azure DevOps への接続を作成する (CLI)

AWS Command Line Interface (AWS CLI) を使用して接続を作成できます。

これを行うには、create-connection コマンドを使用します。

> **Important**
>
> AWS CLI または を通じて作成された接続 AWS CloudFormation は、デフォルトで PENDINGステータスです。CLI または との接続を作成したら CloudFormation、コンソールを使用して接続を編集し、ステータスを にしますAVAILABLE。

接続を作成するには

1. ターミナル (Linux/macOS/Unix) またはコマンドプロンプト (Windows) を開きます。を使用して create-connection コマンド AWS CLI を実行し、接続--connection-nameの --provider-typeと を指定します。この例では、サードパーティープロバイダー名は AzureDevOps で、指定された接続名は MyConnection です。

   ```
   aws codeconnections create-connection --provider-type AzureDevOps --connection-name
    MyConnection
   ```

   成功した場合、このコマンドは次のような接続 ARN 情報を返します。

   ```
   {
   ```

```
    "ConnectionArn": "arn:aws:codeconnections:us-west-2:account_id:connection/
aEXAMPLE-8aad-4d5d-8878-dfcab0bc441f"
}
```

2. コンソールを使用して接続を完了します。詳細については、「Update a pending connection」を参照してください。
3. パイプラインはデフォルトで、接続ソースリポジトリへのコードのプッシュ時に変更を検出するようになっています。パイプライントリガーを手動リリース用または Git タグ用に設定するには、以下のいずれかを行います。

   - 手動リリースでのみ開始するようにパイプラインのトリガー設定を構成するには、設定に以下の行を追加します。

     ```
     "DetectChanges": "false",
     ```

   - トリガーでフィルタリングするようにパイプライントリガー設定を構成する方法の詳細については、「コードプッシュまたはプルリクエストイベントタイプを使用してトリガーを追加する」を参照してください。例えば、以下では、パイプライン JSON 定義のパイプラインレベルに Git タグを追加します。この例では、`release-v0` と `release-v1` が包含する Git タグで、`release-v2` が除外する Git タグです。

     ```
     "triggers": [
                {
                    "providerType": "CodeStarSourceConnection",
                    "gitConfiguration": {
                        "sourceActionName": "Source",
                        "push": [
                            {
                                "tags": {
                                    "includes": [
                                        "release-v0", "release-v1"
                                    ],
                                    "excludes": [
                                        "release-v2"
                                    ]
                                }
                            }
                        ]
                    }
                }
            ]
     ```

Bitbucket Cloud への接続

接続を使用すると、サードパーティープロバイダーを AWS リソースに関連付ける設定を承認して確立できます。サードパーティーのリポジトリをパイプラインのソースとして関連付けるには、接続を使用します。

> **Note**
>
> アカウントで既存の接続を作成または使用する代わりに、別の AWS アカウント間の共有接続を使用できます。「別のアカウントと共有されている接続を使用する」を参照してください。

> **Note**
>
> この機能は、アジアパシフィック (香港)、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 (ジャカルタ)、アジアパシフィック (メルボルン)、アジアパシフィック (大阪)、アフリカ (ケープタウン)、中東 (バーレーン)、中東 (アラブ首長国連邦)、欧州 (スペイン)、欧州 (チューリッヒ)、イスラエル (テルアビブ)、または AWS GovCloud (米国西部) の各リージョンでは使用できません。利用可能なその他のアクションについては、「CodePipeline との製品とサービスの統合」を参照してください。欧州 (ミラノ) リージョンでのこのアクションに関する考慮事項については、「CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアクションの場合)」の注意を参照してください。

CodePipeline で Bitbucket Cloud のソースアクションを追加するには、次のいずれかを選択できます。

- CodePipeline コンソール [Create pipeline] ウィザードまたは [Edit action] ページを使用し、[Bitbucket] プロバイダオプションを選択します。アクションを追加するには Bitbucket Cloud への接続を作成する (コンソール) を参照してください。このコンソールは、接続リソースの作成に役立ちます。

> **Note**
>
> Bitbucket Cloud リポジトリへの接続を作成できます。Bitbucket サーバーなど、インストールされている Bitbucket プロバイダーのタイプはサポートされていません。

- CLI を使用して、次の通りに `CreateSourceConnection` でのアクションのアクション設定を `Bitbucket` プロバイダに追加します。
  - 接続リソースを作成するには、Bitbucket Cloud への接続を作成する (CLI) を参照してCLI で接続リソースを作成します。
  - `CreateSourceConnection` でのアクション設定の例を CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアクションの場合) で使用し、パイプラインを作成する (CLI) で表示されるようにアクションを追加します。

> **Note**
>
> [設定] からデベロッパーツール コンソールを使用して、接続を作成することもできます。[接続を作成する] を参照してください。

開始する前に:

- Bitbucket Cloud など、サードパーティーのリポジトリプロバイダーでアカウントを作成しておく必要があります。
- Bitbucket Cloud リポジトリなどのサードパーティーのコードリポジトリを既に作成しておく必要があります。

> **Note**
>
> Bitbucket Cloud 接続は、接続の作成に使用された Bitbucket Cloud アカウントが所有するリポジトリへのアクセスのみを提供します。
> アプリケーションを Bitbucket Cloud ワークスペースにインストールする場合は、ワークスペースの管理アクセス許可が必要です。アクセス許可がないと、アプリケーションをインストールするオプションは表示されません。

トピック

- Bitbucket Cloud への接続を作成する (コンソール)
- Bitbucket Cloud への接続を作成する (CLI)

Bitbucket Cloud への接続を作成する (コンソール)

次の手順を使用して、CodePipeline コンソールを使用して Bitbucket リポジトリに接続アクションを追加します。

> **Note**
>
> Bitbucket Cloud リポジトリへの接続を作成できます。Bitbucket サーバーなど、インストールされている Bitbucket プロバイダーのタイプはサポートされていません。

ステップ 1 : パイプラインを作成または修正するには

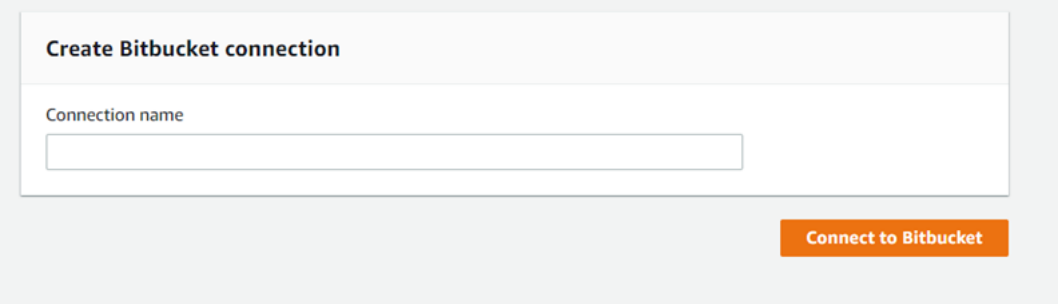
パイプラインを作成または修正するには

1. CodePipeline コンソールにサインインします。
2. 次のいずれかを選択します 。
   - パイプラインの作成を選択します。[パイプラインを作成する] の手順に従い最初の画面を完了し、[次] を選択します。 [ソース] ページの [ソースプロバイダー] の下の [Bitbucket] を選択します。
   - 既存のパイプラインを編集することを選択します。[Edit]、[Edit Stage] の順に選択します。ソースアクションを追加または編集するかを選択します。[アクションの編集] ページで、[Action name (アクション名)] に自分のアクション名を入力します。[アクションプロバイダ] で、[Bitbucket] を選択します。
3. 次のいずれかを行います:
   - [接続] でプロバイダへの接続をまだ作成していない場合は、[Bitbucket への接続] を選択します。ステップ 2 : Bitbucket に接続に進みます。
   - [接続] でプロバイダへの接続を既に作成している場合は、その接続を選択します。ステップ 3 : 接続のソースアクションを保存するに進みます。

ステップ 2: Bitbucket への接続を作成する

Bitbucket Cloud への接続を作成するには

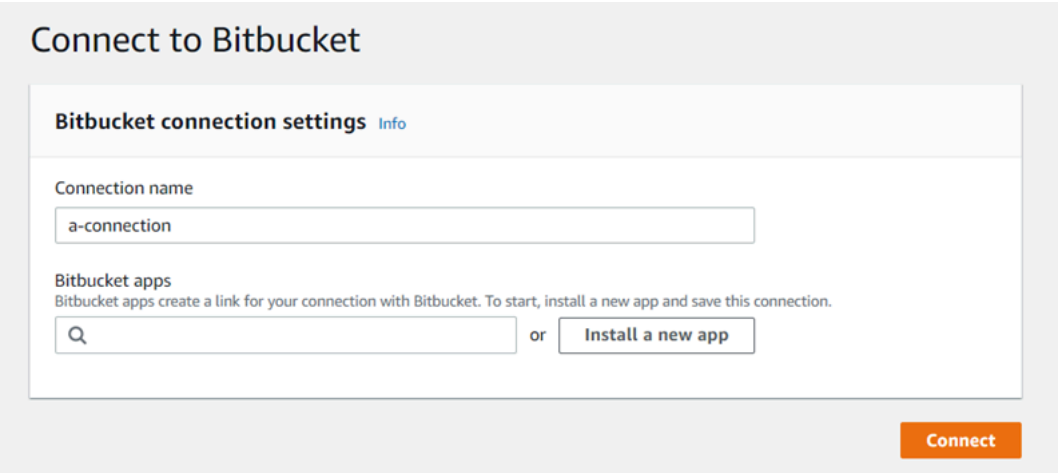
1. リポジトリの [Bitbucket への接続] 設定ページで、接続名を入力し、[Bitbucket への接続] を選択します。



   [Bitbucket アプリ] フィールドが表示されます。

2. [Bitbucket apps] (Bitbucket アプリ) で、アプリのインストールを選択するか、アプリを作成するために [Install a new app] (新しいアプリをインストールする) を選択します。

> **Note**
>
> アプリケーションは、Bitbucket Cloud ワークスペースまたはアカウントごとに 1 回だけインストールします。Bitbucket アプリを既にインストールしている場合は、それを選択してステップ 4 に移動します。



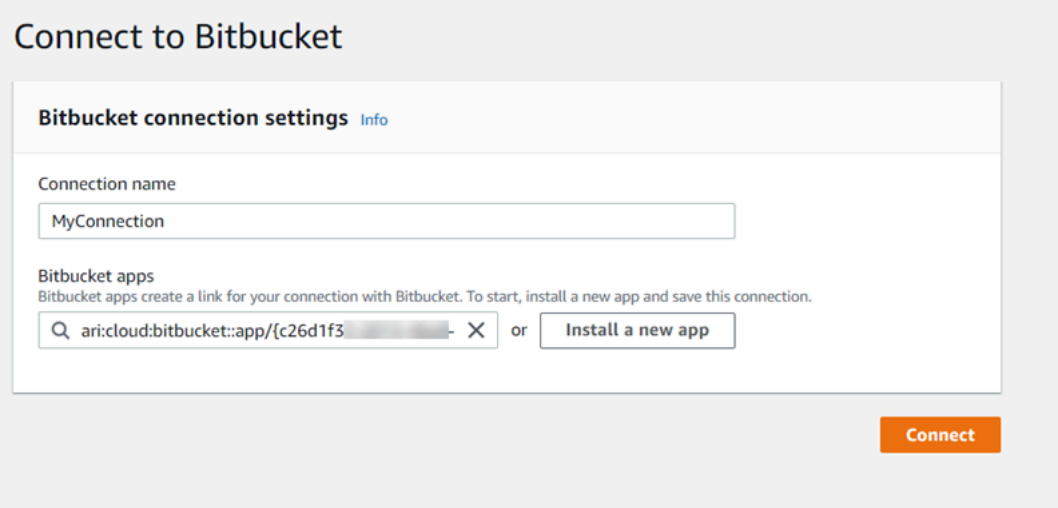
3. Bitbucket Cloud のログインページが表示されたら、認証情報を使用してログインし、続行を選択します。

4. アプリのインストールページで、 AWS CodeStar アプリが Bitbucket アカウントに接続しようとしていることを示すメッセージが表示されます。

   Bitbucket ワークスペースを使用している場合は、[Authorize for] (承認対象) オプションをそのワークスペースに変更します。管理者権限のあるワークスペースのみが表示されます。

   [アクセス権の付与] を選択します。

5. Bitbucketアプリには、新規インストールの接続 ID が表示されます。接続 を選択します。作成された接続が接続リストに表示されます。



ステップ 3: Bitbucket Cloud のソースアクションを保存する

ウィザードで次の手順を使用するか、[アクションを編集] ページで、ソースアクションを接続情報とともに保存します。

接続でソースアクションを完了して保存するには

1. [リポジトリ名] で、サードパーティーのリポジトリの名前を選択します。
2. パイプライントリガーでは、アクションが AWS CodeConnections アクションである場合にトリガーを追加できます。パイプライントリガー設定を構成し、必要に応じてトリガーでフィルタリングするには、「コードプッシュまたはプルリクエストイベントタイプを使用してトリガーを追加する」で詳細を参照してください。
3. [Output artifact format (出力アーティファクトのフォーマット)] で、アーティファクトのフォーマットを選択する必要があります。

- デフォルトのメソッドを使用して Bitbucket Cloud アクションからの出力アーティファクトを保存するには、[CodePipeline デフォルト] を選択します。アクションは、Bitbucket Cloud リポジトリからファイルにアクセスし、パイプラインアーティファクトストアの ZIP ファイルにアーティファクトを保存します。
- リポジトリへの URL 参照を含む JSON ファイルを保存して、ダウンストリームのアクションで Git コマンドを直接実行できるようにするには、[Full clone (フルクローン)] を選択します。このオプションは、CodeBuild ダウンストリームアクションでのみ使用できます。

  このオプションを選択した場合は、Bitbucket、GitHub、GitHub Enterprise Server、または GitLab.com に接続するための CodeBuild GitClone アクセス許可を追加します。 で示されるように CodeBuild プロジェクトサービスロールの権限を更新する必要があります。

4. ウィザード上で [次へ] または [保存] を [アクションを編集] ページで選択します。

Bitbucket Cloud への接続を作成する (CLI)

AWS Command Line Interface (AWS CLI) を使用して接続を作成できます。

> **Note**
>
> Bitbucket Cloud リポジトリへの接続を作成できます。Bitbucket サーバーなど、インストールされている Bitbucket プロバイダーのタイプはサポートされていません。

これを行うには、create-connection コマンドを使用します。

> **Important**
>
> AWS CLI または を通じて作成された接続 AWS CloudFormation は、デフォルトで `PENDING`ステータスです。CLI または との接続を作成したら CloudFormation、コンソールを使用して接続を編集し、ステータスを にします`AVAILABLE`。

接続を作成するには

1. ターミナル (Linux/macOS/Unix) またはコマンドプロンプト (Windows) を開きます。を使用して create-connection コマンド AWS CLI を実行し、接続`--connection-name`の `--provider-`

typeと を指定します。この例では、サードパーティープロバイダー名は Bitbucket で、指定された接続名は MyConnection です。

```
aws codestar-connections create-connection --provider-type Bitbucket --connection-name MyConnection
```

成功した場合、このコマンドは次のような接続 ARN 情報を返します。

```
{
    "ConnectionArn": "arn:aws:codestar-connections:us-west-2:account_id:connection/aEXAMPLE-8aad-4d5d-8878-dfcab0bc441f"
}
```

2. コンソールを使用して接続を完了します。詳細については、「Update a pending connection」を参照してください。
3. パイプラインはデフォルトで、接続ソースリポジトリへのコードのプッシュ時に変更を検出するようになっています。パイプライントリガーを手動リリース用または Git タグ用に設定するには、以下のいずれかを行います。

   - 手動リリースでのみ開始するようにパイプラインのトリガー設定を構成するには、設定に以下の行を追加します。

     ```
     "DetectChanges": "false",
     ```

   - トリガーでフィルタリングするようにパイプライントリガー設定を構成する方法の詳細については、「コードプッシュまたはプルリクエストイベントタイプを使用してトリガーを追加する」を参照してください。例えば、以下では、パイプライン JSON 定義のパイプラインレベルに Git タグを追加します。この例では、release-v0 と release-v1 が包含する Git タグで、release-v2 が除外する Git タグです。

     ```
     "triggers": [
                {
                    "providerType": "CodeStarSourceConnection",
                    "gitConfiguration": {
                        "sourceActionName": "Source",
                        "push": [
                            {
                                "tags": {
                                    "includes": [
                                        "release-v0", "release-v1"
     ```

```
                            ],
                            "excludes": [
                                "release-v2"
                            ]
                        }
                    }
                ]
            }
        }
    ]
```

GitHub コネクション

接続を使用して、サードパーティープロバイダーを AWS リソースに関連付ける設定を承認および確立します。

> **Note**
>
> アカウントで既存の接続を作成または使用する代わりに、別の AWS アカウント間の共有接続を使用できます。「別のアカウントと共有されている接続を使用する」を参照してください。

> **Note**
>
> この機能は、アジアパシフィック (香港)、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 (ジャカルタ)、アジアパシフィック (メルボルン)、アジアパシフィック (大阪)、アフリカ (ケープタウン)、中東 (バーレーン)、中東 (アラブ首長国連邦)、欧州 (スペイン)、欧州 (チューリッヒ)、イスラエル (テルアビブ)、または AWS GovCloud (米国西部) の各リージョンでは使用できません。利用可能なその他のアクションについては、「CodePipeline との製品とサービスの統合」を参照してください。欧州 (ミラノ) リージョンでのこのアクションに関する考慮事項については、「CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアクションの場合)」の注意を参照してください。

CodePipeline で GitHub または GitHub Enterprise Cloud リポジトリのソースアクションを追加するには、次のいずれかを選択できます。

- CodePipeline コンソールのパイプラインを作成ウィザード、または [アクションを編集する] ページを使用して、[GitHub (GitHub アプリ経由)] プロバイダオプションを選択します。アクションを追加するには GitHub (コンソール) への接続を作成する を参照してください。このコンソールは、接続リソースの作成に役立ちます。

  > **Note**
  >
  > GitHub 接続を追加し、パイプラインの [完全クローン作成] オプションを使用してメタデータをクローンする方法を示すチュートリアルについては、「チュートリアル: CodeCommit パイプラインソースで完全なクローンを使用する」を参照してください。

- `CodeStarSourceConnection` アクションのアクション設定を `GitHub` プロバイダに追加するには、 パイプラインを作成する (CLI) に記載されている CLI 手順に従います。

> **Note**
>
> [設定] からデベロッパーツール コンソールを使用して、接続を作成することもできます。[接続を作成する] を参照してください。

開始する前に:

- GitHub でアカウントを作成しておく必要があります。
- GitHub のコードリポジトリを予め作成しておく必要があります。
- 2019 年 12 月 18 日より前に作成された CodePipeline サービスロールを使用する場合は、`codestar-connections:UseConnection` を AWS CodeStar の接続に使用するアクセス許可の更新が必要になることがあります。手順については、「CodePipeline サービスロールにアクセス許可を追加する」を参照してください。

> **Note**
>
> 接続を作成するには、GitHub 組織の所有者である必要があります。組織のリポジトリでない場合、ユーザーがリポジトリの所有者である必要があります。

トピック

- GitHub (コンソール) への接続を作成する
- GitHub (CLI) への接続を作成する

GitHub (コンソール) への接続を作成する

次の手順を使用して、CodePipeline コンソールを使用して GitHub もしくは GitHub Enterprise Cloud リポジトリに接続アクションを追加します。

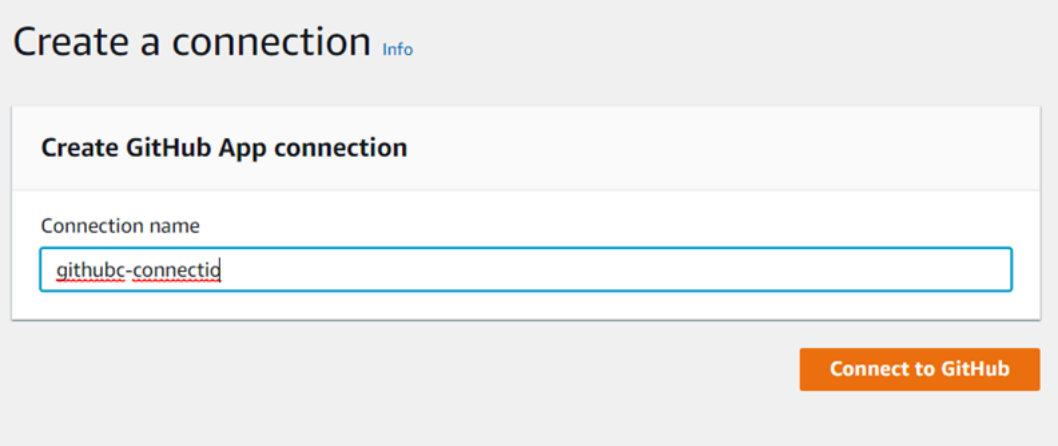
> **Note**
>
> これらのステップでは、[リポジトリアクセス] で特定のリポジトリを選択できます。選択されていないリポジトリは、CodePipeline からアクセスしたり表示したりできなくなります。

ステップ 1 : パイプラインを作成または修正するには

1. CodePipeline コンソールにサインインします。
2. 次のいずれかを選択します 。
   - パイプラインの作成を選択します。[パイプラインを作成する] の手順に従い最初の画面を完了し、[次] を選択します。[送信元] ページの [ソースプロバイダー]で、[GitHub (GitHub アプリ経由)] を選択します。
   - 既存のパイプラインを編集することを選択します。[Edit]、[Edit Stage] の順に選択します。ソースアクションを追加または編集するかを選択します。[アクションを編集する] ページで、[アクション名] に自分のアクション名を入力します。[アクションプロバイダー] で、[GitHub (GitHub アプリ経由)] を選択します。
3. 次のいずれかを行います:
   - [接続] でプロバイダへの接続をまだ作成していない場合は、[GitHub への接続] を選択します。ステップ 2: GitHub への接続を作成する手順に進みます。
   - [接続] でプロバイダへの接続を既に作成している場合は、その接続を選択します。ステップ 3 : 接続のソースアクションを保存するに進みます。

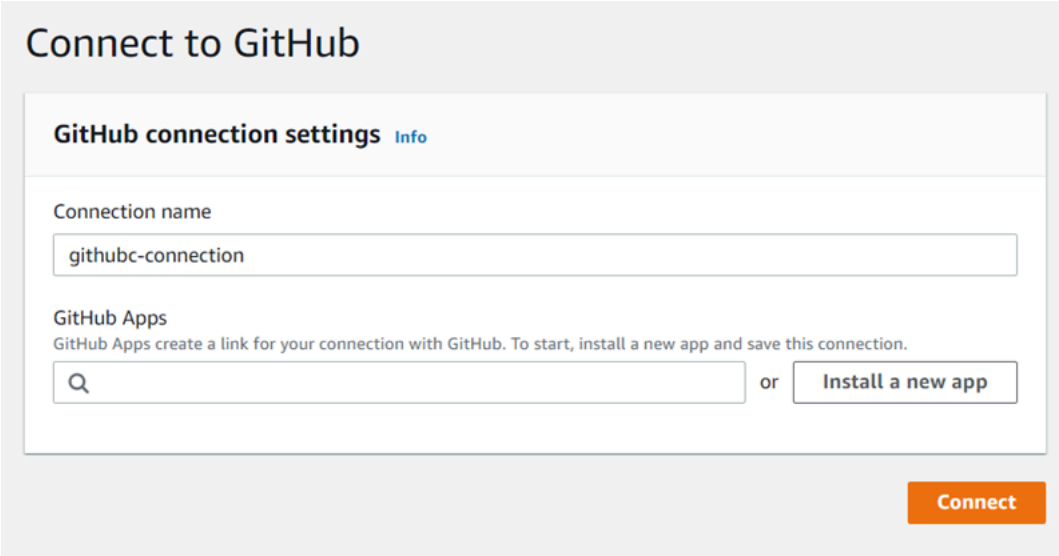
ステップ 2 : GitHub への接続を作成する

接続の作成を選択した後、[GitHub に接続する] ページが表示されます。



GitHub への接続を作成するには

1. [GitHub connection settings] で、[Connection name] に接続名が表示されます。[Connect to GitHub] (GitHub に接続) を選択します。アクセス要求のページが表示されます。
2. GitHub の AWS コネクタの承認を選択します。接続ページには [GitHub Apps] フィールドが表示されます。



3. [GitHub Apps] (Bitbucket アプリ) で、アプリのインストールを選択するか、[Install a new app] (新しいアプリをインストールする) を選択してアプリを作成します。

   特定のプロバイダーへのすべての接続に対してアプリを 1 つインストールします。 AWS Connector for GitHub アプリを既にインストールしている場合は、それを選択してこのステップをスキップします。

> **Note**
>
> ユーザーアクセストークンを作成する場合は、 Connector for GitHub AWS アプリが既にインストールされていることを確認し、アプリのインストールフィールドを空のままにします。 AWS CodeConnections は接続にユーザーアクセストークンを使用します。

4. Install AWS Connector for GitHub ページで、アプリをインストールするアカウントを選択します。

> **Note**
>
> アプリは、GitHub アカウントごとに 1 回だけインストールします。アプリをインストール済みである場合は、Configure (設定) をクリックしてアプリのインストールの変更ページに進むか、戻るボタンでコンソールに戻ることができます。

5. 「Install AWS Connector for GitHub」ページで、デフォルトのままにして、「Install」を選択します。
6. [Connect to GitHub] ページで、新規インストールの接続 ID が GitHub Apps に表示されます。接続 を選択します。

ステップ 3: GitHub のソースアクションを保存する

[アクションを編集] ページで次の手順を使用し、ソースアクションを接続情報とともに保存します。

GitHub のソースアクションを保存するには

1. [リポジトリ名] で、サードパーティーのリポジトリの名前を選択します。
2. パイプライントリガーでは、アクションが AWS CodeConnections アクションである場合にトリガーを追加できます。パイプライントリガー設定を構成し、必要に応じてトリガーでフィルタリングするには、「コードプッシュまたはプルリクエストイベントタイプを使用してトリガーを追加する」で詳細を参照してください。
3. [Output artifact format (出力アーティファクトのフォーマット)] で、アーティファクトのフォーマットを選択する必要があります。

   - デフォルトのメソッドを使用して GitHub アクションからの出力アーティファクトを保存するには、[CodePipeline default] を選択します。アクションは、Bitbucket リポジトリからファイ

ルにアクセスし、パイプラインアーティファクトストアの ZIP ファイルにアーティファクトを保存します。

- リポジトリへの URL 参照を含む JSON ファイルを保存して、ダウンストリームのアクションで Git コマンドを直接実行できるようにするには、[Full clone (フルクローン)] を選択します。このオプションは、CodeBuild ダウンストリームアクションでのみ使用できます。

  このオプションを選択した場合は、Bitbucket、GitHub、GitHub Enterprise Server、または GitLab.com に接続するための CodeBuild GitClone アクセス許可を追加します。 で示されるように CodeBuild プロジェクトサービスロールの権限を更新する必要があります。[完全クローン] オプションを使用する方法を示すチュートリアルについては、チュートリアル: CodeCommit パイプラインソースで完全なクローンを使用する を参照してください。

4. ウィザード上で [次へ] または [保存] を [アクションを編集] ページで選択します。

GitHub (CLI) への接続を作成する

AWS Command Line Interface (AWS CLI) を使用して接続を作成できます。

これを行うには、create-connection コマンドを使用します。

> **Important**
>
> AWS CLI または を通じて作成された接続 AWS CloudFormation は、デフォルトで PENDINGステータスです。CLI または との接続を作成したら CloudFormation、コンソールを使用して接続を編集し、ステータスを にしますAVAILABLE。

接続を作成するには

1. ターミナル (Linux/macOS/Unix) またはコマンドプロンプト (Windows) を開きます。を使用して create-connection コマンド AWS CLI を実行し、接続--connection-nameの --provider-typeと を指定します。この例では、サードパーティープロバイダー名は GitHub で、指定された接続名は MyConnection です。

   ```
   aws codestar-connections create-connection --provider-type GitHub --connection-name
    MyConnection
   ```

   成功した場合、このコマンドは次のような接続 ARN 情報を返します。

```
{
    "ConnectionArn": "arn:aws:codestar-connections:us-west-2:account_id:connection/
aEXAMPLE-8aad-4d5d-8878-dfcab0bc441f"
}
```

2. コンソールを使用して接続を完了します。詳細については、「Update a pending connection」を参照してください。
3. パイプラインはデフォルトで、接続ソースリポジトリへのコードのプッシュ時に変更を検出するようになっています。パイプライントリガーを手動リリース用または Git タグ用に設定するには、以下のいずれかを行います。

   - 手動リリースでのみ開始するようにパイプラインのトリガー設定を構成するには、設定に以下の行を追加します。

     ```
     "DetectChanges": "false",
     ```

   - トリガーでフィルタリングするようにパイプライントリガー設定を構成する方法の詳細については、「コードプッシュまたはプルリクエストイベントタイプを使用してトリガーを追加する」を参照してください。例えば、以下では、パイプライン JSON 定義のパイプラインレベルにタグを追加します。この例では、`release-v0` と `release-v1` が包含する Git タグで、`release-v2` が除外する Git タグです。

     ```
     "triggers": [
                {
                    "providerType": "CodeStarSourceConnection",
                    "gitConfiguration": {
                        "sourceActionName": "Source",
                        "push": [
                            {
                                "tags": {
                                    "includes": [
                                        "release-v0", "release-v1"
                                    ],
                                    "excludes": [
                                        "release-v2"
                                    ]
                                }
                            }
                        ]
                    }
     ```

```
            }
        ]
```

GitHub Enterprise Server 接続

接続を使用すると、サードパーティープロバイダーを AWS リソースに関連付ける設定を承認して確立できます。サードパーティーのリポジトリをパイプラインのソースとして関連付けるには、接続を使用します。

> **Note**
>
> アカウントで既存の接続を作成または使用する代わりに、別の AWS アカウント間の共有接続を使用できます。「別のアカウントと共有されている接続を使用する」を参照してください。

> **Note**
>
> この機能は、アジアパシフィック (香港)、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 (ジャカルタ)、アジアパシフィック (メルボルン)、アジアパシフィック (大阪)、アフリカ (ケープタウン)、中東 (バーレーン)、中東 (アラブ首長国連邦)、欧州 (スペイン)、欧州 (チューリッヒ)、イスラエル (テルアビブ)、または AWS GovCloud (米国西部) の各リージョンでは使用できません。利用可能なその他のアクションについては、「CodePipeline との製品とサービスの統合」を参照してください。欧州 (ミラノ) リージョンでのこのアクションに関する考慮事項については、「CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアクションの場合)」の注意を参照してください。

CodePipeline で GitHub Enterprise Cloud のソースアクションを追加するには、次のいずれかを選択できます。

- CodePipeline コンソール [Create pipeline] ウィザードまたは [Edit action] ページを使用し、[GitHub Enterprise Server] プロバイダオプションを選択します。アクションを追加するには GitHub Enterprise Server への接続を作成する (コンソール) を参照してください。このコンソールは、ホストリソースと接続リソースの作成に役立ちます。

- CLI を使用して、次の通りに `CreateSourceConnection` アクションのアクション設定を `GitHubEnterpriseServer` プロバイダに追加し、リソースを作成します。
  - 接続リソースを作成するには、GitHub Enterprise Server (CLI) への接続を作成します。 を参照してCLI でホストリソースと接続リソースを作成します。
  - `CreateSourceConnection` でのアクション設定の例を CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアクションの場合) で使用し、パイプラインを作成する (CLI) で表示されるようにアクションを追加します。

> **Note**
>
> [設定] からデベロッパーツール コンソールを使用して、接続を作成することもできます。[接続を作成する] を参照してください。

開始する前に:

- GitHub Enterprise Server でアカウントを作成し、インフラストラクチャに GitHub Enterprise Server インスタンスをインストールしておく必要があります。

  > **Note**
  >
  > 各 VPC は、一度に 1 つのホスト (GitHub Enterprise Server インスタンス) にのみ関連付けることができます。

- GitHub Enterprise Server のコードリポジトリを予め作成しておく必要があります。

トピック

- GitHub Enterprise Server への接続を作成する (コンソール)
- GitHub Enterprise Server (CLI) への接続を作成します。

GitHub Enterprise Server への接続を作成する (コンソール)

次の手順を使用して、CodePipeline コンソールを使用して GitHub もしくは GitHub Enterprise Cloud リポジトリに接続アクションを追加します。

> **Note**
>
> GitHub Enterprise Server 接続は、GitHub Enterprise Server 接続の作成に使用されたアカウントが所有するリポジトリへのアクセスだけを提供します。

開始する前に:

GitHub Enterprise Server へホスト接続するには、接続のホストリソースを作成する手順を完了している必要があります。[接続のホストを管理する] を参照してください。

ステップ 1 : パイプラインを作成または修正するには

パイプラインを作成または修正するには

1. CodePipeline コンソールにサインインします。
2. 次のいずれかを選択します 。
   - パイプラインの作成を選択します。[パイプラインを作成する] の手順に従い最初の画面を完了し、[次] を選択します。[送信元] ページの [ソースプロバイダー] から、[GitHub Enterprise Server] を選択します。
   - 既存のパイプラインを編集することを選択します。[Edit]、[Edit Stage] の順に選択します。ソースアクションを追加または編集するかを選択します。[アクションの編集] ページで、[Action name (アクション名)] に自分のアクション名を入力します。[Action provider] で、[GitHub Enterprise Server]を選択します。
3. 次のいずれかを行います:
   - [接続] でプロバイダへの接続をまだ作成していない場合は、[GitHub Enterprise server への接続] を選択します。ステップ 2 に進む： GitHub Enterprise Server への接続を作成する
   - [接続] でプロバイダへの接続を既に作成している場合は、その接続を選択します。ステップ 3： 接続のソースアクションを保存するに進みます。

GitHub Enterprise Server への接続を作成する

接続の作成を選択した後、[GitHub Enterprise server に接続する] ページが表示されます。

> **Important**
>
> AWS CodeConnections リリースの既知の問題により、 は GitHub Enterprise Server バージョン 2.22.0 をサポートしていません。接続するには、バージョン 2.22.1 または入手可能な最新のバージョンにアップグレードします。

GitHub Enterprise Server に接続するには

1. [Connection name] (接続名) に、接続の名前を入力します。
2. [URL] に、サーバーのエンドポイントを入力します。

   > **Note**
   >
   > 提供された URL がすでに接続用の GitHub Enterprise Server をセットアップするために使用されている場合は、そのエンドポイント用に以前に作成されたホストリソース ARN を選択するように求められます。

3. Amazon VPC でサーバーを起動し、VPC に接続する場合は、[Use a VPC] (VPC を使用) をクリックして、以下を完了します。

   a. [VPC ID] で、 VPC ID を選択します。Hub Enterprise Server インスタンスがインストールされているインフラストラクチャに VPC を選択するか、VPN または Direct Connect を介してGitHub Enterprise Server インスタンスにアクセスできる VPC を選択します。

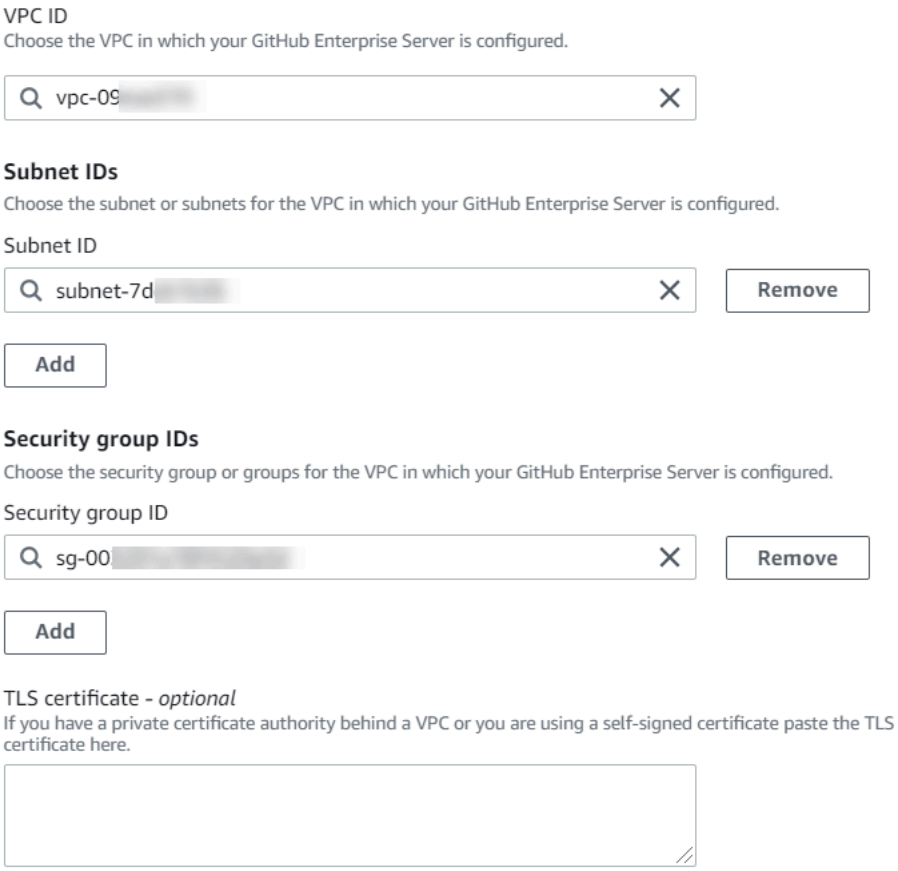
   b. [サブネット ID] で、[Add] を選択します。このフィールドで、ホストに使用するサブネット ID を選択します。最大 10 個のサブネットを選択できます。

      GitHub Enterprise Server インスタンスがインストールされているインフラストラクチャのサブネット、または VPN または Direct Connect を介してインストールされた GitHub Enterprise Server インスタンスにアクセスできるサブネットを選択してください。

   c. [Security group IDs] (セキュリティグループ ID) で、[Add] (追加) を選択します。このフィールドで、ホストに使用するセキュリティグループを選択します。最大 10 個のセキュリティグループを選択できます。

      GitHub Enterprise Server インスタンスがインストールされているインフラストラクチャのセキュリティグループ、または VPN または Direct Connect を介してインストールされた GitHub Enterprise Server インスタンスにアクセスできるセキュリティグループを選択してください。

d. プライベート VPC を設定していて、非公開認証局を使用して TLS 検証を実行するように GitHub Enterprise Server インスタンスを設定している場合は、[TLS証明書] に証明書 ID を入力します。TLS 証明書の値は、証明書のパブリックキーである必要があります。



4. [Connect to GitHub Enterprise Server] (GitHub Enterprise Server に接続する) を選択します。作成された接続は、Pending (保留中) のステータスで表示されます。指定したサーバ情報との接続用に、ホストリソースが作成されます。ホスト名には、URL が使用されます。
5. 保留中の接続の更新を選択します。
6. メッセージが表示されたら、GitHub Enterprise のログインページで、GitHub Enterprise の認証情報でサインインします。
7. [GitHub アプリを作成する] ページで、アプリの名前を選択します。
8. [GitHub の承認] ページで、[<app-name> を承認] を選択します。
9. [アプリインストール] ページに、コネクターアプリをインストールする準備ができたことを示すメッセージが表示されます。複数の組織がある場合は、アプリをインストールする組織を選択するように求められる場合があります。

   アプリをインストールするリポジトリ設定を選択します。[インストール] を選択します。

10. 接続ページには、作成された接続が Available (使用可能) ステータスで表示されます。

ステップ 3: GitHub Enterprise Server のソースアクションを保存する

ウィザードで次の手順を使用するか、[アクションを編集] ページで、ソースアクションを接続情報とともに保存します。

接続でソースアクションを完了して保存するには

1. [リポジトリ名] で、サードパーティーのリポジトリの名前を選択します。
2. パイプライントリガーでは、アクションが AWS CodeConnections アクションである場合にトリガーを追加できます。パイプライントリガー設定を構成し、必要に応じてトリガーでフィルタリングするには、「コードプッシュまたはプルリクエストイベントタイプを使用してトリガーを追加する」で詳細を参照してください。
3. [Output artifact format (出力アーティファクトのフォーマット)] で、アーティファクトのフォーマットを選択する必要があります。

   - デフォルトのメソッドを使用して GitHub Enterprise Server アクションからの出力アーティファクトを保存するには、[CodePipeline default] を選択します。このアクションは、GitHub Enterprise Server リポジトリからファイルにアクセスし、パイプラインアーティファクトストアの ZIP ファイルにアーティファクトを保存します。
   - リポジトリへの URL 参照を含む JSON ファイルを保存して、ダウンストリームのアクションで Git コマンドを直接実行できるようにするには、[Full clone (フルクローン)] を選択します。このオプションは、CodeBuild ダウンストリームアクションでのみ使用できます。
4. ウィザード上で [次へ] または [保存] を [アクションを編集] ページで選択します。

GitHub Enterprise Server (CLI) への接続を作成します。

AWS Command Line Interface (AWS CLI) を使用して接続を作成できます。

これを行うには、create-connection コマンドを使用します。

> **Important**
>
> AWS CLI または を通じて作成された接続 AWS CloudFormation は、デフォルトで `PENDING`ステータスです。CLI または との接続を作成したら CloudFormation、コンソールを使用して接続を編集し、ステータスを にします`AVAILABLE`。

AWS Command Line Interface (AWS CLI) を使用して、インストールされた接続用のホストを作成できます。

> **Note**
>
> ホストは、GitHub Enterprise Server アカウントごとに 1 回だけ作成しまう。特定の GitHub Enterprise Server アカウントへの接続はすべて、同じホストを使用します。

ホストを使用して、サードパーティーのプロバイダがインストールされているインフラストラクチャのエンドポイントを表します。CLI でホストの作成を完了すると、ホストは [Pending] ステータスになります。次に、ホストのステータスが [Available] に移行するよう設定もしくは登録します。ホストが使用可能になったら、接続を作成する手順を完了します。

これを行うには、create-host コマンドを使用します。

> **Important**
>
> を通じて作成されたホスト AWS CLI は、デフォルトで Pendingステータスになります。CLI でホストを作成後、コンソールまたは CLI でホストを設定し、ステータスを Available にします。

ホストを作成するには

1. ターミナル (Linux/macOS/Unix) またはコマンドプロンプト (Windows) を開きます。 AWS CLI を使用して create-host コマンドを実行し、--provider-endpoint接続に --name、--provider-type、 を指定します。この例では、サードパーティープロバイダー名は GitHubEnterpriseServer で、エンドポイントは my-instance.dev です。

   ```
   aws codestar-connections create-host --name MyHost --provider-type
    GitHubEnterpriseServer --provider-endpoint "https://my-instance.dev"
   ```

   成功した場合、このコマンドは次のようなホストの Amazonリソースネーム (ARN) 情報を返します。

   ```
   {
       "HostArn": "arn:aws:codestar-connections:us-west-2:account_id:host/My-
   Host-28aef605"
   ```

```
}
```

この手順の後、ホストのステータスは PENDING になります。

2. コンソールでホストのセットアップを完了し、ホストのステータスを Available に移行します。

GitHub Enterprise Server への接続を作成するには

1. ターミナル (Linux/macOS/Unix) またはコマンドプロンプト (Windows) を開きます。を使用して create-connection コマンド AWS CLI を実行し、接続--connection-nameの --host-arnとを指定します。

```
aws codestar-connections create-connection --host-arn arn:aws:codestar-
connections:us-west-2:account_id:host/MyHost-234EXAMPLE --connection-name
 MyConnection
```

成功した場合、このコマンドは次のような接続 ARN 情報を返します。

```
{
    "ConnectionArn": "arn:aws:codestar-connections:us-west-2:account_id:connection/
aEXAMPLE-8aad"
}
```

2. コンソールを使用して、保留中の接続を設定します。
3. パイプラインはデフォルトで、接続ソースリポジトリへのコードのプッシュ時に変更を検出するようになっています。パイプライントリガーを手動リリース用または Git タグ用に設定するには、以下のいずれかを行います。

   - 手動リリースでのみ開始するようにパイプラインのトリガー設定を構成するには、設定に以下の行を追加します。

     ```
     "DetectChanges": "false",
     ```

   - トリガーでフィルタリングするようにパイプライントリガー設定を構成する方法の詳細については、「コードプッシュまたはプルリクエストイベントタイプを使用してトリガーを追加する」を参照してください。例えば、以下では、パイプライン JSON 定義のパイプラインレベルにタグを追加します。この例では、release-v0 と release-v1 が包含する Git タグで、release-v2 が除外する Git タグです。

```
"triggers": [
            {
                "providerType": "CodeStarSourceConnection",
                "gitConfiguration": {
                    "sourceActionName": "Source",
                    "push": [
                        {
                            "tags": {
                                "includes": [
                                    "release-v0", "release-v1"
                                ],
                                "excludes": [
                                    "release-v2"
                                ]
                            }
                        }
                    ]
                }
            }
        ]
```

GitLab.com への接続

接続を使用すると、サードパーティープロバイダーを AWS リソースに関連付ける設定を承認して確立できます。サードパーティーのリポジトリをパイプラインのソースとして関連付けるには、接続を使用します。

> **Note**
>
> アカウントで既存の接続を作成または使用する代わりに、別の AWS アカウント間の共有接続を使用できます。「別のアカウントと共有されている接続を使用する」を参照してください。

> **Note**
>
> この機能は、アジアパシフィック (香港)、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 (ジャカルタ)、アジアパシフィック (メルボルン)、アジアパシフィック (大阪)、アフリカ (ケープタウン)、中東 (バーレーン)、中東 (アラブ首長国連邦)、欧州 (ス

ペイン)、欧州 (チューリッヒ)、イスラエル (テルアビブ)、または AWS GovCloud (米国西部) の各リージョンでは使用できません。利用可能なその他のアクションについては、「CodePipeline との製品とサービスの統合」を参照してください。欧州 (ミラノ) リージョンでのこのアクションに関する考慮事項については、「CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアクションの場合)」の注意を参照してください。

CodePipeline で GitLab.com のソースアクションを追加するには、次のいずれかを選択できます。

- CodePipeline コンソールの [パイプラインの作成] ウィザードまたは [アクションの編集] ページを使用して、[GitLab] プロバイダーオプションを選択します。アクションを追加するには GitLab.com への接続を作成する (コンソール) を参照してください。このコンソールは、接続リソースの作成に役立ちます。
- CLI を使用して、次の通りに `CreateSourceConnection` でのアクションのアクション設定を `GitLab` プロバイダに追加します。
  - 接続リソースを作成するには、GitLab.com への接続を作成する (CLI) を参照してCLI で接続リソースを作成します。
  - `CreateSourceConnection` でのアクション設定の例を CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアクションの場合) で使用し、パイプラインを作成する (CLI) で表示されるようにアクションを追加します。

> **Note**
>
> [設定] からデベロッパーツール コンソールを使用して、接続を作成することもできます。[接続を作成する] を参照してください。

> **Note**
>
> この接続のインストールを GitLab.com で承認すると、アカウントにアクセスしてデータを処理するアクセス許可を AWS のサービスに付与したものとみなされます。また、アプリケーションをアンインストールすれば、アクセス許可をいつでも取り消すことができます。

開始する前に:

- GitLab.com でアカウントを作成しておく必要があります。

> **Note**
>
> Connections は、接続の作成と承認に使用されたアカウントで所有するリポジトリへのアクセスだけを提供します。

> **Note**
>
> GitLab で、自分が所有者ロールを持っているリポジトリへの接続を作成すると、その接続を CodePipeline などのリソースを含むリポジトリで使用できます。グループ内のリポジトリでは、グループの所有者である必要はありません。

- パイプラインのソースを指定するには、gitlab.com にリポジトリを作成しておく必要があります。

トピック

- GitLab.com への接続を作成する (コンソール)
- GitLab.com への接続を作成する (CLI)

GitLab.com への接続を作成する (コンソール)

以下のステップを使用して、CodePipeline コンソールで GitLab 内のプロジェクト (リポジトリ) 用に接続アクションを追加します。

パイプラインを作成または修正するには

1. CodePipeline コンソールにサインインします。
2. 次のいずれかを選択します 。
   - パイプラインの作成を選択します。[パイプラインを作成する] の手順に従い最初の画面を完了し、[次] を選択します。[ソース] ページの [ソースプロバイダー] で、[GitLab] を選択します。
   - 既存のパイプラインを編集することを選択します。[Edit]、[Edit Stage] の順に選択します。ソースアクションを追加または編集するかを選択します。[アクションを編集する] ページ

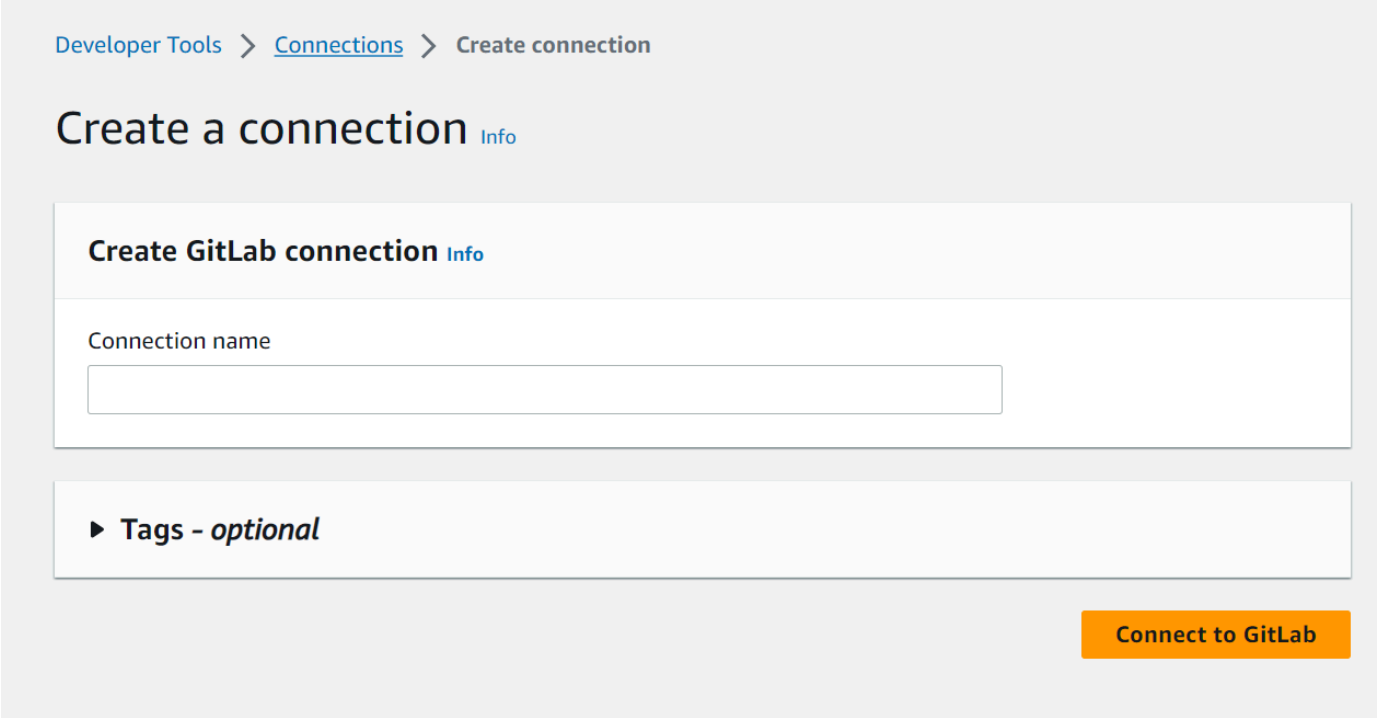
で、[アクション名] に自分のアクション名を入力します。[アクションプロバイダー] で、[GitLab] を選択します。

3. 次のいずれかを行います:

   - [接続] でプロバイダーへの接続をまだ作成していない場合は、[GitLab への接続] を選択します。ステップ 4 に進んで、接続を作成します。
   - [接続] でプロバイダへの接続を既に作成している場合は、その接続を選択します。ステップ 9 に進みます。

> **Note**
>
> GitLab.com 接続が作成される前にポップアップウィンドウを閉じた場合は、ページを更新する必要があります。

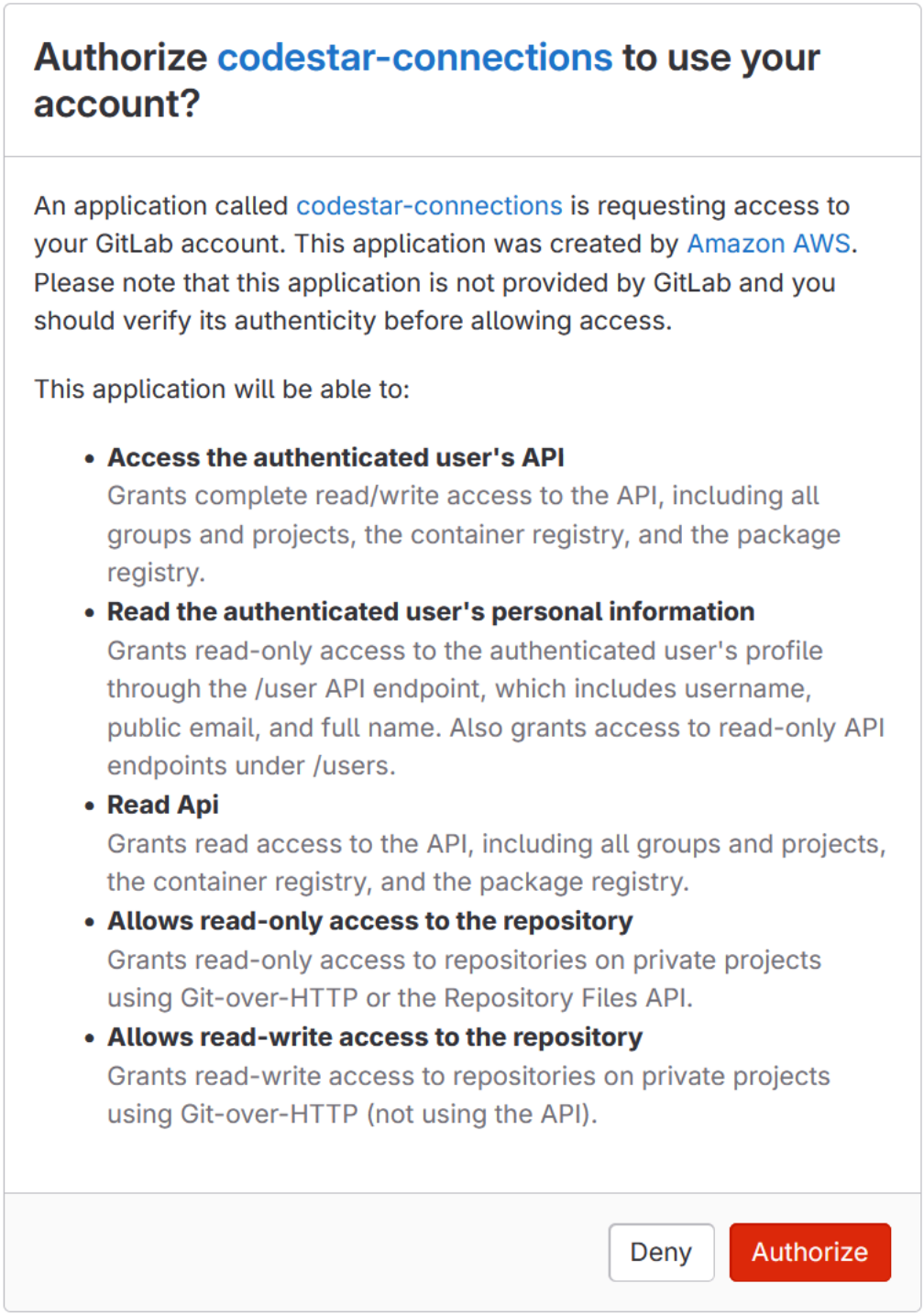
4. GitLab.com リポジトリへの接続を作成するには、[プロバイダーを選択する] で、[GitLab] を選択します。[接続名] に、作成する接続の名前を入力します。 [GitLab に接続] を選択します。



5. GitLab.com のサインインページが表示されたら、認証情報を使用してログインし、[サインイン] を選択します。

6. 初めて接続を承認する場合は、承認ページが表示され、GitLab.com アカウントにアクセスするための接続の承認を求めるメッセージが表示されます。

   [承認] を選択します。



7. ブラウザは接続コンソールページに戻ります。[GitLab 接続を作成] の下で、新しい接続は [接続名] に表示されます。
8. [GitLab に接続] を選択します。

   CodePipeline コンソールに戻ります。

   > **Note**
   >
   > GitLab.com への接続が正常に作成されると、メインウィンドウに成功のバナーが表示されます。
   > 現在のマシンで以前に GitLab にログインしたことがない場合は、ポップアップウィンドウを手動で閉じる必要があります。

9. [リポジトリ名] で、プロジェクトのパスと名前空間を指定して、GitLab 内のプロジェクトの名前を選択します。例えば、グループレベルのリポジトリの場合は、リポジトリ名を `group-name/repository-name` の形式で入力します。パスと名前空間の詳細については、https://docs.gitlab.com/ee/api/projects.html#get-single-project で `path_with_namespace` フィールドを参照してください。GitLab の名前空間の詳細については、https://docs.gitlab.com/ee/user/namespace/ を参照してください。

   > **Note**
   >
   > GitLab 内のグループでは、プロジェクトのパスと名前空間を手動で指定する必要があります。例えば、グループ `mygroup` 内のリポジトリの名前が `myrepo` の場合は、「`mygroup/myrepo`」と入力します。プロジェクトのパスと名前空間は GitLab の URL で見つけることができます。

10. パイプライントリガーでは、アクションが AWS CodeConnections アクションである場合にトリガーを追加できます。パイプライントリガー設定を構成し、必要に応じてトリガーでフィルタリングするには、「コードプッシュまたはプルリクエストイベントタイプを使用してトリガーを追加する」で詳細を参照してください。
11. [ブランチ名] で、パイプラインでソースの変更を検出するブランチを選択します。

> **Note**
>
> ブランチ名が自動的に入力されない場合は、リポジトリへの所有者アクセス権がありません。プロジェクト名が無効であるか、使用している接続がプロジェクト/リポジトリにアクセスできないかのいずれかです。

12. [Output artifact format (出力アーティファクトのフォーマット)] で、アーティファクトのフォーマットを選択する必要があります。

    - デフォルトのメソッドを使用して GitLab.com アクションからの出力アーティファクトを保存するには、[CodePipeline default] を選択します。アクションは、GitLab.com リポジトリからファイルにアクセスし、パイプラインアーティファクトストアの ZIP ファイルにアーティファクトを保存します。
    - リポジトリへの URL 参照を含む JSON ファイルを保存して、ダウンストリームのアクションで Git コマンドを直接実行できるようにするには、[Full clone (フルクローン)] を選択します。このオプションは、CodeBuild ダウンストリームアクションでのみ使用できます。

      このオプションを選択した場合は、Bitbucket、GitHub、GitHub Enterprise Server、または GitLab.com に接続するための CodeBuild GitClone アクセス許可を追加します。 で示されるように CodeBuild プロジェクトサービスロールの権限を更新する必要があります。[完全クローン] オプションを使用する方法を示すチュートリアルについては、チュートリアル: CodeCommit パイプラインソースで完全なクローンを使用する を参照してください。

13. ソースアクションを保存して続行することを選択します。

GitLab.com への接続を作成する (CLI)

AWS Command Line Interface (AWS CLI) を使用して接続を作成できます。

これを行うには、create-connection コマンドを使用します。

> **Important**
>
> AWS CLI または を通じて作成された接続 AWS CloudFormation は、デフォルトで `PENDING`ステータスです。CLI または との接続を作成したら CloudFormation、コンソールを使用して接続を編集し、ステータスを にします`AVAILABLE`。

接続を作成するには

1. ターミナル (Linux/macOS/Unix) またはコマンドプロンプト (Windows) を開きます。を使用して create-connection コマンド AWS CLI を実行し、接続--connection-nameの --provider-typeと を指定します。この例では、サードパーティープロバイダー名は GitLab で、指定された接続名は MyConnection です。

```
aws codestar-connections create-connection --provider-type GitLab --connection-name
 MyConnection
```

成功した場合、このコマンドは次のような接続 ARN 情報を返します。

```
{
    "ConnectionArn": "arn:aws:codestar-connections:us-west-2:account_id:connection/
aEXAMPLE-8aad-4d5d-8878-dfcab0bc441f"
}
```

2. コンソールを使用して接続を完了します。詳細については、「Update a pending connection」を参照してください。
3. パイプラインはデフォルトで、接続ソースリポジトリへのコードのプッシュ時に変更を検出するようになっています。パイプライントリガーを手動リリース用または Git タグ用に設定するには、以下のいずれかを行います。

   - 手動リリースでのみ開始するようにパイプラインのトリガー設定を構成するには、設定に以下の行を追加します。

```
"DetectChanges": "false",
```

   - トリガーでフィルタリングするようにパイプライントリガー設定を構成する方法の詳細については、「コードプッシュまたはプルリクエストイベントタイプを使用してトリガーを追加する」を参照してください。例えば、以下では、パイプライン JSON 定義のパイプラインレベルにタグを追加します。この例では、release-v0 と release-v1 が包含する Git タグで、release-v2 が除外する Git タグです。

```
"triggers": [
            {
                "providerType": "CodeStarSourceConnection",
                "gitConfiguration": {
                    "sourceActionName": "Source",
                    "push": [
```

```
                    {
                        "tags": {
                            "includes": [
                                "release-v0", "release-v1"
                            ],
                            "excludes": [
                                "release-v2"
                            ]
                        }
                    }
                ]
            }
        }
    ]
```

GitLab セルフマネージドの接続

接続を使用すると、サードパーティープロバイダーを AWS リソースに関連付ける設定を承認して確立できます。サードパーティーのリポジトリをパイプラインのソースとして関連付けるには、接続を使用します。

> **Note**
>
> アカウントで既存の接続を作成または使用する代わりに、別の AWS アカウント間の共有接続を使用できます。「別のアカウントと共有されている接続を使用する」を参照してください。

> **Note**
>
> この機能は、アジアパシフィック (香港)、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 (ジャカルタ)、アジアパシフィック (メルボルン)、アジアパシフィック (大阪)、アフリカ (ケープタウン)、中東 (バーレーン)、中東 (アラブ首長国連邦)、欧州 (スペイン)、欧州 (チューリッヒ)、イスラエル (テルアビブ)、または AWS GovCloud (米国西部) の各リージョンでは使用できません。利用可能なその他のアクションについては、「CodePipeline との製品とサービスの統合」を参照してください。欧州 (ミラノ) リージョンでのこのアクションに関する考慮事項については、「CodeStarSourceConnection (Bitbucket

Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアクションの場合)」の注意を参照してください。

CodePipeline で GitLab セルフマネージドソースアクションを追加するには、次のいずれかを選択できます。

- CodePipeline コンソールの [パイプラインの作成] ウィザードまたは [アクションの編集] ページを使用して、[GitLab セルフマネージド] プロバイダーオプションを選択します。アクションを追加するには GitLab セルフマネージドへの接続を作成する (コンソール) を参照してください。このコンソールは、ホストリソースと接続リソースの作成に役立ちます。
- CLI を使用して、次の通りに `CreateSourceConnection` アクションのアクション設定を `GitLabSelfManaged` プロバイダに追加し、リソースを作成します。
  - 接続リソースを作成するには、GitLab セルフマネージドへのホストと接続を作成する (CLI) を参照してCLI でホストリソースと接続リソースを作成します。
  - `CreateSourceConnection` でのアクション設定の例を CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアクションの場合) で使用し、パイプラインを作成する (CLI) で表示されるようにアクションを追加します。

> **Note**
>
> [設定] からデベロッパーツール コンソールを使用して、接続を作成することもできます。[接続を作成する] を参照してください。

開始する前に:

- GitLab でアカウントを作成済みで、セルフマネージドインストールの GitLab Enterprise Edition または GitLab Community Edition を持っている必要があります。詳細については、https://docs.gitlab.com/ee/subscriptions/self_managed/ を参照してください。

> **Note**
>
> Connections は、接続の作成と承認に使用されたアカウント用のアクセスだけを提供します。

> **Note**
>
> GitLab で所有者ロールを持つリポジトリへの接続を作成し、その接続を CodePipeline などのリソースで使用できます。グループ内のリポジトリでは、グループの所有者である必要はありません。

- `api` スコープで GitLab 個人用アクセストークン (PAT) を既に作成している必要があります。GitLab インスタンスで管理モードが有効になっている場合、PAT には`admin_mode`スコープも含める必要があります。詳細については、GitLab ウェブサイトの「個人用アクセストークン」と「アクセストークンスコープ」を参照してください。PAT を作成して使用するには、管理者である必要があります。

> **Note**
>
> PAT はホストの認可に使用され、それ以外の方法で保存または接続に使用されることはありません。ホストを設定するには、一時的な PAT を作成し、ホストを設定した後に PAT を削除できます。

- ホストを事前に設定することもできます。VPC の有無にかかわらず、ホストをセットアップできます。VPC 設定の詳細とホストの作成に関する追加情報については、「ホストの作成」を参照してください。

トピック

- GitLab セルフマネージドへの接続を作成する (コンソール)
- GitLab セルフマネージドへのホストと接続を作成する (CLI)

GitLab セルフマネージドへの接続を作成する (コンソール)

以下の手順を使用して、CodePipeline コンソールを使用して GitLab セルフマネージドリポジトリの接続アクションを追加します。

> **Note**
>
> GitLab セルフマネージド接続は、接続の作成に使用された GitLab セルフマネージドアカウントが所有するリポジトリへのアクセスのみを提供します。

開始する前に:

GitLab セルフマネージドへのホスト接続のためには、接続のホストリソースを作成する手順を完了している必要があります。[接続のホストを管理する] を参照してください。

ステップ 1 : パイプラインを作成または修正するには

パイプラインを作成または修正するには

1. CodePipeline コンソールにサインインします。
2. 次のいずれかを選択します 。
   - パイプラインの作成を選択します。[パイプラインを作成する] の手順に従い最初の画面を完了し、[次] を選択します。[ソース] ページの [ソースプロバイダー] で、[GitLab セルフマネージド] を選択します。
   - 既存のパイプラインを編集することを選択します。[Edit]、[Edit Stage] の順に選択します。ソースアクションを追加または編集するかを選択します。[アクションを編集する] ページで、[アクション名] に自分のアクション名を入力します。[アクションプロバイダー] で、[GitLab セルフマネージド] を選択します。
3. 次のいずれかを行います:
   - [接続] で、プロバイダーへの接続をまだ作成していない場合は、[GitLab セルフマネージドへの接続] を選択します。「ステップ 2: GitLab セルフマネージドへの接続を作成する」に進みます。
   - プロバイダーへの接続を作成済みの場合は、[接続] で接続を選択して、「ステップ 3: GitLab セルフマネージドソースアクションを保存する」に進みます。

ステップ 2: GitLab セルフマネージドへの接続を作成する

接続の作成を選択した後、[GitLab セルフマネージドへの接続] ページが表示されます。

GitLab セルフマネージドに接続するには

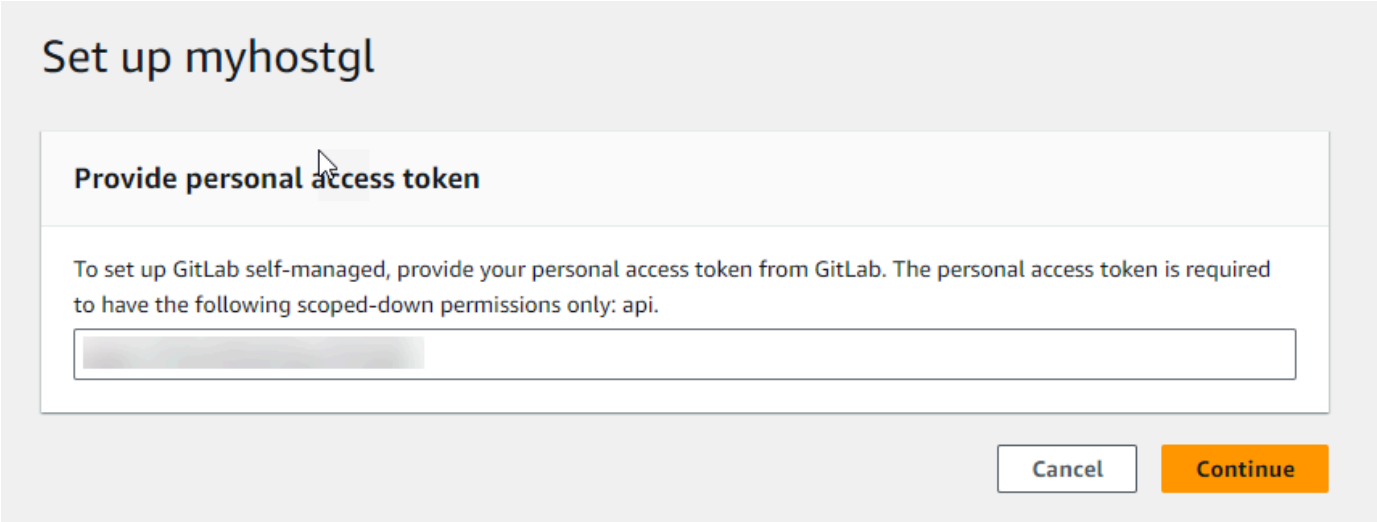
1. [Connection name] (接続名) に、接続の名前を入力します。
2. [URL] に、サーバーのエンドポイントを入力します。

> **Note**
>
> 提供された URL が既に接続用のホストのセットアップに使用されていた場合、そのエンドポイント用に以前に作成されたホストリソース ARN を選択するように求められます。

3. Amazon VPC でサーバーを起動し、VPC で接続する場合は、[VPC を使用] を選択して、VPC の情報を指定します。
4. [GitLab セルフマネージドへの接続] を選択します。作成された接続は、Pending (保留中) のステータスで表示されます。指定したサーバ情報との接続用に、ホストリソースが作成されます。ホスト名には、URL が使用されます。
5. 保留中の接続の更新を選択します。
6. プロバイダーにアクセスし続けることを確認するリダイレクトメッセージを示すページが開いた場合は、[続行] を選択します。プロバイダーの承認を入力します。
7. [***host_name*** のセットアップ] ページが表示されます。「個人用アクセストークンを提供する」で、GitLab PAT に `api`スコープを指定します。GitLab インスタンスで管理モードが有効になっている場合、PAT には`admin_mode`スコープも含める必要があります。詳細については、GitLab ウェブサイトの「アクセストークンスコープ」を参照してください。

> **Note**
>
> PAT を作成して使用できるのは管理者のみです。

[続行] をクリックしてください。



8. 接続ページには、作成された接続が Available (使用可能) ステータスで表示されます。

ステップ 3: GitLab セルフマネージドソースアクションを保存する

ウィザードで次の手順を使用するか、[アクションを編集] ページで、ソースアクションを接続情報とともに保存します。

接続でソースアクションを完了して保存するには

1. [リポジトリ名] で、サードパーティーのリポジトリの名前を選択します。
2. パイプライントリガーでは、アクションが AWS CodeConnections アクションである場合にトリガーを追加できます。パイプライントリガー設定を構成し、必要に応じてトリガーでフィルタリングするには、「コードプッシュまたはプルリクエストイベントタイプを使用してトリガーを追加する」で詳細を参照してください。
3. [Output artifact format (出力アーティファクトのフォーマット)] で、アーティファクトのフォーマットを選択する必要があります。
   - デフォルトのメソッドを使用して GitLab セルフマネージドアクションからの出力アーティファクトを保存するには、[CodePipeline デフォルト] を選択します。アクションは、リポジトリからファイルにアクセスして、パイプラインアーティファクトストアの ZIP ファイルにアーティファクトを保存します。
   - リポジトリへの URL 参照を含む JSON ファイルを保存して、ダウンストリームのアクションで Git コマンドを直接実行できるようにするには、[Full clone (フルクローン)] を選択します。このオプションは、CodeBuild ダウンストリームアクションでのみ使用できます。
4. ウィザード上で [次へ] または [保存] を [アクションを編集] ページで選択します。

GitLab セルフマネージドへのホストと接続を作成する (CLI)

AWS Command Line Interface (AWS CLI) を使用して接続を作成できます。

これを行うには、create-connection コマンドを使用します。

> **Important**
>
> AWS CLI または を通じて作成された接続 AWS CloudFormation は、デフォルトで PENDINGステータスです。CLI または との接続を作成したら CloudFormation、コンソールを使用して接続を編集し、ステータスを にしますAVAILABLE。

AWS Command Line Interface (AWS CLI) を使用して、インストールされた接続用のホストを作成できます。

ホストを使用して、サードパーティーのプロバイダがインストールされているインフラストラクチャのエンドポイントを表します。CLI でホストの作成を完了すると、ホストは [Pending] ステータスになります。次に、ホストのステータスが [Available] に移行するよう設定もしくは登録します。ホストが使用可能になったら、接続を作成する手順を完了します。

これを行うには、create-host コマンドを使用します。

> **Important**
>
> を通じて作成されたホスト AWS CLI は、デフォルトで Pendingステータスになります。CLI でホストを作成後、コンソールまたは CLI でホストを設定し、ステータスを Available にします。

ホストを作成するには

1. ターミナル (Linux/macOS/Unix) またはコマンドプロンプト (Windows) を開きます。 AWS CLI を使用して create-host コマンドを実行し、--provider-endpoint接続に --name、--provider-type、 を指定します。この例では、サードパーティープロバイダー名は GitLabSelfManaged で、エンドポイントは my-instance.dev です。

   ```
   aws codestar-connections create-host --name MyHost --provider-type
    GitLabSelfManaged --provider-endpoint "https://my-instance.dev"
   ```

成功した場合、このコマンドは次のようなホストの Amazonリソースネーム (ARN) 情報を返します。

```
{
    "HostArn": "arn:aws:codestar-connections:us-west-2:account_id:host/My-
Host-28aef605"
}
```

この手順の後、ホストのステータスは PENDING になります。

2. コンソールでホストのセットアップを完了し、ホストのステータスを Available に移行します。

GitLab セルフマネージドへの接続を作成するには

1. ターミナル (Linux/macOS/Unix) またはコマンドプロンプト (Windows) を開きます。を使用して create-connection コマンド AWS CLI を実行し、接続--connection-nameの --host-arnとを指定します。

```
aws codestar-connections create-connection --host-arn arn:aws:codestar-
connections:us-west-2:account_id:host/MyHost-234EXAMPLE --connection-name
 MyConnection
```

成功した場合、このコマンドは次のような接続 ARN 情報を返します。

```
{
    "ConnectionArn": "arn:aws:codestar-connections:us-west-2:account_id:connection/
aEXAMPLE-8aad"
}
```

2. コンソールを使用して、保留中の接続を設定します。
3. パイプラインはデフォルトで、接続ソースリポジトリへのコードのプッシュ時に変更を検出するようになっています。パイプライントリガーを手動リリース用または Git タグ用に設定するには、以下のいずれかを行います。

   - 手動リリースでのみ開始するようにパイプラインのトリガー設定を構成するには、設定に以下の行を追加します。

```
"DetectChanges": "false",
```

- トリガーでフィルタリングするようにパイプライントリガー設定を構成する方法の詳細については、「コードプッシュまたはプルリクエストイベントタイプを使用してトリガーを追加する」を参照してください。例えば、以下では、パイプライン JSON 定義のパイプラインレベルにタグを追加します。この例では、`release-v0` と `release-v1` が包含する Git タグで、`release-v2` が除外する Git タグです。

```
"triggers": [
            {
                "providerType": "CodeStarSourceConnection",
                "gitConfiguration": {
                    "sourceActionName": "Source",
                    "push": [
                        {
                            "tags": {
                                "includes": [
                                    "release-v0", "release-v1"
                                ],
                                "excludes": [
                                    "release-v2"
                                ]
                            }
                        }
                    ]
                }
            }
        ]
```

別のアカウントと共有されている接続を使用する

AWS RAM を使用して、共有接続の作成と管理を行えます。これにより、AWS アカウント間で接続を共有してサードパーティーのリポジトリにアクセスできます。そのため、アカウント間で CodePipeline パイプラインで 1 つの接続を使用でき、ユーザーが各アカウントで個別の接続を管理および管理する必要がなくなります。

CodePipeline で共有接続を使用するには、次の手順を実行します。

- [設定]でデベロッパーツールコンソールを使用して、接続を作成します。[接続を作成する] を参照してください。
- AWS RAM を使用してリソース共有を設定します。「AWS アカウント で接続を共有する」を参照してください。
- CodePipeline コンソールのパイプライン作成ウィザード、または [アクションを編集する] ページを使用して、Bitbucket プロバイダーオプションなどの接続プロバイダーを選択すると、ターゲットアカウントと共有されている接続を選択できます。

トリガーとフィルタリングを使用してパイプラインを自動的に開始する

トリガーを使用すると、特定のブランチやプルリクエストの変更を検出したときなど、特定のイベントタイプやフィルタリングされたイベントタイプに応じて開始するようにパイプラインを設定できます。トリガーは、GitHub、Bitbucket、GitLab など、CodePipeline の `CodeStarSourceConnection` アクションを実行する接続を使用するソースアクションに対して設定できます。接続を使用するソースアクションの詳細については、「を使用してパイプラインにサードパーティーのソースプロバイダーを追加する AWS CodeConnections」を参照してください。

CodeCommit や S3 などのソースアクションは、自動変更検出を使用して、変更があったときにパイプラインを開始します。詳細については、「CodeCommit ソースアクションと EventBridge」を参照してください。

トリガーは、コンソールまたは CLI を使用して指定します。

フィルタータイプは、以下のように指定します。

- フィルターなし

  このトリガー設定は、アクション設定の一環として指定したデフォルトブランチへのあらゆるプッシュに応じてパイプラインを開始します。
- フィルターを指定

  コードプッシュのブランチ名などの特定のフィルターでパイプラインを開始し、正確なコミットを取得するフィルターを追加します。これにより、変更があっても自動的に開始しないようにパイプラインを設定することもできます。
  - プッシュ
    - 有効なフィルターの組み合わせは以下のとおりです。
      - Git タグ

        包含または除外
      - ブランチ

        包含または除外
      - ブランチ + ファイルパス

包含または除外

- プルリクエスト
  - 有効なフィルターの組み合わせは以下のとおりです。
    - ブランチ

      包含または除外
    - ブランチ + ファイルパス

      包含または除外
- 変更を検出しない

  トリガーを追加せず、変更があってもパイプラインを自動的に開始しません。

次の表は、イベントタイプ別の有効なフィルターオプションを示しています。また、アクション設定の自動変更検出がデフォルトで true または false となるトリガー設定も示しています。

トリガーの設定	イベントタイプ	フィルターのオプション	変更を検出する
トリガーを追加 – フィルターなし	なし	なし	true
トリガーを追加 – コードプッシュ時にフィルタリング	プッシュイベント	Git タグ、ブランチ、ファイルパス	false
トリガーを追加 – プルリクエストのフィルタリング	プルリクエスト	ブランチ、ファイルパス	false
トリガーなし - 検出しない	なし	なし	false

> **Note**
>
> このトリガータイプは自動変更検出を (Webhook トリガータイプとして) 使用します。このトリガータイプを使用するソースアクションプロバイダーは、コードプッシュ用に設定された接続 (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、GitLab セルフマネージド) です。

フィールド定義とトリガーのその他のリファレンスについては、を参照してください。

JSON 構造内のフィールド定義のリストについては、「`triggers`」を参照してください。

フィルタリングでは、「構文での glob パターンの使用」で詳述しているように、正規表現パターンを glob 形式でサポートしています。

> **Note**
>
> ファイルパスでトリガーをフィルタリングするパイプラインの場合、ファイルパスフィルターを含むブランチの最初の作成時にパイプラインが開始しないことがあります。詳細については、「ファイルパスによるトリガーフィルタリングを使用する接続を持つパイプラインは、ブランチの作成時に開始しない可能性があります」を参照してください。

トリガーフィルターに関する考慮事項

トリガーを使用するときは、以下の考慮事項が適用されます。

- ソースアクションごとに複数のトリガーを追加することはできません。
- トリガーには複数のフィルタータイプを追加できます。例については、4: 競合する包含と除外の 2 つのプッシュフィルタータイプを持つトリガーを参照してください。
- ブランチフィルターとファイルパスフィルターを含むトリガーの場合、ブランチを初めてプッシュすると、新しく作成したブランチでは変更されたファイルのリストにアクセスできないため、パイプラインは実行されません。
- プルリクエストをマージすると、プッシュ (ブランチフィルター) とプルリクエスト (ブランチフィルター) のトリガー設定が交差して、2 つのパイプライン実行がトリガーされる場合があります。
- プルリクエストイベントでパイプラインをトリガーするフィルターの場合、クローズドプルリクエストイベントタイプでは、接続用のサードパーティーリポジトリプロバイダーがマージイベントに対して別のステータスを持つ場合があります。例えば、Bitbucket では、マージの Git イベントは

プルリクエストクローズイベントではありません。ただし、GitHub では、プルリクエストのマージはクローズイベントです。詳細については、「プロバイダーによるトリガーのプルリクエストイベント」を参照してください。

- パイプライン内の複数のソースアクションが接続を介して同じリポジトリの異なるブランチを参照する場合、パイプラインを確実にトリガーするブランチは 1 つだけです。接続のウェブフックサブスクリプションは、ブランチごとではなく、パイプラインとリポジトリの組み合わせに登録されます。回避策として、ブランチごとに個別のパイプラインを使用します。

プロバイダーによるトリガーのプルリクエストイベント

次の表は、プルリクエストのクローズなど、プロバイダーによるプルリクエストイベントタイプにつながる Git イベントの概要を示しています。

	接続のリポジトリプロバイダー			
トリガーの PR イベント	Bitbucket	GitHub	GHES	GitLab
開始 - このオプションは、ブランチ/ファイルパスのプルリクエストが作成されると、パイプラインをトリガーします。	プルリクエストを作成すると、開始済み Git イベントが発生します。	プルリクエストを作成すると、開始済み Git イベントが発生します。	プルリクエストを作成すると、開始済み Git イベントが発生します。	プルリクエストを作成すると、開始済み Git イベントが発生します。
更新 - このオプションは、ブランチ/ファイルパスのプルリクエストのリビジョンが公開されると、パイプラインをトリガーします。	更新を発行すると、更新済み Git イベントが発生します。	更新を発行すると、更新済み Git イベントが発生します。	更新を発行すると、更新済み Git イベントが発生します。	更新を発行すると、更新済み Git イベントが発生します。

	接続のリポジトリプロバイダー			
トリガーの PR イベント	Bitbucket	GitHub	GHES	GitLab
終了済み - このオプションは、ブランチ/ファイルパスのプルリクエストが終了したときにパイプラインをトリガーします。	Bitbucket でプルリクエストをマージすると、終了済み Git イベントが発生します。重要: マージせずに Bitbucket でプルリクエストを手動で閉じても、終了済み Git イベントは発生しません。	プルリクエストをマージするか手動で閉じると、終了済み Git イベントが発生します。	プルリクエストをマージするか手動で閉じると、終了済み Git イベントが発生します。	プルリクエストをマージするか手動で閉じると、終了済み Git イベントが発生します。

トリガーフィルターの例

プッシュとプルリクエストのイベントタイプのフィルターを含む Git 設定では、指定した複数のフィルターが互いに競合する場合があります。プッシュおよびプルリクエストのイベントの有効なフィルターの組み合わせの例を次に示します。トリガーには、トリガー設定に 2 つのプッシュフィルタータイプなど、複数のフィルタータイプを含めることができます。プッシュリクエストフィルタータイプとプルリクエストフィルタータイプは、それらの間で OR 演算を使用します。つまり、一致するとパイプラインが開始されます。同様に、各フィルタータイプには、filePaths や branches などの複数のフィルターを含めることができます。これらのフィルターは AND 演算を使用します。つまり、完全一致のみがパイプラインを開始します。各フィルタータイプには、包含と除外を含めることができます。除外は包含よりも優先されます。ブランチまたはファイルパスが除外パターンと一致する場合、インクルードパターンとも一致しても、パイプラインはトリガーされません。コミットが複数のファイルを変更すると、各ファイルはフィルタに対して個別に評価されます。変更されたファイルが合格した場合 (インクルードに一致し、インクルードと一致しない場合) 、パイプラインが開始されます。ブランチ名など、包含/除外内の名前は、OR 演算を使用します。次のリストは、Git 設定オブジェクトの各部分の演算をまとめたものです。

JSON 構造内のフィールド定義のリストと、包含と除外の詳細なリファレンスについては、「triggers」を参照してください。

Example 1: ブランチとファイルパスのフィルターを含むフィルタータイプ (AND 演算)

プルリクエストなど単一フィルタータイプの場合、複数のフィルターを組み合わせることができます。これらのフィルターは AND 演算を使用するため、完全に一致する場合にのみパイプラインが開始します。次の例は、2 つの異なるフィルター (filePaths と branches) を持つプッシュイベントタイプの Git 設定を示しています。次の例で、filePaths は branches と AND 演算されます。

```
{
  "filePaths": {
    "includes": ["common/**/*.js"]
  },
  "branches": {
    "includes": ["feature/**"]
  }
}
```

上の Git 設定の場合、次の例は AND 演算の成功により、パイプライン実行を開始するイベントを示しています。つまり、ファイルパス common/app.js はフィルターに含まれ、指定されたブランチ refs/heads/feature/triggers に影響がない場合でも、パイプラインは AND として開始されます。

```
{
  "ref": "refs/heads/feature/triggers",
  ...
  "commits": [
    {
      ...
      "modified": [
        "common/app.js"
      ]
      ...
    }
  ]
}
```

次の例は、ブランチはフィルタリングできるが、ファイルパスはフィルタリングできないため、パイプライン実行を開始しない、上記設定を持つトリガーのイベントを示しています。

```
{
   "ref": "refs/heads/feature/triggers",
  ...
  "commits": [
    {
      ...
      "modified": [
        "src/Main.java"
      ]
      ...
    }
  ]
}
```

Example 2: 包含よりも優先される除外

1 つのフィルター内では、 除外が include よりも優先されます。次の例は、設定オブジェクト内に 1 つのフィルター (branches) を含む Git 設定を示しています。つまり、ブランチが除外パターン (feature-branch例では) に一致する場合、インクルードパターンとも一致してもパイプラインはトリガーされません。main ブランチなど、包含パターンが一致し、除外パターンが一致しない場合、パイプラインがトリガーされます。

たとえば、次の JSON の例の場合:

- コミットを main ブランチにプッシュすると、パイプラインがトリガーされます。
- コミットを feature-branch ブランチにプッシュすると、パイプラインはトリガーされません。

```
{
  "branches": {
      "Includes": [
          "main"
      ],
      "Excludes": [
          "feature-branch"
      ]
   }
```

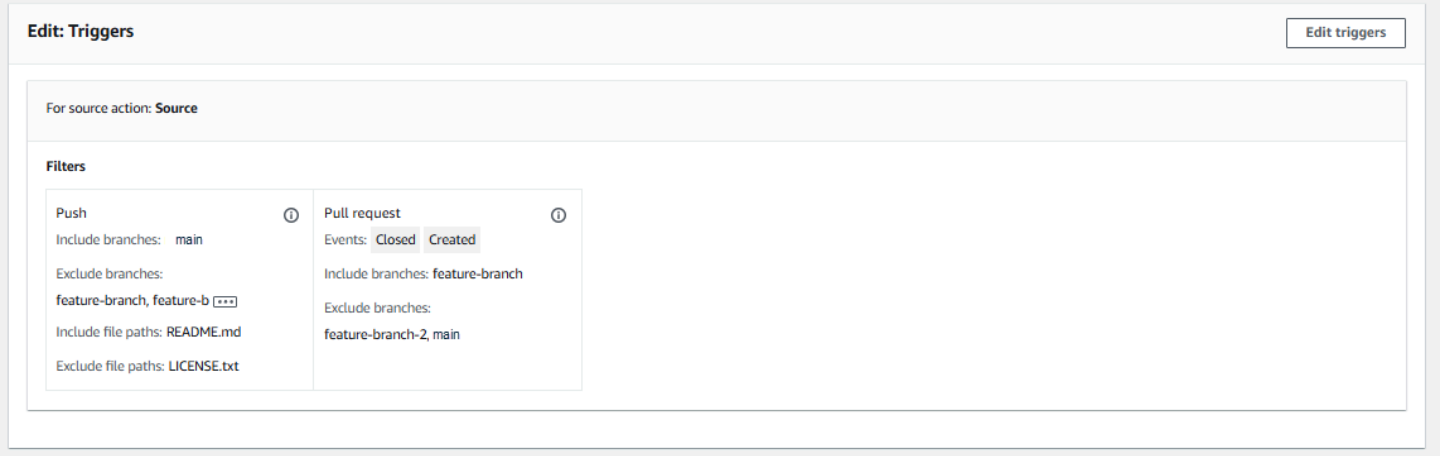
Example 3: プッシュおよびプルリクエストフィルタータイプ (OR オペレーション) 、ファイルパスとブランチのフィルター (AND オペレーション) 、および包含/除外 (除外が優先) を持つトリガー

プッシュイベントタイプとプルリクエストイベントタイプを含むトリガーなどのトリガー設定オブジェクトは、2 つのイベントタイプ間で OR 演算を使用します。次の例は、main ブランチを含むプッシュイベントタイプと、同じブランチ main を除外する 1 つのプルリクエストイベントタイプのトリガー設定を示しています。さらに、プッシュイベントタイプには、1 つのファイルパス LICENSE.txt が除外され、1 つのファイルパス README.MD が含まれます。2 番目のイベントタイプでは、feature-branchブランチ (含まれる) Created で Closedまたは のいずれかのプルリクエストがパイプラインを開始し、 feature-branch-2 または mainブランチ (含まれない) でプルリクエストを作成または閉じるときにパイプラインが開始しません。各イベントタイプ内では、 除外が include よりも優先されます。たとえば、feature-branchブランチのプルリクエストイベント (プルリクエストに含まれる) の場合、プッシュイベントタイプはfeature-branchブランチを除外するため、プッシュはパイプラインをトリガーしません。

次の例では

- README.MD ファイルパス (包含) の main ブランチ (包含) にコミットをプッシュすると、パイプラインがトリガーされます。
- feature-branch ブランチ (除外) でコミットをプッシュすると、パイプラインはトリガーされません。
- 包含ブランチで、README.MD ファイルパス (包含) を編集すると、パイプラインがトリガーされます。
- 含まれているブランチでは、LICENSE.TXTファイルパス (除外) のみを編集してもパイプラインはトリガーされません。ただし、同じコミットも変更 README.MD (含まれている) された場合、各ファイルは個別に評価されるため、パイプラインがトリガーされます。
- feature-branch ブランチでプルリクエストを閉じると、プルリクエストイベントタイプと CLOSED イベントタイプに feature-branch が含まれているため、パイプラインがトリガーされます。

次の画像は、構成を示しています。



以下は、設定の JSON の例です。

```
"triggers": [
            {
                "providerType": "CodeStarSourceConnection",
                "gitConfiguration": {
                    "sourceActionName": "Source",
                    "push": [
                        {
                            "branches": {
                                "includes": [
                                    "main"
                                ],
                                "excludes": [
                                    "feature-branch",
                                    "feature-branch-2"
                                ]
                            },
                            "filePaths": {
                                "includes": [
                                    "README.md"
                                ],
                                "excludes": [
                                    "LICENSE.txt"
                                ]
                            }
                        }
                    ],
                    "pullRequest": [
                        {
                            "events": [
```

```
                            "CLOSED",
                            "OPEN"
                        ],
                        "branches": {
                            "includes": [
                                "feature-branch"
                            ],
                            "excludes": [
                                "feature-branch-2",
                                "main"
                            ]
                        }
                    }
                ]
            }
        }
    ]
},
```

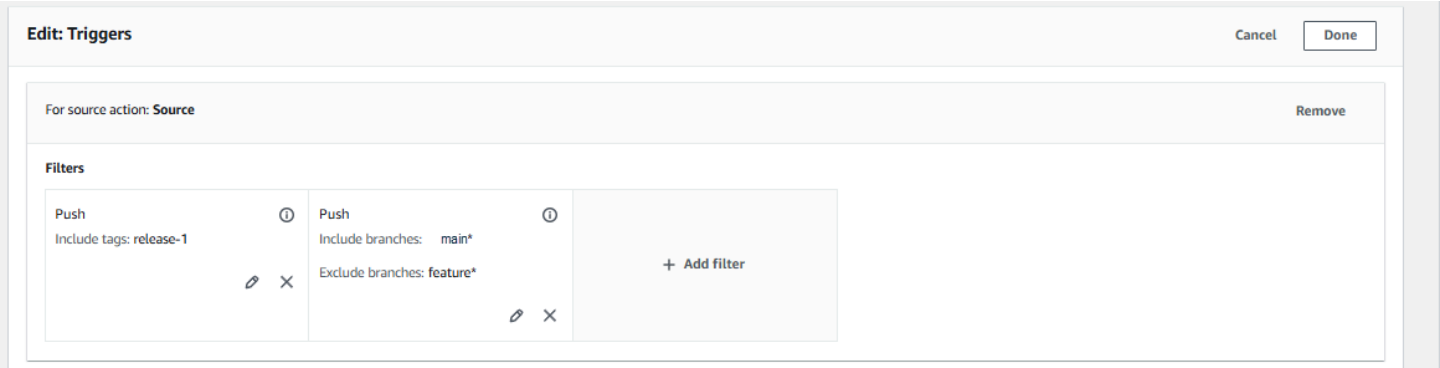
Example 4: 競合する包含と除外の 2 つのプッシュフィルタータイプを持つトリガー

次の図は、タグ release-1 (包含) でフィルタリングするように指定するプッシュフィルタータイプを示しています。2 番目のプッシュフィルタータイプが追加され、ブランチ main をフィルタリングするように指定され(包含)、feature* ブランチへのプッシュを開始しないように指定されます (除外)。

次の例では

- feature-branch ブランチのタグ release-1 (最初のプッシュフィルターに含まれる) (2 番目のプッシュフィルターfeature*の を除く) からリリースをプッシュすると、2 つのプッシュフィルタータイプがそれらの間で OR オペレーションを使用し、最初のプッシュフィルター (タグrelease-1) が一致するため、パイプラインがトリガーされます。
- main ブランチ (2 番目のプッシュフィルターに含まれる) からリリースをプッシュすると、パイプラインは開始されます。

次の [編集] ページの例は、2 つのプッシュフィルタータイプと、それらのフィルタータイプの包含と除外の設定を示しています。



以下は、設定の JSON の例です。

```
"triggers": [
            {
                "providerType": "CodeStarSourceConnection",
                "gitConfiguration": {
                    "sourceActionName": "Source",
                    "push": [
                        {
                            "tags": {
                                "includes": [
                                    "release-1"
                                ]
                            }
                        },
                        {
                            "branches": {
                                "includes": [
                                    "main*"
                                ],
                                "excludes": [
                                    "feature*"
                                ]
                            }
                        }
                    ]
                }
            }
        ]
    },
```

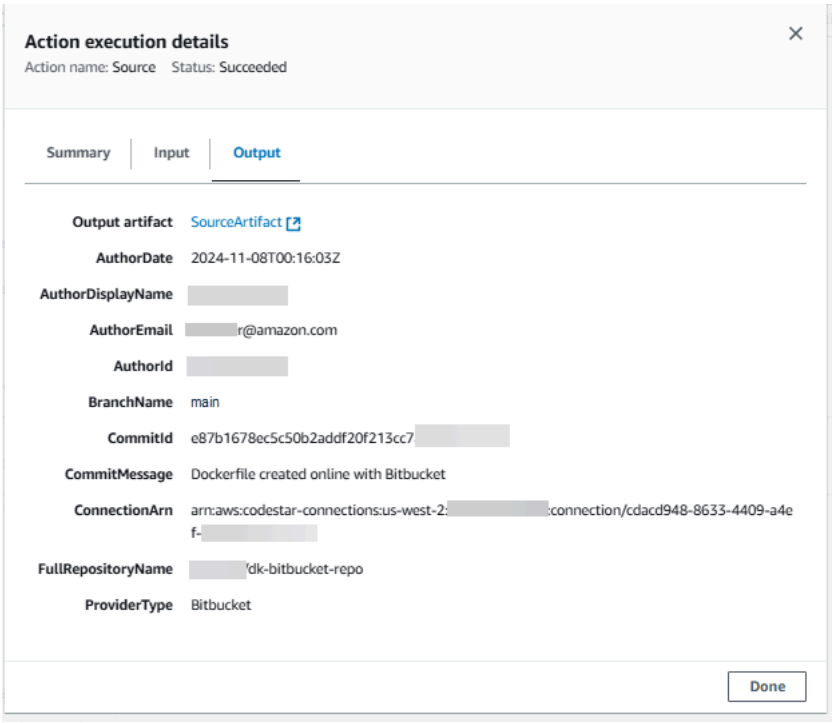
Example 5: デフォルトのアクション設定 BranchName が手動起動に使用される間に設定されたトリガー

アクション設定のデフォルト BranchName フィールドは、パイプラインを手動で開始するときに使用する単一のブランチを定義しますが、フィルター付きのトリガーは、指定した任意のブランチに使用できます。

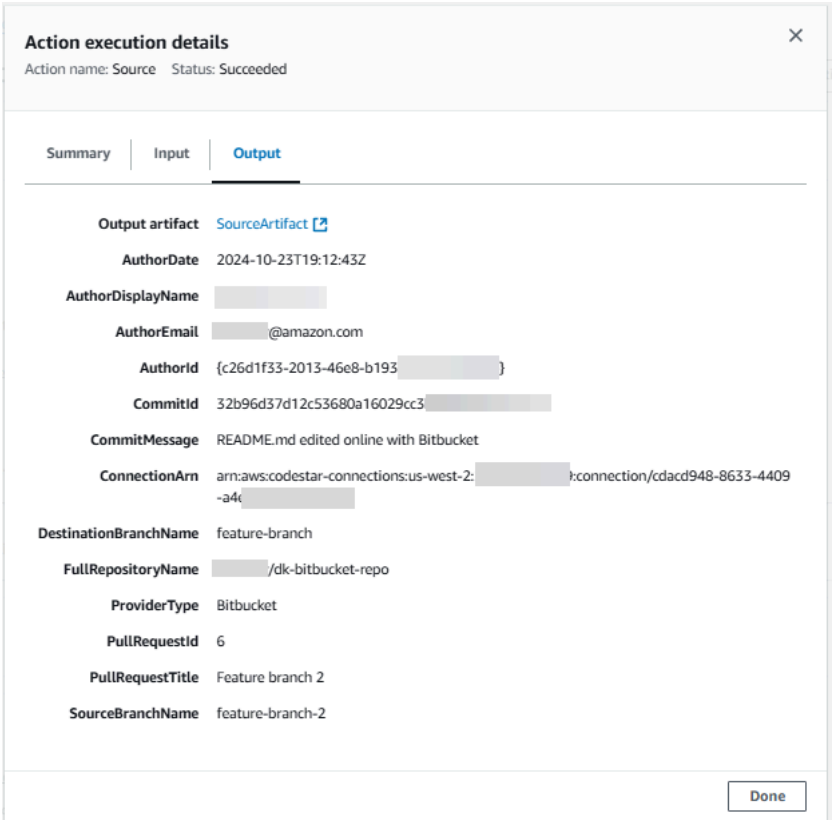
以下は、BranchName フィールドを示すアクション設定の JSON の例です。

```
{
                "name": "Source",
                "actions": [
                    {
                        "name": "Source",
                        "actionTypeId": {
                            "category": "Source",
                            "owner": "AWS",
                            "provider": "CodeStarSourceConnection",
                            "version": "1"
                        },
                        "runOrder": 1,
                        "configuration": {
                            "BranchName": "main",
                            "ConnectionArn": "ARN",
                            "DetectChanges": "false",
                            "FullRepositoryId": "owner-name/my-bitbucket-repo",
                            "OutputArtifactFormat": "CODE_ZIP"
                        },
                        "outputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "inputArtifacts": [],
                        "region": "us-west-2",
                        "namespace": "SourceVariables"
                    }
                ],
```

次のアクション出力の例は、パイプラインを手動で開始したときに使用されたデフォルトのブランチ main を示しています。



次のアクション出力の例は、プルリクエストでフィルタリングされたときにトリガーに使用されたプルリクエストとブランチを示しています。



フィルターを使用せずにコードプッシュにトリガーを追加する

トリガーを使用すると、コードプッシュやプルリクエストなど、特定のイベントタイプで開始するようにパイプラインを設定できます。トリガーは、GitHub、Bitbucket、GitLab など、CodePipeline の CodeStarSourceConnection アクションを実行する接続を使用するソースアクションに対して設定できます。

トリガーを追加する (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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

   AWS アカウントに関連付けられているすべてのパイプラインの名前とステータスが表示されます。
2. [名前] で、編集するパイプラインの名前を選択します。または、パイプライン作成ウィザードで以下の手順を使用します。
3. パイプライン詳細ページで、[編集] を選択します。
4. [編集] ページで、編集するソースアクションを選択します。[トリガーの編集] を選択します。[トリガーを追加] を選択します。
5. [トリガータイプ] で、[フィルターなし] を選択します。

コードプッシュまたはプルリクエストイベントタイプを使用してトリガーを追加する

タグやブランチのプッシュ、特定のファイルパスの変更、特定のブランチに開かれたプルリクエストなど、さまざまな Git イベントに対してパイプライン実行を開始するようにパイプライントリガーのフィルターを設定できます。 AWS CodePipeline コンソール、または の create-pipelineおよび update-pipeline コマンドを使用して AWS CLI 、トリガーフィルターを設定できます。

> **Note**
>
> アクション設定の `BranchName` フィールドは 1 つのブランチを定義しますが、フィルター付きのトリガーは、指定した任意のブランチに使用できます。プッシュリクエストまたはプルリクエストでブランチをフィルタリングするためにトリガーが使用されるパイプラインの場合、パイプラインはアクション設定でデフォルトの `BranchName` フィールドブランチを使用しません。ただし、パイプラインを手動で開始すると、アクション設定の `BranchName`

フィールドのブランチがデフォルトになります。例については、5: デフォルトのアクション設定 BranchName が手動起動に使用される間に設定されたトリガーを参照してください。

以下のトリガータイプに対してフィルターを指定できます。

- プッシュ

  プッシュトリガーは、変更をソースリポジトリにプッシュすると、パイプラインを開始します。実行では、プッシュ先のブランチ (つまり、ターゲットブランチ) からのコミットを使用します。プッシュトリガーは、ブランチ、ファイルパス、または Git タグでフィルタリングできます。
- プルリクエスト

  プルリクエストトリガーは、プルリクエストをソースリポジトリでオープン、更新、またはクローズすると、パイプラインを開始します。実行では、プル元のブランチ (つまり、ソースブランチ) からのコミットを使用します。プルリクエストトリガーは、ブランチとファイルパスでフィルタリングできます。

  プルリクエストでサポートされているイベントタイプは次のとおりです。その他すべてのプルリクエストイベントは無視されます。
  - [オープン済み]
  - 更新
  - クローズ (マージ済み)

> **Note**
>
> 特定のプルリクエストイベントの動作は、プロバイダーによって異なる場合があります。詳細については、「プロバイダーによるトリガーのプルリクエストイベント」を参照してください。

パイプライン定義を使用すると、同じプッシュトリガー設定内でさまざまなフィルターを組み合わせることができます。パイプライン定義の詳細については、「プッシュリクエストイベントタイプとプルリクエストイベントタイプのフィルターを追加する (CLI)」を参照してください。フィールド定義のリストについては、このガイドの「パイプライン構造リファレンス」の「トリガー」を参照してください。

トピック

- プッシュリクエストイベントタイプとプルリクエストイベントタイプのフィルターを追加する (コンソール)
- プッシュリクエストイベントタイプとプルリクエストイベントタイプのフィルターを追加する (CLI)
- プッシュおよびプルリクエストイベントタイプのフィルターを追加する (CloudFormation テンプレート)

プッシュリクエストイベントタイプとプルリクエストイベントタイプのフィルターを追加する (コンソール)

コンソールを使用して、プッシュイベントのフィルターを追加し、ブランチまたはファイルパスを含めるか除外することができます。

フィルターを追加する (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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

   AWS アカウントに関連付けられているすべてのパイプラインの名前とステータスが表示されます。

2. [名前] で、編集するパイプラインの名前を選択します。または、パイプライン作成ウィザードで以下の手順を使用します。
3. パイプライン詳細ページで、[編集] を選択します。
4. [編集] ページで、編集するソースアクションを選択します。[トリガーの編集] を選択します。[フィルターを指定] を選択します。
5. [イベントタイプ] で、以下のオプションから [プッシュ] を選択します。

   - 変更をソースリポジトリにプッシュする場合は、[プッシュ] を選択してパイプラインを開始します。これを選択すると、フィールドでブランチとファイルパスのフィルター、または Git タグのフィルターを指定できます。
   - プルリクエストをソースリポジトリでオープン、更新、またはクローズする場合は、[プルリクエスト] を選択してパイプラインを開始します。これを選択すると、フィールドでターゲットブランチとファイルパスのフィルターを指定できます。

6. [プッシュ] の [フィルタータイプ] で、次のいずれかのオプションを選択します。

- [ブランチ] を選択して、トリガーでモニタリングするソースリポジトリ内のブランチを指定し、ワークフローの実行を開始するタイミングを判断できるようにします。[含める] に、トリガー設定で指定するブランチ名のパターンを glob 形式で入力し、指定したブランチ内の変更に応じてパイプラインが開始するようにします。[除外する] に、トリガー設定で指定するブランチ名の正規表現パターンを glob 形式で入力し、指定したブランチ内の変更を無視してパイプラインが開始しないようにします。詳細については「構文での glob パターンの使用」を参照してください。

  > **Note**
  >
  > [含める] と [除外する] の両方のパターンが同じである場合、デフォルトではパターンを除外します。

  ブランチ名を定義するには、glob パターンを使用できます。例えば、`main*` を使用すると、`main` で始まるすべてのブランチが一致します。詳細については「構文での glob パターンの使用」を参照してください。

  プッシュトリガーの場合は、プッシュ先のブランチ (ターゲットブランチ) を指定します。プルリクエストトリガーでは、プルリクエストを開く送信先ブランチを指定します。

- (オプション) [ファイルパス] で、トリガーのファイルパスを指定します。必要に応じて、[含める] と [除外する] に名前を入力します。

  ファイルパス名を定義するには、glob パターンを使用できます。例えば、`prod*` を使用すると、`prod` で始まるすべてのファイルパスが一致します。詳細については「構文での glob パターンの使用」を参照してください。

- [タグ] を選択して、Git タグで開始するようにパイプラインのトリガー設定を構成します。[含める] に、トリガー設定で指定するタグ名のパターンを glob 形式で入力し、指定したタグのリリース時にパイプラインが開始するようにします。[除外する] に、トリガー設定で指定するタグ名の正規表現パターンを glob 形式で入力し、指定したタグのリリース時にパイプラインが開始しないようにします。[含める] と [除外する] のタグパターンが同じである場合、デフォルトではそのタグパターンは除外されます。

7. [プッシュ] の [フィルタータイプ] で、次のいずれかのオプションを選択します。

   - [ブランチ] を選択して、トリガーでモニタリングするソースリポジトリ内のブランチを指定し、ワークフローの実行を開始するタイミングを判断できるようにします。[含める] に、トリガー設定で指定するブランチ名のパターンを glob 形式で入力し、指定したブランチ内の変更

に応じてパイプラインが開始するようにします。[除外する] に、トリガー設定で指定するブランチ名の正規表現パターンを glob 形式で入力し、指定したブランチ内の変更を無視してパイプラインが開始しないようにします。詳細については「構文での glob パターンの使用」を参照してください。

> **Note**
>
> [含める] と [除外する] の両方のパターンが同じである場合、デフォルトではパターンを除外します。

ブランチ名を定義するには、glob パターンを使用できます。例えば、`main*` を使用すると、`main` で始まるすべてのブランチが一致します。詳細については「構文での glob パターンの使用」を参照してください。

プッシュトリガーの場合は、プッシュ先のブランチ (ターゲットブランチ) を指定します。プルリクエストトリガーでは、プルリクエストを開く送信先ブランチを指定します。

- (オプション) [ファイルパス] で、トリガーのファイルパスを指定します。必要に応じて、[含める] と [除外する] に名前を入力します。

  ファイルパス名を定義するには、glob パターンを使用できます。例えば、`prod*` を使用すると、`prod` で始まるすべてのファイルパスが一致します。詳細については「構文での glob パターンの使用」を参照してください。
- [プルリクエスト] を選択して、指定したプルリクエストイベントで開始するようにパイプライントリガーを設定します。

プッシュリクエストイベントタイプとプルリクエストイベントタイプのフィルターを追加する (CLI)

パイプライン JSON を更新して、トリガーのフィルターを追加できます。

を使用してパイプライン AWS CLI を作成または更新するには、 `create-pipeline` または `update-pipeline` コマンドを使用します。

次の JSON 構造の例では、`create-pipeline` のフィールド定義のリファレンスを提供します。

フィールド定義のリストについては、このガイドの「パイプライン構造リファレンス」の「トリガー」を参照してください。

```
{
    "pipeline": {
        "name": "MyServicePipeline",
        "triggers": [
            {
                "provider": "Connection",
                "gitConfiguration": {
                    "sourceActionName": "ApplicationSource",
                    "push": [
                        {
                            "filePaths": {
                                "includes": [
                                    "projectA/**",
                                    "common/**/*.js"
                                ],
                                "excludes": [
                                    "**/README.md",
                                    "**/LICENSE",
                                    "**/CONTRIBUTING.md"
                                ]
                            },
                            "branches": {
                                "includes": [
                                    "feature/**",
                                    "release/**"
                                ],
                                "excludes": [
                                    "mainline"
                                ]
                            },
                            "tags": {
                                "includes": [
                                    "release-v0", "release-v1"
                                ],
                                "excludes": [
                                    "release-v2"
                                ]
                            }
                        }
                    ],
                    "pullRequest": [
                        {
                            "events": [
```

```
                            "CLOSED"
                        ],
                        "branches": {
                            "includes": [
                                "feature/**",
                                "release/**"
                            ],
                            "excludes": [
                                "mainline"
                            ]
                        },
                        "filePaths": {
                            "includes": [
                                "projectA/**",
                                "common/**/*.js"
                            ],
                            "excludes": [
                                "**/README.md",
                                "**/LICENSE",
                                "**/CONTRIBUTING.md"
                            ]
                        }
                    }
                ]
            }
        }
    ],
    "stages": [
        {
            "name": "Source",
            "actions": [
                {
                    "name": "ApplicationSource",
                    "configuration": {
                        "BranchName": "mainline",
                        "ConnectionArn": "arn:aws:codestar-connections:eu-
central-1:111122223333:connection/fe9ff2e8-ee25-40c9-829e-65f8EXAMPLE",
                        "FullRepositoryId": "monorepo-example",
                        "OutputArtifactFormat": "CODE_ZIP"
                    }
                }
            ]
        }
    ]
```

```
    }
}
```

プッシュおよびプルリクエストイベントタイプのフィルターを追加する (CloudFormation テンプレート)

でパイプラインリソースを更新 CloudFormation して、トリガーフィルタリングを追加できます。

次のテンプレートスニペットの例では、トリガーフィールド定義の YAML リファレンスを提供します。フィールド定義のリストについては、このガイドの「パイプライン構造リファレンス」の「トリガー」を参照してください。

接続ソースとトリガーフィルター設定の完全なテンプレート例については、 CloudFormation 「ユーザーガイド」の「2 つのステージとトリガー設定を持つパイプライン」を参照してください。

```
pipeline:
  name: MyServicePipeline
  executionMode: PARALLEL
  triggers:
    - provider: CodeConnection
      gitConfiguration:
        sourceActionName: ApplicationSource
        push:
          - filePaths:
              includes:
                - projectA/**
                - common/**/*.js
              excludes:
                - '**/README.md'
                - '**/LICENSE'
                - '**/CONTRIBUTING.md'
            branches:
              includes:
                - feature/**
                - release/**
              excludes:
                - mainline
          - tags:
              includes:
                - release-v0
                - release-v1
              excludes:
```

```
                - release-v2
        pullRequest:
          - events:
              - CLOSED
            branches:
              includes:
                - feature/**
                - release/**
              excludes:
                - mainline
            filePaths:
              includes:
                - projectA/**
                - common/**/*.js
              excludes:
                - '**/README.md'
                - '**/LICENSE'
                - '**/CONTRIBUTING.md'
  stages:
    - name: Source
      actions:
        - name: ApplicationSource
          configuration:
            BranchName: mainline
            ConnectionArn: arn:aws:codestar-connections:eu-
central-1:111122223333:connection/fe9ff2e8-ee25-40c9-829e-65f85EXAMPLE
            FullRepositoryId: monorepo-example
            OutputArtifactFormat: CODE_ZIP
```

トリガーを追加して変更検出をオフにする

トリガーを使用すると、コードプッシュやプルリクエストなど、特定のイベントタイプで開始するようにパイプラインを設定できます。トリガーは、GitHub、Bitbucket、GitLab など、CodePipeline の CodeStarSourceConnection アクションを実行する接続を使用するソースアクションに対して設定できます。

トリガーを追加して変更検出をオフにする (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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

AWS アカウントに関連付けられているすべてのパイプラインの名前とステータスが表示されます。

2. [名前] で、編集するパイプラインの名前を選択します。または、パイプライン作成ウィザードで以下の手順を使用します。
3. パイプライン詳細ページで、[編集] を選択します。
4. [編集] ページで、編集するソースアクションを選択します。[トリガーの編集] を選択します。[トリガーを追加] を選択します。
5. [トリガータイプ] で、[変更を検出しない] を選択します。

パイプラインを手動で開始および停止する

AWS CodePipeline コンソールまたは を使用して AWS CLI 、パイプラインを手動で開始および停止できます。パイプラインを自動的に開始する場合は、イベントベースの変更検出を使用する方法、またはトリガー設定を使用する方法があります。

トピック

- CodePipeline でパイプラインを編集する
- CodePipeline でパイプライン実行を停止します。

CodePipeline でパイプラインを編集する

各パイプライン実行はそれぞれ異なるトリガーに基づいて開始できます。各パイプライン実行には、パイプラインの開始方法に応じて、それぞれ異なるタイプのトリガーを設定できます。各実行のトリガータイプはパイプラインの実行履歴に表示されます。トリガータイプは以下のようにソースアクションプロバイダーによって異なります。

> **Note**
>
> ソースアクションごとに複数のトリガーを指定することはできません。

- パイプラインの作成: パイプラインが作成されると、パイプライン実行が自動的に開始されます。これは実行履歴では `CreatePipeline` トリガータイプです。
- 修正されたオブジェクトの変更: このカテゴリは 実行履歴で `PutActionRevision` トリガータイプを表します。
- コードプッシュに対するブランチとコミットの変更検出: このカテゴリは実行履歴で `CloudWatchEvent` トリガータイプを表します。ソースリポジトリ内のソースコミットとブランチに対する変更が検出されると、パイプラインが開始されます。このトリガータイプは自動変更検出を使用します。このトリガータイプを使用するソースアクションプロバイダーは S3 と CodeCommit です。このタイプは、パイプラインを開始するスケジュールにも使用されます。「スケジュールに基づいたパイプラインの開始」を参照してください。
- ソース変更のポーリング: このカテゴリは実行履歴で `PollForSourceChanges` トリガータイプを表します。ポーリングによりソースリポジトリ内のソースコミットとブランチに対する変更が検出されると、パイプラインが開始されます。このトリガータイプは推奨されないため、自動変更検

出を使用するように移行する必要があります。このトリガータイプを使用するソースアクションプロバイダーは S3 と CodeCommit です。

- サードパーティーソースに対する Webhook イベント: このカテゴリは実行履歴で `Webhook` トリガータイプを表します。Webhook イベントによって変更が検出されると、パイプラインが開始されます。このトリガータイプは自動変更検出を使用します。このトリガータイプを使用するソースアクションプロバイダーは、コードプッシュ用に設定された接続 (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、GitLab セルフマネージド) です。
- サードパーティーソースに対する WebhookV2 イベント: このカテゴリは実行履歴で `WebhookV2` トリガータイプを表します。このタイプは、パイプライン定義に含まれるトリガーに基づいて実行されます。指定した Git タグを含むリリースが検出されると、パイプラインが開始されます。Git タグを使用すると、名前などの識別子でコミットをマークし、その重要性を他のリポジトリユーザーに強調できます。また、Git タグを使用してリポジトリの履歴にある特定のコミットを識別することもできます。このトリガータイプは自動変更検出を無効にします。このトリガータイプを使用するソースアクションプロバイダーは、Git タグ用に設定された接続 (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com) です。
- パイプラインの手動開始: このカテゴリは実行履歴で `StartPipelineExecution` トリガータイプを表します。コンソールまたは を使用して AWS CLI 、パイプラインを手動で開始できます。詳細については、「パイプラインを手動で開始する」を参照してください。
- RollbackStage: このカテゴリは実行履歴で `RollbackStage` トリガータイプを表します。コンソールまたは を使用して AWS CLI 、ステージを手動でまたは自動的にロールバックできます。詳細については、「ステージロールバックの設定」を参照してください。

自動変更検出トリガータイプを使用するソースアクションをパイプラインに追加すると、そのアクションは追加のリソースと連携して動作します。各ソースアクションの作成については、変更検出のためのこれらの追加のリソースのため、別のセクションで詳しく説明します。自動変更検出に必要な各ソースプロバイダーと変更検出方法の詳細については、「ソースアクションと変更検出方法」を参照してください。

トピック

- パイプラインを手動で開始する
- スケジュールに基づいたパイプラインの開始
- ソースリビジョンオーバーライドでパイプラインを開始する

パイプラインを手動で開始する

デフォルトでは、パイプラインは作成時に自動で開始され、その後ソースリポジトリ内で変更があるたびに自動的に開始されます。ただし、再度パイプラインを通して、最新のリビジョンを再実行する場合があります。パイプラインを通して、最新のリビジョンを手動で再実行するには、CodePipeline コンソールまたは AWS CLI と start-pipeline-execution コマンドを使用します。

トピック

- パイプラインを手動で開始する (コンソール)
- パイプラインを手動で開始する (CLI)

パイプラインを手動で開始する (コンソール)

パイプラインを手動で開始し、最新のリビジョンをパイプラインにより実行するには

1. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [Name] で、開始するパイプラインの名前を選択します。
3. パイプラインの詳細ページで、[Release change] を選択します。パイプラインがパラメータ (パイプライン変数) を渡すように設定されている場合、[変更のリリース] を選択すると、[変更のリリース] ウィンドウが開きます。[パイプライン変数] で、パイプラインレベルの変数のフィールドに、このパイプライン実行に渡す値を入力します。詳細については、「変数リファレンス」を参照してください。

   これにより、ソースアクションで指定した各ソース場所における最新のリビジョンがパイプラインにより開始されます。

パイプラインを手動で開始する (CLI)

パイプラインを手動で開始し、アーティファクトの最新バージョンをパイプラインにより実行するには

1. ターミナル (Linux、macOS、または Unix) またはコマンドプロンプト (Windows) を開き、AWS CLI を使用して start-pipeline-execution コマンドを実行し、開始するパイプラインの名前を指定します。たとえば、[*MyFirstPipeline*] という名前のパイプラインにより最後の変更を実行するには、以下のようにします。

```
aws codepipeline start-pipeline-execution --name MyFirstPipeline
```

パイプラインレベルの変数が設定されているパイプラインを開始するには、start-pipeline-execution コマンドにオプションの --variables 引数を使用してパイプラインを開始し、実行で使用される変数を追加します。例えば、値が 1 の変数 var1 を追加するには、以下のコマンドを使用します。

```
aws codepipeline start-pipeline-execution --name MyFirstPipeline --variables
 name=var1,value=1
```

2. 成功したかどうかを確認するには、返されたオブジェクトを表示します。このコマンドでは、以下のような 実行 ID が返ります。

```
{
    "pipelineExecutionId": "c53dbd42-This-Is-An-Example"
}
```

> **Note**
>
> パイプラインを開始したら、CodePipeline コンソールで、または get-pipeline-state コマンドを実行して、進行状況をモニタリングできます。詳細については、「パイプラインを表示する (コンソール)」および「パイプラインの詳細と履歴を表示する (CLI)」を参照してください。

スケジュールに基づいたパイプラインの開始

EventBridge でルールを設定して、スケジュールに基づいてパイプラインを開始することができます。

パイプラインを開始するスケジュールの EventBridge ルールを作成する (コンソール)

スケジュールをイベントソースとする EventBridge ルールを作成するには

1. Amazon EventBridge コンソール (https://console.aws.amazon.com/events/) を開きます。
2. ナビゲーションペインで [ルール] を選択します。
3. [ルールの作成] を選択してから、[ルールの詳細] で [スケジュール] を選択します。

4. 一定間隔または式を使用してスケジュールを設定します。詳細については、「ルールのスケジュール式」を参照してください。
5. [ターゲット] で、[CodePipeline] を選択します。
6. このスケジュールのパイプラインを実行するには、パイプライン ARN を入力します。

> **Note**
>
> パイプライン ARN は、コンソールの [設定] に表示されます。「パイプラインの ARN とサービスロール ARN (コンソール) を表示します。」を参照してください。

7. 次のいずれかを選択して、EventBridge ルールに関連付けられたターゲットを呼び出すためのアクセス許可を EventBridge に与える IAM サービスロールを作成または指定します (この場合、ターゲットは CodePipeline)。
   - パイプラインの実行を開始するためのアクセス許可を EventBridge に与えるサービスロールを作成するには、[この特定のリソースに対して新しいロールを作成する] を選択します。
   - パイプラインの実行を開始するためのアクセス許可を EventBridge に付与するサービスロールを指定するには、[既存のロールの使用] を選択します。
8. [詳細の設定] を選択します。
9. [Configure rule details] ページでルールの名前と説明を入力してから、[State] を選択してルールを有効化します。
10. 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。

パイプラインを開始するスケジュールの EventBridge ルールを作成する (CLI)

を使用してルール AWS CLI を作成するには、 put-rule コマンドを呼び出し、以下を指定します。

- 作成中のルールを一意に識別する名前。この名前は、 AWS アカウントに関連付けられた CodePipeline で作成するすべてのパイプラインで一意である必要があります。
- ルールのためのスケジュール式。

スケジュールをイベントソースとする EventBridge ルールを作成するには

1. put-rule コマンドを呼び出し、`--name` と `--schedule-expression` パラメータを含めます。

例:

以下のサンプルコマンドでは、--schedule-expression を使用して、スケジュールに従って EventBridge をフィルタ処理する MyRule2 という名前のルールを作成します。

```
aws events put-rule --schedule-expression 'cron(15 10 ? * 6L 2002-2005)' --name
 MyRule2
```

2. CodePipeline をターゲットとして追加するには、put-targets コマンドを呼び出し、次のパラメータを含めます。

   - --rule パラメータは、put-rule を使用して作成した rule_name で使用されます。
   - --targets パラメータは、ターゲットリストのリスト Id とターゲットパイプラインの ARN で使用されます。

   次のサンプルコマンドでは、MyCodeCommitRepoRule と呼ばれるルールに対して指定し、ターゲット Id は 1 番で構成されています。これは、ルールのターゲットのリストが何であるかを示し、この場合は ターゲット 1 です。このサンプルコマンドでは、パイプラインのサンプルの ARN も指定されます。パイプラインは、リポジトリ内に変更が加えられると開始します。

```
aws events put-targets --rule MyCodeCommitRepoRule --targets
 Id=1,Arn=arn:aws:codepipeline:us-west-2:80398EXAMPLE:TestPipeline
```

3. EventBridge が CodePipeline を使用してルールを呼び出すためのアクセス許可を付与します。詳細については、 デベロッパーガイドの [Amazon EventBridge のリソースベースのポリシーを使用する] を参照してください。

   a. 次のサンプルを使用して、EventBridge にサービスロールの引き受けを許可する信頼ポリシーを作成します。このスクリプトに trustpolicyforEB.json という名前を付けます。

      JSON

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

```
            "Principal": {
                "Service": "events.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

b. 次のコマンドを使用して、Role-for-MyRule ロールを作成し、信頼ポリシーをアタッチします。

```
aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document
 file://trustpolicyforEB.json
```

c. 次のサンプルに示すように、MyFirstPipeline というパイプラインに対して、アクセス権限ポリシー JSON を作成します。アクセス権限ポリシーに permissionspolicyforEB.json と名前を付けます。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:StartPipelineExecution"
            ],
            "Resource": [
                "arn:aws:codepipeline:us-
west-2:111122223333:MyFirstPipeline"
            ]
        }
    ]
}
```

d. 次のコマンドを実行して、作成した Role-for-MyRule ロールに新しい CodePipeline-Permissions-Policy-for-EB アクセス権限ポリシーをアタッチします。

```
aws iam put-role-policy --role-name Role-for-MyRule --policy-name CodePipeline-
Permissions-Policy-For-EB --policy-document file://permissionspolicyforCWE.json
```

ソースリビジョンオーバーライドでパイプラインを開始する

オーバーライドを使用して、パイプライン実行用に指定した特定のソースリビジョン ID でパイプラインを開始できます。例えば、CodeCommit ソースからの特定のコミット ID を処理するパイプラインを開始する場合、パイプラインの開始時にコミット ID をオーバーライドとして追加できます。

> **Note**
>
> 入力変換エントリを使用してソースオーバーライドを作成し、パイプラインイベントの EventBridge で `revisionValue` を使用することもできます。ここで、`revisionValue` はオブジェクトキー、コミット、またはイメージ ID のソースイベント変数から派生します。詳細については、Amazon ECR ソースアクションと EventBridge リソース、イベントに対して有効にしたソースを使用して Amazon S3 ソースアクションに接続する、または CodeCommit ソースアクションと EventBridge の手順に含まれる入力変換エントリのオプション手順を参照してください。

ソースリビジョンには `revisionType` として 4 つタイプがあります。

- `COMMIT_ID`
- `IMAGE_DIGEST`
- `S3_OBJECT_VERSION_ID`
- `S3_OBJECT_KEY`

> **Note**
>
> `COMMIT_ID` タイプと `IMAGE_DIGEST` タイプのソースリビジョンの場合、ソースリビジョン ID は、すべてのブランチをまたいでリポジトリ内のすべてのコンテンツに適用されます。

> **Note**
>
> ソースリビジョンの `S3_OBJECT_VERSION_ID` タイプと `S3_OBJECT_KEY` タイプは、特定の ObjectKey と VersionID を持つソースを上書きするために、個別に使用することも、一緒に使用することもできます。`S3_OBJECT_KEY` の場合、設定パラメータ

`AllowOverrideForS3ObjectKey` を `true` に設定する必要があります。S3 のソース設定パラメータの詳細については、「設定パラメータ」を参照してください。

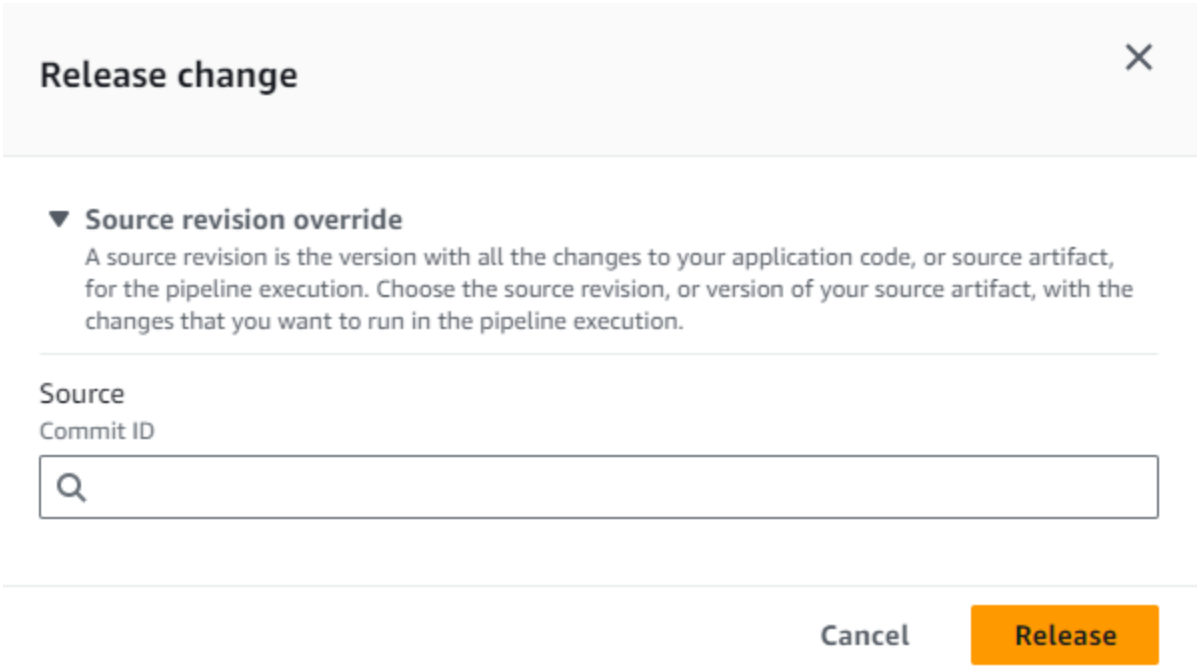
トピック

- ソースリビジョンオーバーライドでパイプラインを開始する (コンソール)
- ソースリビジョンオーバーライドでパイプラインを開始する (CLI)

ソースリビジョンオーバーライドでパイプラインを開始する (コンソール)

パイプラインを手動で開始し、最新のリビジョンをパイプラインにより実行するには

1. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [Name] で、開始するパイプラインの名前を選択します。
3. パイプラインの詳細ページで、[Release change] を選択します。[リリース変更: を選択すると、[リリース変更] ウィンドウが開きます。[ソースリビジョンオーバーライド: では、矢印を選択してフィールドを展開します。[ソース] に、ソースリビジョン ID を入力します。例えば、パイプラインに CodeCommit ソースがある場合は、使用するフィールドからコミット ID を選択します。



ソースリビジョンオーバーライドでパイプラインを開始する (CLI)

パイプラインを手動で開始し、パイプラインを通じてアーティファクトの指定したソースリビジョン ID を実行するには

1. ターミナル (Linux、macOS、または Unix) またはコマンドプロンプト (Windows) を開き、AWS CLI を使用して start-pipeline-execution コマンドを実行し、開始するパイプラインの名前を指定します。また、--source-revisions 引数を使用して、ソースリビジョン ID を指定することもできます。ソースリビジョンは、actionName、revisionType、oおよび revisionValue で構成されます。有効な れヴぃしお n Type の値は `COMMIT_ID | IMAGE_DIGEST | S3_OBJECT_VERSION_ID | S3_OBJECT_KEY` です。

   次の例では、 という名前のパイプラインを介して指定された変更の実行を開始するためcodecommit-pipeline、次のコマンドはソースアクション名として Source、リビジョンタイプとして `COMMIT_ID`、コミット ID として を指定します`78a25c18755ccac3f2a9eec099dEXAMPLE`。

   ```
   aws codepipeline start-pipeline-execution --name codecommit-pipeline --source-
   revisions
    actionName=Source,revisionType=COMMIT_ID,revisionValue=78a25c18755ccac3f2a9eec099dEXAMPLE
    --region us-west-1
   ```

2. 成功したかどうかを確認するには、返されたオブジェクトを表示します。このコマンドでは、以下のような 実行 ID が返ります。

   ```
   {
       "pipelineExecutionId": "c53dbd42-This-Is-An-Example"
   }
   ```

   > **Note**
   >
   > パイプラインを開始したら、CodePipeline コンソールで、または get-pipeline-state コマンドを実行して、進行状況をモニタリングできます。詳細については、「パイプラインを表示する (コンソール)」および「パイプラインの詳細と履歴を表示する (CLI)」を参照してください。

CodePipeline でパイプライン実行を停止します。

パイプライン実行がパイプラインを介して開始されると、パイプライン実行は一度に 1 つのステージに入り、ステージ内のすべてのアクション実行の処理中はステージをロックします。これらの進行中のアクションは、パイプラインの実行が停止されたときに、アクションが完了または中止されるように処理する必要があります。

パイプラインの実行を停止する方法は 2 つあります。

- Stop and wait: 進行中のアクションがすべて完了するまで (つまり、アクション`Failed`のステータスが `Succeeded`または になるまで) 、実行を AWS CodePipeline 停止します。このオプションは、進行中のアクションを保持します。進行中のアクションが完了するまで、実行は `Stopping` 状態になります。その後、実行は `Stopped` 状態になります。アクションが完了すると、ステージがロック解除します。

  [Stop and wait (停止して待機)] を選択し、実行がまだ `Stopping` 状態にある間に待機するのを止める場合は、[中止] を選択できます。

- Stop and abandon: 進行中のアクションが完了するのを待たずに実行を AWS CodePipeline 停止します。進行中のアクションが中止される間、実行は非常に短い時間 `Stopping` 状態になります。実行が停止すると、アクションの実行が `Abandoned` 状態である間、パイプライン実行は `Stopped` 状態になります。ステージのロックが解除されます。

  `Stopped` 状態のパイプライン実行の場合、実行が停止されたステージ内のアクションを再試行できます。

> **Warning**
>
> このオプションを使用すると、タスクが失敗する可能性またはタスクの順序が正しくなくなる可能性があります。

トピック

- パイプライン実行を停止する (コンソール)
- インバウンド実行を停止します (コンソール)。
- パイプライン実行を停止する (CLI)
- インバウンド実行 (CLI) を停止します。

パイプライン実行を停止する (コンソール)

コンソールを使用してパイプラインの実行を停止できます。実行を選択し、パイプラインの実行を停止する方法を選択します。

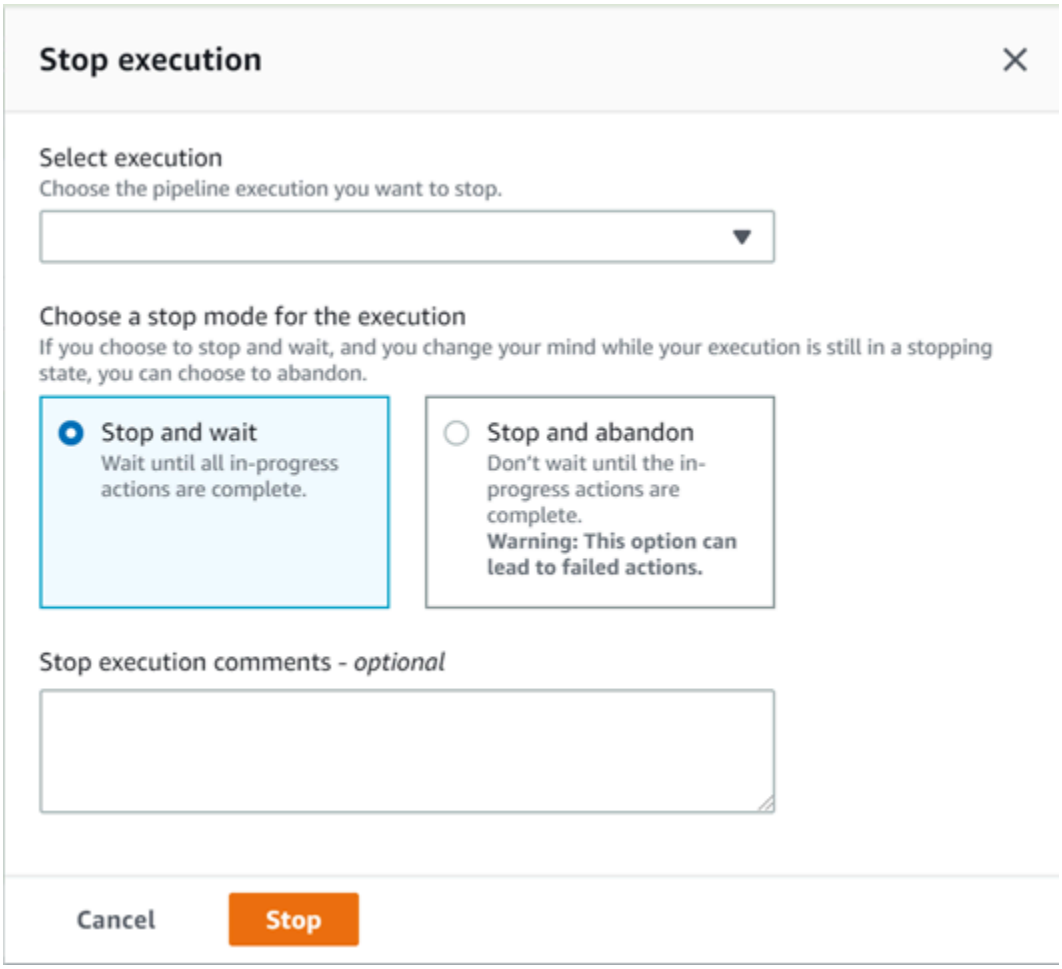
> **Note**
>
> インバウンド実行であるパイプラインの実行を停止することもできます。インバウンド実行を停止する方法については、インバウンド実行を停止します (コンソール)。 を参照してください。

1. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. 次のいずれかを行います。

   > **Note**
   >
   > 実行を停止する前に、ステージの前でトランジションを無効にすることをお勧めします。このようにすれば、実行が停止したためにステージがロック解除するときに、ステージは後続のパイプライン実行を受け付けません。

   - [名前] で、停止する実行を含むパイプラインの名前を選択します。パイプラインの詳細ページで、[Stop execution (実行の停止)] を選択します。
   - [View history (履歴の表示)] を選択します。履歴ページで、[Stop execution (実行を停止)] を選択します。
3. [Stop execution (実行を停止)] ページの [Select execution (実行の選択)] で、停止する実行を選択します。

   > **Note**
   >
   > 実行は、進行中の場合にのみ表示されます。すでに完了している実行は表示されません。



4. [Select an action to apply to execution (実行に適用するアクションの選択)] で、次のいずれかを選択します。

   - 進行中のすべてのアクションが完了するまで実行が停止しないようにするには、[Stop and wait (停止して待機)] を選択します。

     > **Note**
     >
     > 実行がすでに [停止] 状態になっている場合、[Stop and wait (停止して待機)] を選択することはできませんが、[Stop and abandon (停止して中止)] は選択できます。

   - 進行中のアクションが完了するのを待たずに停止するには、[Stop and abandon (停止して中止)] を選択します。

> **Warning**
>
> このオプションを使用すると、タスクが失敗する可能性またはタスクの順序が正しくなくなる可能性があります。

5. （オプション）コメントを入力します。これらのコメントは、実行ステータスとともに、実行の履歴ページに表示されます。
6. [停止] を選択します。

> **Important**
>
> このアクションを元に戻すことはできません。

7. パイプラインの視覚化で実行ステータスを次のように表示します。

   - [Stop and wait (停止して待機)] を選択した場合、実行中のアクションが完了するまで、選択した実行が続行されます。
     - 成功バナーメッセージがコンソールの上部に表示されます。
     - 現在のステージでは、進行中のアクションが `InProgress` 状態で継続されます。アクションが進行中の間、パイプラインの実行は `Stopping` 状態です。

       アクションが完了 (つまり、アクションが失敗または成功) した後、パイプラインの実行は `Stopped` 状態に変更され、アクションは `Failed` または `Succeeded` 状態に変わります。実行の詳細ページでアクションの状態を表示することもできます。実行ステータスは、実行履歴ページまたは実行詳細ページで表示できます。
     - パイプラインの実行が一時的に `Stopping` 状態に変わった後、`Stopped` 状態に変わります。実行ステータスは、実行履歴ページまたは実行詳細ページで表示できます。
   - [Stop and abandon (停止して中止)] を選択した場合、実行は進行中のアクションが完了するまで待機しません。
     - 成功バナーメッセージがコンソールの上部に表示されます。
     - 現在のステージでは、進行中のアクションが `Abandoned` のステータスに変わります 。また、実行の詳細ページでアクションのステータスを表示することもできます。
     - パイプラインの実行が一時的に `Stopping` 状態に変わった後、`Stopped` 状態に変わります。実行ステータスは、実行履歴ページまたは実行詳細ページで表示できます。

パイプライン実行ステータスは、実行履歴ビューと詳細履歴ビューで表示できます。

インバウンド実行を停止します (コンソール)。

コンソールを使用してインバウンドの実行を停止できます。インバウンド実行は、トランジションが無効になっているステージの入力を待っているパイプラインの実行です。トランジションが有効な場合、`InProgress` であるインバウンド実行は引き続きステージを入力し続けます。`Stopped` であるインバウンド実行はステージを入力しません。

> **Note**
>
> インバウンド実行が停止した後は、再試行できません。

インバウンド実行が表示されない場合、無効なステージ移行段階での保留中の実行はありません。

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

   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。
2. インバウンド実行を停止するパイプラインの名前を選択します。次のいずれかを行います。

   - [パイプライン] ビューで、インバウンド実行 ID を選択し、実行を停止することを選択します。
   - パイプラインを選択し、View history を選択します。実行履歴で、インバウンド実行 ID を選択し、実行を停止することを選択します。
3. Stop execution モーダルで、上記のセクションの手順に従って実行 ID を選択し、停止方法を指定します。

   get-pipeline-state のコマンドを実行して、インバウンド実行のステータスを表示します。

パイプライン実行を停止する (CLI)

を使用してパイプライン AWS CLI を手動で停止するには、次のパラメータを指定して stop-pipeline-execution コマンドを使用します。

- 実行 ID (必須)
- コメント (オプション)
- パイプライン名 (必須)
- 中止フラグ (オプション、デフォルトは false)

コマンド形式:

```
aws codepipeline stop-pipeline-execution --pipeline-name Pipeline_Name --pipeline-
execution-id Execution_ID [--abandon | --no-abandon] [--reason STOP_EXECUTION_REASON]
```

1. ターミナル (Linux/macOS/Unix) またはコマンドプロンプト (Windows) を開きます。
2. パイプライン実行を停止するには、次のいずれかを選択します。

   - 進行中のすべてのアクションが完了するまで実行が停止しないようにするには、[Stop and wait (停止して待機)] を選択します。これを行うには、no-abandon パラメータを含めます。パラメータを指定しない場合、コマンドのデフォルトは [Stop and wait (停止して待機)] になります。 AWS CLI を使用して stop-pipeline-execution コマンドを実行し、パイプラインの名前と実行 ID を指定します。たとえば、指定された (停止) と「待機) を用いて、*MyFirstPipeline* と名付けられたパイプラインを停止するには:

     ```
     aws codepipeline stop-pipeline-execution --pipeline-name MyFirstPipeline --
     pipeline-execution-id d-EXAMPLE --no-abandon
     ```

     たとえば、*MyFirstPipeline* という名前のパイプラインを停止し、(停止) と (待機) オプションをデフォルトにし、コメントを含めるようにするには

     ```
     aws codepipeline stop-pipeline-execution --pipeline-name MyFirstPipeline --
     pipeline-execution-id d-EXAMPLE --reason "Stopping execution after the build
      action is done"
     ```

     > **Note**
     >
     > 実行がすでに [停止] 状態になっている場合、[Stop and wait (停止して待機)] を選択することはできません。実行がすでに [停止] 状態になっている場合、[Stop and abandon (停止して中止)] を選択することはできません。

- 進行中のアクションが完了するのを待たずに停止するには、[Stop and abandon (停止して中止)] を選択します。abandon パラメータを指定します。 AWS CLI を使用して stop-pipeline-execution コマンドを実行し、パイプラインの名前と実行 ID を指定します。

  たとえば、*MyFirstPipeline* という名前のパイプラインを停止し、(中止)オプションを指定し、コメントを含めるようにするには

  ```
  aws codepipeline stop-pipeline-execution --pipeline-name MyFirstPipeline --
  pipeline-execution-id d-EXAMPLE --abandon --reason "Stopping execution for a bug
   fix"
  ```

インバウンド実行 (CLI) を停止します。

CLI を使用して、インバウンドの実行を停止できます。インバウンド実行は、トランジションが無効になっているステージの入力を待っているパイプライン実行です。トランジションが有効な場合、`InProgress` であるインバウンド実行は引き続きステージを入力し続けます。`Stopped` であるインバウンド実行はステージを入力しません。

> **Note**
>
> インバウンド実行が停止した後は、再試行できません。

インバウンド実行が表示されない場合、無効なステージ移行段階での保留中の実行はありません。

を使用してインバウンド実行 AWS CLI を手動で停止するには、次のパラメータを指定して stop-pipeline-execution コマンドを使用します。

- インバウンドの実行 ID (必須)
- コメント (オプション)
- パイプライン名 (必須)
- 中止フラグ (オプション、デフォルトは false)

コマンド形式:

```
aws codepipeline stop-pipeline-execution --pipeline-name Pipeline_Name --
pipeline-execution-id Inbound_Execution_ID [--abandon | --no-abandon] [--
reason STOP_EXECUTION_REASON]
```

上記の手順に従って、コマンドを入力し、停止方法を指定します。

get-pipeline-state コマンドを使用して、インバウンド実行のステータスを表示します。

パイプライン実行の履歴を表示し、モードを設定する

パイプラインの進行状況を分析するには、パイプラインとアクションの実行履歴を表示できます。パイプライン実行を処理するためのリリース方法を設定するには、実行モードを設定します。

トピック

- CodePipeline で実行を表示する
- パイプライン実行モードを設定または変更する

CodePipeline で実行を表示する

AWS CodePipeline コンソールまたは を使用して、実行ステータス AWS CLI の表示、実行履歴の表示、失敗したステージまたはアクションの再試行を行うことができます。

トピック

- パイプライン実行の履歴を表示する (コンソール)
- 実行のステータスを表示する (コンソール)
- インバウンドの実行(コンソール)を表示します。
- パイプライン実行ソースのリビジョンを表示する (コンソール)
- アクション実行を表示する (コンソール)
- アクションアーティファクトとアーティファクトストア情報を表示する (コンソール)
- パイプラインの詳細と履歴を表示する (CLI)

パイプライン実行の履歴を表示する (コンソール)

CodePipeline コンソールを使用して、アカウントのすべてのパイプラインのリストを表示できます。パイプラインで最後に実行されたアクション、ステージ間の移行が有効か無効か、失敗したアクションの有無、その他の情報など、各パイプラインの詳細を表示することもできます。履歴が記録されたすべてのパイプライン実行の詳細を示す履歴ページを表示することもできます。

> **Note**
>
> 特定の実行モードから別の実行モードに切り替えると、パイプラインビューと履歴が変わる場合があります。詳細については、「パイプライン実行モードを設定または変更する」を参照してください。

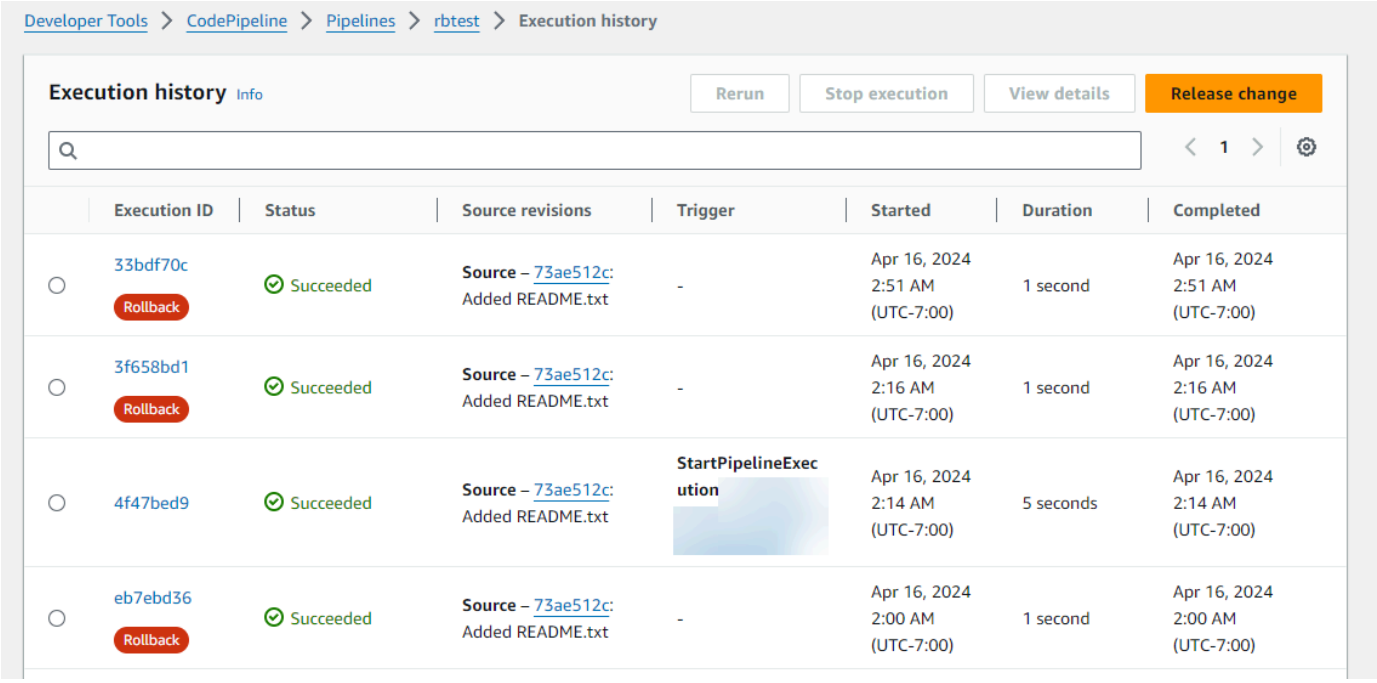
実行履歴は、最大 12 か月間保持されます。

コンソールを使用して、パイプラインの実行履歴を表示できます。履歴には、各実行のステータス、ソースのリビジョン、タイミングの詳細が含まれます。

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

   AWS アカウントに関連付けられているすべてのパイプラインの名前とそのステータスが表示されます。
2. [Name] で、パイプラインの名前を選択します。
3. [View history (履歴の表示)] を選択します。

> **Note**
>
> PARALLEL 実行モードのパイプラインの場合、メインパイプラインビューには、パイプライン構造や進行中の実行は表示されません。PARALLEL 実行モードのパイプラインの場合、パイプライン構造にアクセスするには、表示する実行の ID を実行履歴ページで選択します。左側のナビゲーションで [履歴] を選択し、並列実行の実行 ID を選択して、[可視化] タブでパイプラインを表示します。



4. パイプラインの各実行に関連するステータス、ソースリビジョン、変更の詳細、トリガーが表示されます。ロールバックされたパイプライン実行については、コンソールの詳細画面に実行タイプ Rollback が表示されます。自動ロールバックのトリガーに失敗した実行については、失敗した実行 ID が表示されます。
5. 実行を選択します。詳細ビューには、実行の詳細、[タイムライン] タブ、[視覚化] タブ、[変数] タブが表示されます。パイプラインレベルの変数の値はパイプラインの実行時に解決され、実行ごとに実行履歴で確認できます。

> **Note**
>
> パイプラインアクションからの出力変数は、アクションの実行ごとの履歴の下にある [出力変数] タブで確認できます。

実行のステータスを表示する (コンソール)

パイプラインのステータスは、実行履歴ページの [ステータス] で確認できます。実行 ID リンクを選択し、次にアクションのステータスを表示します。

パイプライン、ステージ、およびアクションの有効な状態は以下のとおりです。

> **Note**
>
> 次のパイプライン状態は、インバウンド実行であるパイプライン実行にも適用されます。受信実行とそのステータスを表示するには、インバウンドの実行(コンソール)を表示します。を参照してください。

パイプラインレベルの状態

パイプラインの状態	説明
InProgress	パイプライン実行が現在実行中です。
停止中	パイプライン実行は、パイプライン実行を [Stop and wait (停止して待機)] するか、[Stop and abandon (停止して中止)] する要求により、停止しています。
停止	停止プロセスが完了し、パイプラインの実行が停止します。
成功	パイプライン実行が正常に完了しました。
優先	このパイプライン実行が次のステージの完了を待機している間に、新しいパイプライン実行が進行し、代わりにパイプラインを継続しました。
失敗	パイプライン実行が正常に完了しませんでした。

ステージレベルの状態

ステージの状態	説明
InProgress	このステージは現在実行中です。
停止中	ステージ実行は、パイプライン実行を [Stop and wait (停止して待機)] するか、[Stop and abandon (停止して中止)] する要求により、停止しています。
停止	停止プロセスが完了し、ステージの実行が停止します。
成功	ステージは正常に完了しました。

ステージの状態	説明
失敗	ステージは正常に完了しませんでした。

アクションレベルの状態

アクションの状態	説明
InProgress	アクションは現在実行中です。
中止	パイプラインの実行を [Stop and abandon (停止して中止)] する要求により、アクションは中止されます。
成功	アクションは正常に完了しました。
失敗	承認アクションでは、失敗の状態は、アクションがレビュー者により拒否されたか、または誤ったアクション設定のために失敗したことを意味します。

インバウンドの実行(コンソール)を表示します。

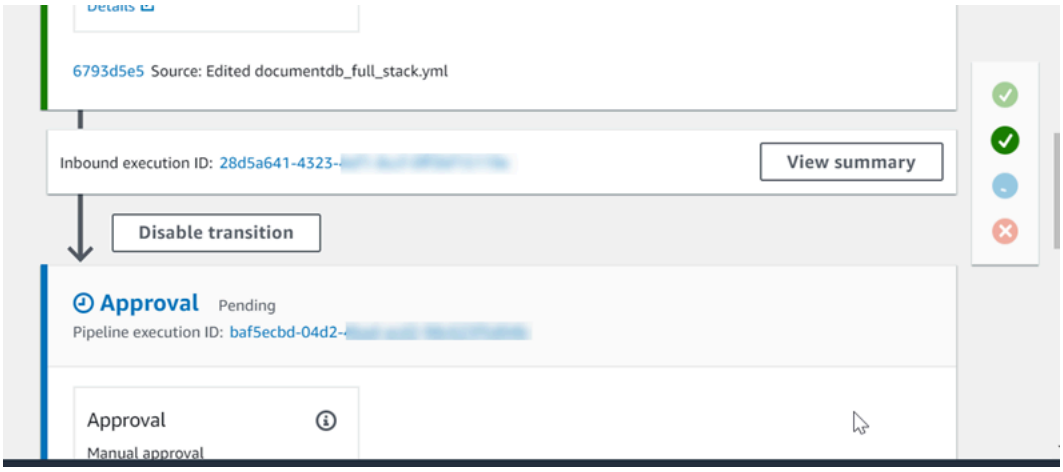
受信実行のステータスと詳細を表示するには、コンソールを使用します。トランジションが有効な場合、またはステージが使用可能になると、`InProgress` であるインバウンド実行が続き、ステージを入力します。`Stopped` のステータスを用いたインバウンド実行は、ステージを入力しません。パイプラインが編集されている場合、インバウンド実行ステータスは `Failed` に変わります。パイプラインを編集すると、進行中のすべての実行は続行されず、実行ステータスは `Failed` に変わります。

インバウンド実行が表示されない場合、無効なステージ移行段階での保留中の実行はありません。

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

   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。
2. インバウンドの実行を表示したいパイプラインの名前を選択し、以下のいずれかの操作を行います：

- [View (表示)] を選択します。パイプラインダイアグラムでは、無効なトランジションの前にある Inbound execution ID フィールドで、インバウンド実行 ID を表示できます。



  View summary を選択し、実行 ID、ソーストリガー、次のステージの名前などの実行の詳細を表示します。

- パイプラインを選択し、View history を選択します。

パイプライン実行ソースのリビジョンを表示する (コンソール)

パイプラインの実行に使用するソースアーティファクト (パイプラインの最初のステージで生成された出力アーティファクト) に関する詳細を表示できます。この詳細には、コミット ID などの識別子、チェックインコメント、および CLI を使用した場合は、パイプラインのビルドアクションのバージョン番号などが含まれます。一部のリビジョンタイプでは、コミットの URL を表示して開くことができます。ソースのリビジョンは、以下で構成されます。

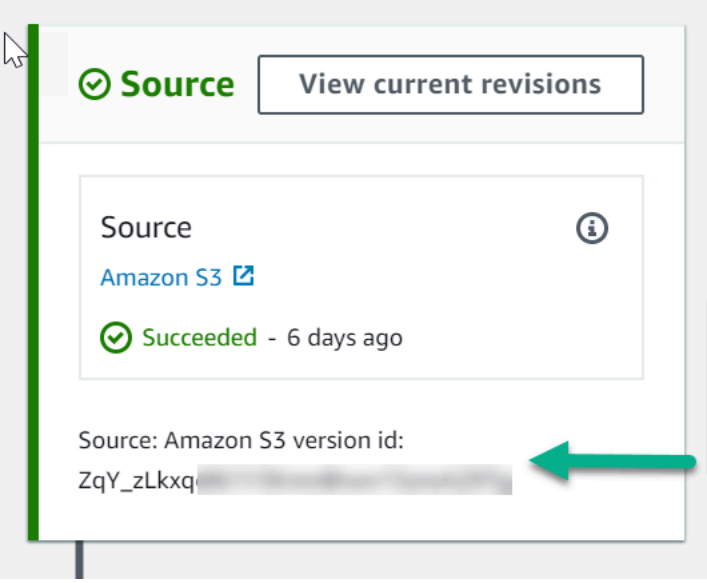
- Summary: アーティファクトの最新のリビジョンに関する概要。GitHub および CodeCommit リポジトリの場合は、コミットメッセージ Amazon S3 バケットまたはアクションの場合は、オブジェクトメタデータに指定された codepipeline-artifact-revision-summary キーのユーザー指定コンテンツ。
- revisionUrl: アーティファクトリビジョンのリビジョン URL (例: 外部リポジトリ URL)。
- revisionId: アーティファクトリビジョンのリビジョン ID。例えば、CodeCommit または GitHub リポジトリ中のソース変更の場合、これはコミット ID です。GitHub またはリポジトリに保存されているアーティファクトの場合、コミット ID はコミットの詳細ページにリンクされています。

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

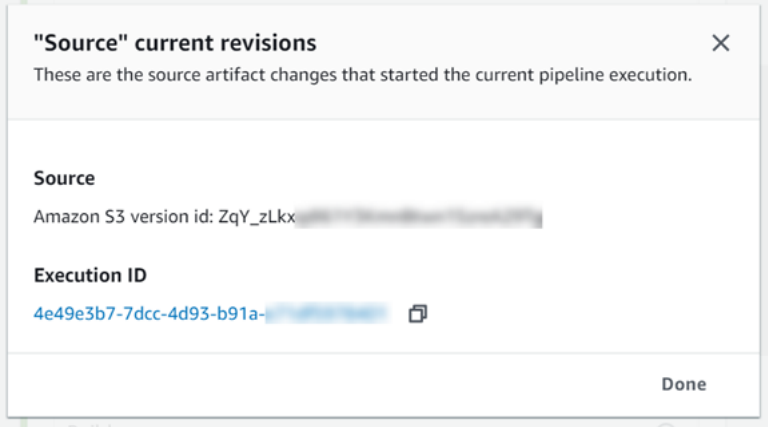
   に関連付けられているすべてのパイプラインの名前 AWS アカウント が表示されます。

2. ソースリビジョンの詳細を表示するパイプラインの名前を選択します。次のいずれかを行います。

   - [View history (履歴の表示)] を選択します。[Source revisions (ソースリビジョン)] に、各実行のソース変更が一覧表示されます。
   - ソースリビジョンの詳細を表示するアクションを見つけ、そのステージの下部にあるリビジョン情報を確認します。



   ソース情報を表示するには、[View current revisions (現行リビジョンを表示)] を選択します。Amazon S3 バケットに格納されているアーティファクトを除いて、この情報詳細ビューのコミット ID などの識別子は、アーティファクトのソース情報ページにリンクされています。



アクション実行を表示する (コンソール)

パイプラインのアクションの詳細として、アクションの実行 ID、入力アーティファクト、出力アーティファクト、ステータスなどを表示できます。アクションの詳細を表示するには、コンソールでパイプラインを選択し、次に実行 ID を選択します。

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

   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。
2. アクションの詳細を表示するパイプラインの名前を選択し、次に [View history (履歴の表示)] を選択します。
3. [Execution ID (実行 ID)] で、アクションの実行の詳細を表示する実行 ID を選択します。
4. [Timeline (タイムライン)] タブで以下の情報を確認できます。

   a. [アクション名] でリンクを選択し、アクションの詳細ページを開きます。このページで、ステータス、ステージ名、アクション名、設定データ、およびアーティファクト情報を確認できます。

   b. [プロバイダー] で、アクションプロバイダーの詳細を表示するリンクを選択します。例えば、上記の例のパイプラインでは、ステージングまたはプロダクションステージで CodeDeploy を選択すると、そのステージ用に設定された CodeDeploy アプリケーションのための CodeDeploy コンソールページが表示されます。

アクションアーティファクトとアーティファクトストア情報を表示する (コンソール)

アクションの入力および出力アーティファクトの詳細を表示できます。そのアクションのアーティファクト情報に関連付けられたリンクを選択することもできます。アーティファクトストアではバージョニングを使用しているため、アクションの実行ごとに一意の入力および出力アーティファクトの場所が割り当てられます。

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

   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。
2. アクションの詳細を表示するパイプラインの名前を選択し、次に [View history (履歴の表示)] を選択します。
3. [Execution ID (実行 ID)] で、アクションの詳細を表示する実行 ID を選択します。
4. [Timeline (タイムライン)] タブの [アクション名] で、アクションの詳細ページを開くリンクを選択します。
5. 詳細ページの [実行] で、アクションの実行のステータスとタイミングを表示します。
6. [設定] タブで、アクションのリソース設定 (CodeBuild のビルドプロジェクト名など) を表示します。
7. [アーティファクト] タブで、[アーティファクトタイプ] と [アーティファクトプロバイダー] のアーティファクト詳細を表示します。[アーティファクト名] のリンクを選択してアーティファクトストアのアーティファクトを表示します。
8. [出力変数] タブで、アクションの実行のためにパイプライン内のアクションから解決された変数を確認できます。

パイプラインの詳細と履歴を表示する (CLI)

パイプラインとパイプラインの実行の詳細を表示するには、以下のコマンドを実行します。

- list-pipelines コマンドを使用して、 に関連付けられているすべてのパイプラインの概要を表示します AWS アカウント。
- get-pipeline コマンドは、1 つのパイプラインの詳細を表示します。
- list-pipeline-executions パイプラインの最新の実行の概要を取得します。

- get-pipeline-execution パイプラインの実行に関する情報 (アーティファクト、パイプラインの実行 ID、パイプラインの名前、バージョン、ステータスなど) を表示します。
- get-pipeline-state コマンドでパイプライン、ステージ、およびアクションのステータスを表示します。
- list-action-executions でパイプラインのアクション実行の詳細を表示します。

トピック

- list-pipeline-executions を使用して実行履歴を表示する(CLI)
- get-pipeline-state でパイプラインの状態を表示する (CLI)
- get-pipeline-state でインバウンド実行のステータスを表示する (CLI)
- get-pipeline-execution でステータスとソースリビジョンを表示する (CLI)
- list-action-executions でアクション実行を表示する (CLI)

`list-pipeline-executions` を使用して実行履歴を表示する(CLI)

パイプラインの実行履歴を表示できます。

- パイプラインの過去の実行の詳細を表示するには、パイプラインの一意の名前を指定して、list-pipeline-executions コマンドを実行します。例えば、*MyFirstPipeline* という名前のパイプラインの現在の状態の詳細を表示するには、以下のように入力します：

  ```
  aws codepipeline list-pipeline-executions --pipeline-name MyFirstPipeline
  ```

  このコマンドは、履歴が記録されたすべてのパイプライン実行に関する概要情報を返します。概要には、開始時刻と終了時刻、期間、ステータスが含まれます。

  ロールバックされたパイプライン実行については、実行タイプ Rollback が表示されます。自動ロールバックのトリガーに失敗した実行については、失敗した実行 ID が表示されます。

  以下の例では、3 つの実行があった *MyFirstPipeline* という名前のパイプラインについて返されたデータを示しています。

  ```
  {
      "pipelineExecutionSummaries": [
          {
              "pipelineExecutionId": "eb7ebd36-353a-4551-90dc-18ca5EXAMPLE",
              "status": "Succeeded",
  ```

```
            "startTime": "2024-04-16T09:00:28.185000+00:00",
            "lastUpdateTime": "2024-04-16T09:00:29.665000+00:00",
            "sourceRevisions": [
                {
                    "actionName": "Source",
                    "revisionId": "revision_ID",
                    "revisionSummary": "Added README.txt",
                    "revisionUrl": "console-URL"
                }
            ],
            "trigger": {
                "triggerType": "StartPipelineExecution",
                "triggerDetail": "trigger_ARN"
            },
            "executionMode": "SUPERSEDED"
        },
        {
            "pipelineExecutionId": "fcd61d8b-4532-4384-9da1-2aca1EXAMPLE",
            "status": "Succeeded",
            "startTime": "2024-04-16T08:58:56.601000+00:00",
            "lastUpdateTime": "2024-04-16T08:59:04.274000+00:00",
            "sourceRevisions": [
                {
                    "actionName": "Source",
                    "revisionId": "revision_ID",
                    "revisionSummary": "Added README.txt",
                    "revisionUrl": "console_URL"
                }
            ],
            "trigger": {
                "triggerType": "StartPipelineExecution",
                "triggerDetail": "trigger_ARN"
            },
            "executionMode": "SUPERSEDED"
        }
```

パイプラインの実行について詳細を表示するには、パイプライン実行の一意の ID を指定して、get-pipeline-execution コマンドを実行します。例えば、前の例の最初の実行の詳細を表示するには、以下のように入力します。

```
aws codepipeline get-pipeline-execution --pipeline-name MyFirstPipeline --pipeline-execution-id 7cf7f7cb-3137-539g-j458-d7eu3EXAMPLE
```

このコマンドは、パイプラインの実行の概要情報 (アーティファクトの詳細、パイプラインの実行 ID、名前、バージョン、パイプラインのステータスなど) を返します。

以下の例では、*MyFirstPipeline* という名前のパイプラインについて返されたデータを示しています。

```
{
    "pipelineExecution": {
        "pipelineExecutionId": "3137f7cb-7cf7-039j-s83l-d7eu3EXAMPLE",
        "pipelineVersion": 2,
        "pipelineName": "MyFirstPipeline",
        "status": "Succeeded",
        "artifactRevisions": [
            {
                "created": 1496380678.648,
                "revisionChangeIdentifier": "1496380258.243",
                "revisionId": "7636d59f3c461cEXAMPLE8417dbc6371",
                "name": "MyApp",
                "revisionSummary": "Updating the application for feature 12-4820"
            }
        ]
    }
}
```

get-pipeline-state でパイプラインの状態を表示する (CLI)

CLI を使って パイプライン、ステージ、およびアクションのステータスを 表示できます。

- パイプラインの現在の状態の詳細を表示するには、パイプラインの一意の名前を指定して、get-pipeline-state コマンドを実行します。例えば、*MyFirstPipeline* という名前のパイプラインの現在の状態の詳細を表示するには、以下のように入力します :

```
aws codepipeline get-pipeline-state --name MyFirstPipeline
```

このコマンドは、パイプラインのすべてのステージの現在のステータスと、それらのステージでのアクションのステータスを返します。

以下の例では、*MyFirstPipeline* という名前の 3 ステージパイプラインについて返されたデータを示しています。ここでは、最初と 2 番目のステージとアクションは成功を示し、3 つ目のステージとアクションは失敗を示しており、2 番目と 3 番目のステージ間の遷移は無効です。

```
{
    "updated": 1427245911.525,
    "created": 1427245911.525,
    "pipelineVersion": 1,
    "pipelineName": "MyFirstPipeline",
    "stageStates": [
        {
            "actionStates": [
                {
                    "actionName": "Source",
                    "entityUrl": "https://console.aws.amazon.com/s3/home?#",
                    "latestExecution": {
                        "status": "Succeeded",
                        "lastStatusChange": 1427298837.768
                    }
                }
            ],
            "stageName": "Source"
        },
        {
            "actionStates": [
                {
                    "actionName": "Deploy-CodeDeploy-Application",
                    "entityUrl": "https://console.aws.amazon.com/codedeploy/home?
#",
                    "latestExecution": {
                        "status": "Succeeded",
                        "lastStatusChange": 1427298939.456,
                        "externalExecutionUrl": "https://console.aws.amazon.com/?
#",
                        "externalExecutionId": ""c53dbd42-This-Is-An-Example"",
                        "summary": "Deployment Succeeded"
                    }
                }
            ],
            "inboundTransitionState": {
                "enabled": true
            },
```

```
            "stageName": "Staging"
        },
        {
            "actionStates": [
                {
                    "actionName": "Deploy-Second-Deployment",
                    "entityUrl": "https://console.aws.amazon.com/codedeploy/home?
#",
                    "latestExecution": {
                        "status": "Failed",
                        "errorDetails": {
                            "message": "Deployment Group is already deploying
 deployment ...",
                            "code": "JobFailed"
                        },
                        "lastStatusChange": 1427246155.648
                    }
                }
            ],
            "inboundTransitionState": {
                "disabledReason": "Disabled while I investigate the failure",
                "enabled": false,
                "lastChangedAt": 1427246517.847,
                "lastChangedBy": "arn:aws:iam::80398EXAMPLE:user/CodePipelineUser"
            },
            "stageName": "Production"
        }
    ]
}
```

get-pipeline-state でインバウンド実行のステータスを表示する (CLI)

インバウンドの実行のステータスを表示するのに、CLI を使用できます。トランジションが有効な場合、またはステージが使用可能な場合、InProgress であるインバウンド実行が続き、ステージを入力します。Stopped のステータスを用いたインバウンド実行は、ステージを入力しません。パイプラインが編集されている場合、インバウンド実行ステータスは Failed に変わります。パイプラインを編集すると、進行中のすべての実行は続行されず、実行ステータスは Failed に変わります。

- パイプラインの現在の状態の詳細を表示するには、パイプラインの一意の名前を指定して、get-pipeline-state コマンドを実行します。例えば、*MyFirstPipeline* という名前のパイプラインの現在の状態の詳細を表示するには、以下のように入力します：

```
aws codepipeline get-pipeline-state --name MyFirstPipeline
```

このコマンドは、パイプラインのすべてのステージの現在のステータスと、それらのステージでのアクションのステータスを返します。出力には、各ステージのパイプライン実行 ID、および、無効なトランジションを持つステージのインバウンド実行 ID があるかどうかも表示されます。

以下の例では、*MyFirstPipeline* という名前の 2 ステージのパイプラインについて返されたデータを示しています。ここでは、第 1 ステージが有効なトランジションと成功したパイプライン実行を示し、Beta と名付けられた第 2 ステージが無効なトランジションとインバウンド実行 ID を示しています。インバウンドの実行は InProgress、Stopped、またはFAILED の状態を持つことができます。

```
{
    "pipelineName": "MyFirstPipeline",
    "pipelineVersion": 2,
    "stageStates": [
        {
            "stageName": "Source",
            "inboundTransitionState": {
                "enabled": true
            },
            "actionStates": [
                {
                    "actionName": "SourceAction",
                    "currentRevision": {
                        "revisionId": "PARcnxX_u0SMRBnKh83pHL09.zPRLLMu"
                    },
                    "latestExecution": {
                        "actionExecutionId": "14c8b311-0e34-4bda-EXAMPLE",
                        "status": "Succeeded",
                        "summary": "Amazon S3 version id: PARcnxX_u0EXAMPLE",
                        "lastStatusChange": 1586273484.137,
                        "externalExecutionId": "PARcnxX_u0EXAMPLE"
                    },
                    "entityUrl": "https://console.aws.amazon.com/s3/home?#"
                }
```

```
            ],
            "latestExecution": {
                "pipelineExecutionId": "27a47e06-6644-42aa-EXAMPLE",
                "status": "Succeeded"
            }
        },
        {
            "stageName": "Beta",
            "inboundExecution": {
                "pipelineExecutionId": "27a47e06-6644-42aa-958a-EXAMPLE",
                "status": "InProgress"
            },
            "inboundTransitionState": {
                "enabled": false,
                "lastChangedBy": "USER_ARN",
                "lastChangedAt": 1586273583.949,
                "disabledReason": "disabled"
            },
                    "currentRevision": {
            "actionStates": [
                {
                    "actionName": "BetaAction",
                    "latestExecution": {
                        "actionExecutionId": "a748f4bf-0b52-4024-98cf-EXAMPLE",
                        "status": "Succeeded",
                        "summary": "Deployment Succeeded",
                        "lastStatusChange": 1586272707.343,
                        "externalExecutionId": "d-KFGF3EXAMPLE",
                        "externalExecutionUrl": "https://us-
west-2.console.aws.amazon.com/codedeploy/home?#/deployments/d-KFGF3WTS2"
                    },
                    "entityUrl": "https://us-west-2.console.aws.amazon.com/
codedeploy/home?#/applications/my-application"
                }
            ],
            "latestExecution": {
                "pipelineExecutionId": "f6bf1671-d706-4b28-EXAMPLE",
                "status": "Succeeded"
            }
        }
    ],
    "created": 1585622700.512,
    "updated": 1586273472.662
```

```
}
```

get-pipeline-execution でステータスとソースリビジョンを表示する (CLI)

パイプラインの実行に使用するソースアーティファクト (パイプラインの最初のステージで生成された出力アーティファクト) に関する詳細を表示できます。この詳細には、コミット ID、チェックインコメント、アーティファクト作成後または更新後の時間、CLI の使用時間、ビルドアクションのバージョン番号などの識別子が含まれます。一部のリビジョンタイプでは、アーティファクトバージョンのコミットの URL を表示して開くことができます。ソースのリビジョンは、以下で構成されます。

- Summary: アーティファクトの最新のリビジョンに関する概要。GitHub および AWS CodeCommit リポジトリの場合、コミットメッセージ。Amazon S3 バケットまたはアクションの場合は、オブジェクトメタデータに指定された codepipeline-artifact-revision-summary キーのユーザー指定コンテンツ。
- revisionUrl: アーティファクトリビジョンのコミット ID。GitHub または AWS CodeCommit リポジトリに保存されているアーティファクトの場合、コミット ID はコミットの詳細ページにリンクされます。

get-pipeline-execution コマンドを実行して、パイプライン実行で追加された最新のソースリビジョンに関する情報を表示できます。get-pipeline-state コマンドを実行して、パイプラインのすべてのステージに関する詳細を取得したら、ソースリビジョンの詳細を必要とするステージに適用される実行 ID を特定します。次に、その実行 ID を get-pipeline-execution コマンドで使用します (パイプラインのステージは、別のパイプラインの実行時に正常に完了している場合があるため、別の実行 ID が割り当てられていることがあります。)

つまり、Staging ステージに現在あるアーティファクトに関する詳細を表示する場合は、get-pipeline-state コマンドを実行し、Staging ステージの現在の実行 ID を特定してから、その実行 ID を使用して get-pipeline-execution コマンドを実行します。

パイプラインのステータスとソースリビジョンを表示するには

1. ターミナル (Linux、macOS、Unix) またはコマンドプロンプト (Windows) を開き、 AWS CLI を使用して get-pipeline-state のコマンドを実行します。*MyFirstPipeline* という名前のパイプラインののために、以下のように入力します：

```
aws codepipeline get-pipeline-state --name MyFirstPipeline
```

このコマンドは、各ステージの最新のパイプライン実行 ID を含む、パイプラインの最新の状態を返します。

2. パイプライン実行の詳細を表示するには、get-pipeline-execution コマンドを実行します。この場合、パイプラインの一意の名前と、アーティファクトの詳細を表示する特定の実行のパイプライン実行 ID を指定します。例えば、名前が *MyFirstPipeline* で、実行 ID が 3137f7cb-7cf7-039j-s83l-d7eu3EXAMPLE のパイプラインの実行に関する詳細を表示するには、以下のように入力します：

```
aws codepipeline get-pipeline-execution --pipeline-name MyFirstPipeline --pipeline-
execution-id 3137f7cb-7cf7-039j-s83l-d7eu3EXAMPLE
```

このコマンドは、パイプライン実行の一部である各ソースリビジョンに関する情報と、パイプラインを識別する情報を返します。実行に含まれていたパイプラインステージに関する情報のみが含まれます。パイプライン実行の一部ではなかった、パイプラインの他のステージが存在する可能性があります。

以下の例は、*MyFirstPipeline* という名前のパイプラインの一部に対して返されたデータを示します。ここで、"MyApp" という名前のアーティファクトは GitHub リポジトリに保存されています。

3.
```
{
    "pipelineExecution": {
        "artifactRevisions": [
            {
                "created": 1427298837.7689769,
                "name": "MyApp",
                "revisionChangeIdentifier": "1427298921.3976923",
                "revisionId": "7636d59f3c461cEXAMPLE8417dbc6371",
                "revisionSummary": "Updating the application for feature 12-4820",
                "revisionUrl": "https://api.github.com/repos/anycompany/MyApp/git/
commits/7636d59f3c461cEXAMPLE8417dbc6371"
            }
        ],
        "pipelineExecutionId": "3137f7cb-7cf7-039j-s83l-d7eu3EXAMPLE",
        "pipelineName": "MyFirstPipeline",
        "pipelineVersion": 2,
        "status": "Succeeded",
        "executionMode": "SUPERSEDED",
        "executionType": "ROLLBACK",
        "rollbackMetadata": {
```

```
            "rollbackTargetPipelineExecutionId": "4f47bed9-6998-476c-a49d-
e60beEXAMPLE"
        }
    }
}
```

`list-action-executions` でアクション実行を表示する (CLI)

パイプラインのアクションの実行に関する詳細として、アクションの実行 ID、入力アーティファクト、出力アーティファクト、実行結果、ステータスなどを表示できます。パイプライン実行のアクションのリストを取得するには、実行 ID フィルタを指定します。

> **Note**
>
> 詳細な実行履歴は 2019 年 2 月 21 日以降に実行されたものについて利用できます。

- パイプラインのアクションの実行を表示するには、以下のいずれかを実行します。
  - パイプラインのすべてのアクション実行に関する詳細を表示するには、パイプラインの一意の名前を指定して、list-action-executions コマンドを実行します。例えば、*MyFirstPipeline* という名前のパイプラインについてアクションの実行を表示するには、以下のように入力します:

    ```
    aws codepipeline list-action-executions --pipeline-name MyFirstPipeline
    ```

    以下は、このコマンドの出力例の一部です。

    ```
    {
        "actionExecutionDetails": [
            {
                "actionExecutionId": "ID",
                "lastUpdateTime": 1552958312.034,
                "startTime": 1552958246.542,
                "pipelineExecutionId": "Execution_ID",
                "actionName": "Build",
                "status": "Failed",
                "output": {
                    "executionResult": {
                        "externalExecutionUrl": "Project_ID",
    ```

```
                    "externalExecutionSummary": "Build terminated with state:
 FAILED",
                    "externalExecutionId": "ID"
                },
                "outputArtifacts": []
            },
            "stageName": "Beta",
            "pipelineVersion": 8,
            "input": {
                "configuration": {
                    "ProjectName": "java-project"
                },
                "region": "us-east-1",
                "inputArtifacts": [
                    {
                        "s3location": {
                            "bucket": "codepipeline-us-east-1-ID",
                            "key": "MyFirstPipeline/MyApp/Object.zip"
                        },
                        "name": "MyApp"
                    }
                ],
                "actionTypeId": {
                    "version": "1",
                    "category": "Build",
                    "owner": "AWS",
                    "provider": "CodeBuild"
                }
            }
        },

. . .
```

- パイプラインの実行についてすべてのアクションの実行を表示するには、パイプラインの一意の名前と実行 ID を指定して、list-action-executions コマンドを実行します。例えば、*Execution_ID* のアクションの実行を表示するには、以下のように入力します。

```
aws codepipeline list-action-executions --pipeline-name MyFirstPipeline --filter
 pipelineExecutionId=Execution_ID
```

- 以下は、このコマンドの出力例の一部です。

```
{
```

```
    "actionExecutionDetails": [
        {
            "stageName": "Beta",
            "pipelineVersion": 8,
            "actionName": "Build",
            "status": "Failed",
            "lastUpdateTime": 1552958312.034,
            "input": {
                "configuration": {
                    "ProjectName": "java-project"
                },
                "region": "us-east-1",
                "actionTypeId": {
                    "owner": "AWS",
                    "category": "Build",
                    "provider": "CodeBuild",
                    "version": "1"
                },
                "inputArtifacts": [
                    {
                        "s3location": {
                            "bucket": "codepipeline-us-east-1-ID",
                            "key": "MyFirstPipeline/MyApp/Object.zip"
                        },
                        "name": "MyApp"
                    }
                ]
            },


. . .
```

パイプライン実行モードを設定または変更する

パイプラインの実行モードを設定して、複数の実行を処理する方法を指定できます。

パイプラインの実行モードの詳細については、「パイプライン実行の仕組み」を参照してください。

> **Important**
>
> PARALLEL モードのパイプラインの場合、パイプライン実行モードを QUEUED または SUPERSEDED に編集すると、パイプラインの状態には更新された状態が PARALLEL とし

て表示されません。詳細については、「PARALLEL モードから変更したパイプラインに、以前の実行モードが表示されます。」を参照してください。

> **Important**
>
> PARALLEL モードのパイプラインの場合、パイプライン実行モードを QUEUED または SUPERSEDED に編集すると、各モードのパイプラインのパイプライン定義は更新されません。詳細については、「PARALLEL モードのパイプラインは、QUEUED モードまたは SUPERSEDED モードに変更して編集したときに、パイプライン定義が古いままになります。」を参照してください。

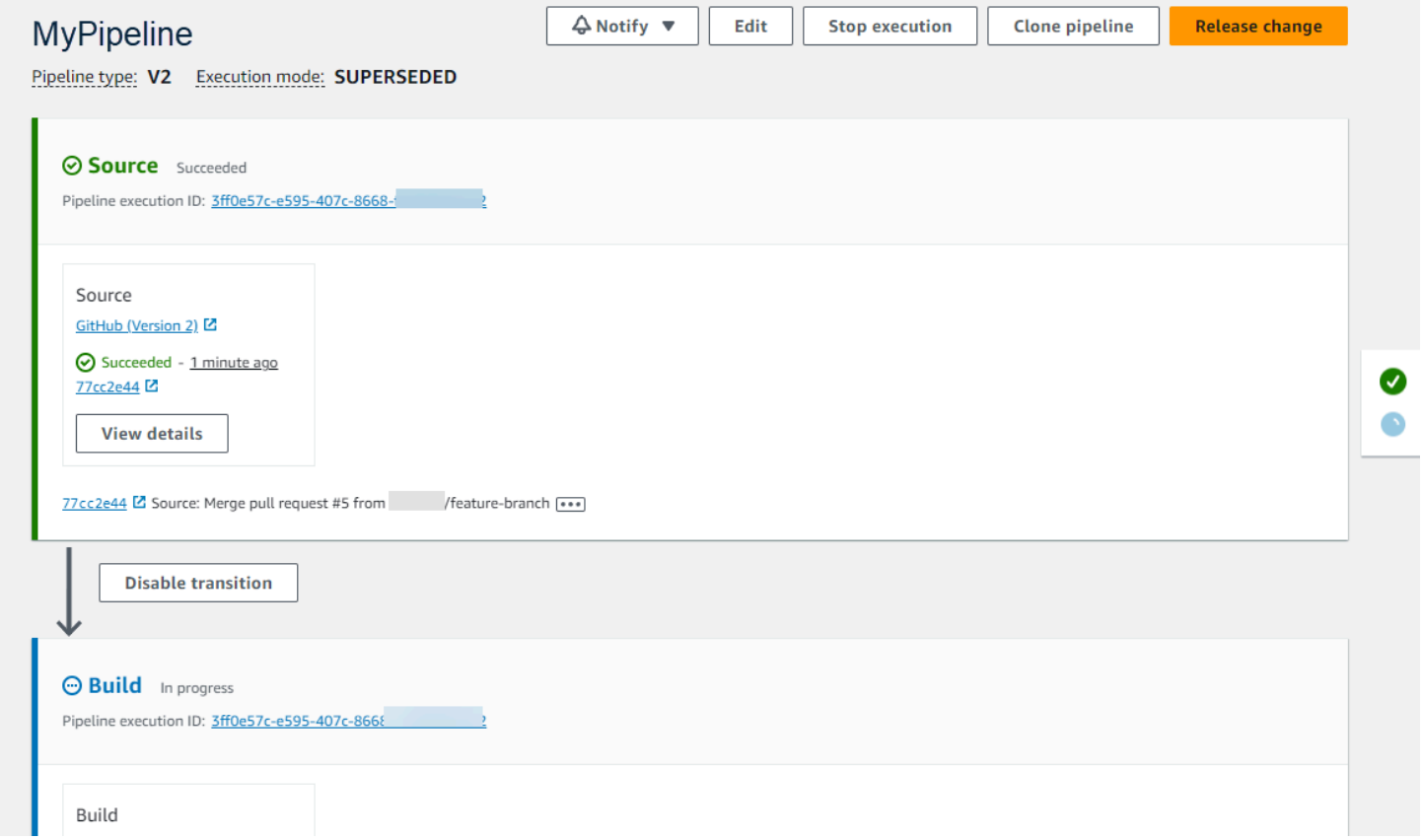
> **Important**
>
> PARALLEL モードのパイプラインでは、ステージのロールバックは使用できません。同様に、ロールバック結果タイプの障害条件を PARALLEL モードのパイプラインに追加することはできません。

実行モードの表示に関する考慮事項

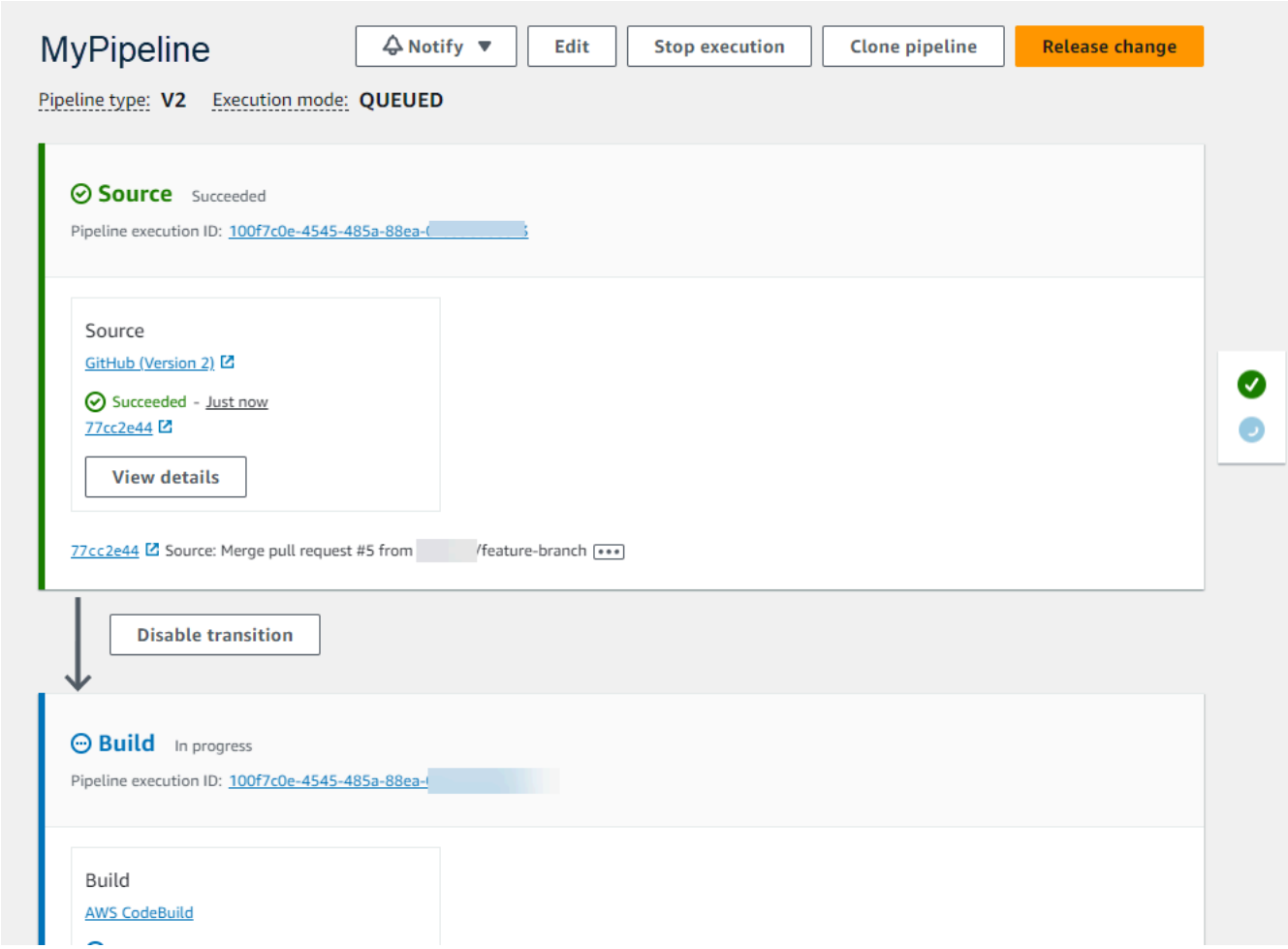
特定の実行モードでパイプラインを表示する場合の考慮事項があります。

SUPERSEDEDED モードと QUEUED モードの場合、進行中の実行を表示するにはパイプラインビューを使用し、詳細と履歴を表示するには実行 ID をクリックします。PARALLEL モードの場合は、実行 ID をクリックして [可視化] タブで進行中の実行を表示します。

次のビューは、CodePipeline の SUPERSEDED モードを示しています。

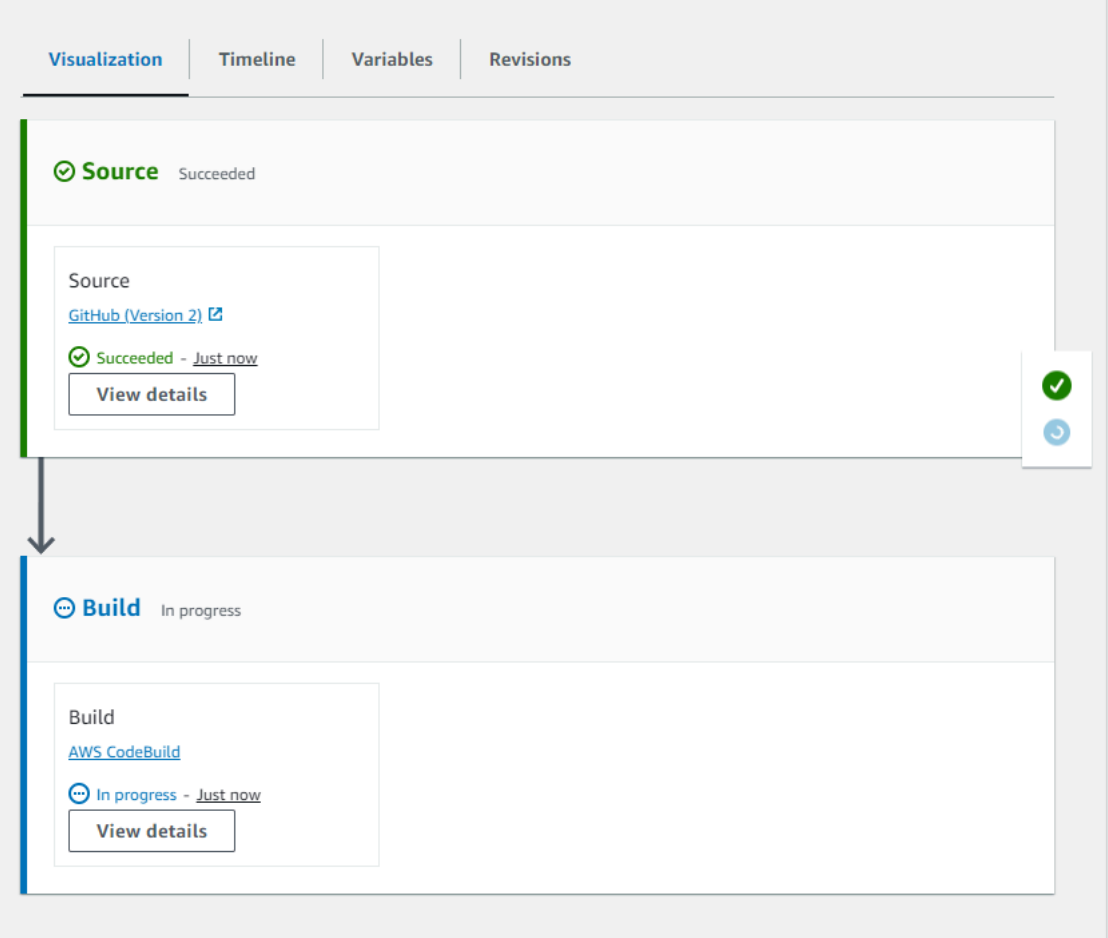


次のビューは、CodePipeline の QUEUED モードを示しています。



次のビューは、CodePipeline の PARALLEL モードを示しています。

> **Important**
>
> PARALLEL モードのパイプラインでは、ステージのロールバックは使用できません。同様に、ロールバック結果タイプの障害条件を PARALLEL モードのパイプラインに追加することはできません。



実行モード間を切り替える場合の考慮事項

パイプラインのモードを変更する場合のパイプラインに関する考慮事項を以下に示します。[編集] モードで、ある実行モードから別の実行モードに切り替えて変更を保存すると、特定のビューや状態が調整される場合があります。

例えば、PARALLEL モードから QUEUED モードまたは SUPERSEDED モードに切り替えると、PARALLEL モードで開始した実行が継続されます。これらは、実行履歴ページで確認できます。パイプラインビューには、以前の QUEUED モードまたは SUPERSEDED モードで実行された実行が表示されます。それ以外の場合は、空の状態が表示されます。

別の例として、QUEUED または SUPERSEDED から PARALLEL モードに切り替えると、パイプラインのビュー/状態ページが表示されなくなります。PARALLEL モードで実行を表示するには、実行の詳細ページで [可視化] タブを使用します。SUPERSEDEDED モードまたは QUEUED モードで開始された実行はキャンセルされます。

次の表に詳細を示します。

モードの変更	保留中およびアクティブな実行の詳細	パイプライン状態の詳細
SUPERSEDEDED から SUPERSEDED へ/SUPERSEDED から QUEUED へ	• アクティブな実行は、進行中のアクションが完了した後にキャンセルされます。 • 保留中の実行はキャンセルされます。	キャンセルされた などのパイプラインの状態は、最初のモードのバージョンと 2 番目のモードの間で保持されます。
QUEUED から QUEUED へ/QUEUED から SUPERSEDED へ	• アクティブな実行は、進行中のアクションが完了した後にキャンセルされます。 • 保留中の実行はキャンセルされます。	キャンセルされた などのパイプラインの状態は、最初のモードのバージョンと 2 番目のモードの間で保持されます。
PARALLEL から PARALLEL へ	すべての実行は、パイプライン定義の更新とは関係なく実行できます。	空になります。並列モードにはパイプライン状態がありません。
SUPERSEDEDED から PARALLEL へ/QUEUED から PARALLEL へ	• アクティブな実行は、進行中のアクションが完了した後にキャンセルされます。 • 保留中の実行はキャンセルされます。	空になります。並列モードにはパイプライン状態がありません。

パイプライン実行モードを設定または変更する (コンソール)

コンソールを使用してパイプライン実行モードを設定できます。

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

   AWS アカウントに関連付けられているすべてのパイプラインの名前とステータスが表示されます。
2. [名前] で、編集するパイプラインの名前を選択します。
3. パイプライン詳細ページで、[編集] を選択します。

4. [編集] ページで、[編集: パイプラインプロパティ] を選択します。
5. パイプラインのモードを選択します。
   - 優先
   - キュー (パイプラインタイプ V2 が必要)
   - 並列 (パイプラインタイプ V2 が必要)
6. [編集] ページで [完了] を選択します。

パイプライン実行モードを設定する (CLI)

を使用してパイプライン実行モード AWS CLI を設定するには、 `create-pipeline`または `update-pipeline` コマンドを使用します。

1. ターミナルセッション (Linux、macOS または Unix) またはコマンドプロンプト (Windows) を開き、get-pipeline コマンドを実行して、パイプライン構造を JSON ファイルにコピーします。たとえば、**MyFirstPipeline** という名前のパイプラインに対して、以下のコマンドを入力します。

   ```
   aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
   ```

   このコマンドは何も返しませんが、作成したファイルは、コマンドを実行したディレクトリにあります。
2. 任意のプレーンテキストエディタで JSON ファイルを開き、設定するパイプライン実行モード (QUEUED など) を反映するようにファイルの構造を変更します。

   ```
   "executionMode": "QUEUED"
   ```

   次の例は、2 つのステージを持つサンプルパイプラインで、実行モードを QUEUED に設定する方法を示しています。

   ```
   {
       "pipeline": {
           "name": "MyPipeline",
           "roleArn": "arn:aws:iam::111122223333:role/service-role/
   AWSCodePipelineServiceRole-us-east-1-dkpippe",
           "artifactStore": {
               "type": "S3",
   ```

```
            "location": "bucket"
        },
        "stages": [
            {
                "name": "Source",
                "actions": [
                    {
                        "name": "Source",
                        "actionTypeId": {
                            "category": "Source",
                            "owner": "AWS",
                            "provider": "CodeCommit",
                            "version": "1"
                        },
                        "runOrder": 1,
                        "configuration": {
                            "BranchName": "main",
                            "OutputArtifactFormat": "CODE_ZIP",
                            "PollForSourceChanges": "true",
                            "RepositoryName": "MyDemoRepo"
                        },
                        "outputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "inputArtifacts": [],
                        "region": "us-east-1",
                        "namespace": "SourceVariables"
                    }
                ]
            },
            {
                "name": "Build",
                "actions": [
                    {
                        "name": "Build",
                        "actionTypeId": {
                            "category": "Build",
                            "owner": "AWS",
                            "provider": "CodeBuild",
                            "version": "1"
                        },
                        "runOrder": 1,
```

```
                        "configuration": {
                            "ProjectName": "MyBuildProject"
                        },
                        "outputArtifacts": [
                            {
                                "name": "BuildArtifact"
                            }
                        ],
                        "inputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "region": "us-east-1",
                        "namespace": "BuildVariables"
                    }
                ]
            }
        ],
        "version": 1,
        "executionMode": "QUEUED"
    }
}
```

3. get-pipeline コマンドを使用して取得したパイプライン構造を使用している場合、JSON ファイルの構造を変更する必要があります。metadata コマンドが使用できるように、ファイルから update-pipeline 行を削除する必要があります。JSON ファイルのパイプライン構造からセクションを削除します ("metadata": { } 行と、"created"、"pipelineARN"、および"updated"フィールド)。

   例えば、構造から以下の行を削除します。

```
"metadata": {
  "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
  "created": "date",
  "updated": "date"
  }
```

   ファイルを保存します。

4. 変更を適用するには、パイプライン JSON ファイルを指定して、update-pipeline コマンドを実行します。

> **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 codepipeline update-pipeline --cli-input-json file://pipeline.json
```

このコマンドは、編集したパイプラインの構造全体を返します。

> **Note**
>
> update-pipeline コマンドは、パイプラインを停止します。update-pipeline コマンドを実行したときにパイプラインによりリビジョンが実行されている場合、その実行は停止します。更新されたパイプラインによりそのリビジョンを実行するには、パイプラインを手動で開始する必要があります。

ステージをロールバックまたは再試行する

AWS CodePipeline コンソールまたは AWS CLI を使用して、ステージまたはステージ内のアクションを手動でロールバックまたは再試行できます。ロールバックまたは再試行を設定済みの結果として使用するステージの条件を設定するには、「ステージの条件を設定する」を参照してください。

トピック

- 失敗したステージまたは失敗したアクションのステージ再試行の設定
- ステージロールバックの設定

失敗したステージまたは失敗したアクションのステージ再試行の設定

パイプラインを最初から再実行することなく、失敗したステージを再試行できます。そのためには、ステージ内の失敗したアクションを再試行するか、ステージ内の最初のアクションから始めてステージ内のすべてのアクションを再試行します。ステージ内の失敗したアクションを再試行する場合、進行中のすべてのアクションはそのまま動作し続け、失敗したアクションは再度トリガーされます。失敗したアクションのあるステージ内で最初のアクションから再試行する場合、ステージに進行中のアクションがないことが必要です。ステージを再試行するには、すべてのアクションが失敗しているか、一部のアクションが失敗して他のアクションが成功している必要があります。

> **Important**
>
> 失敗したステージを再試行すると、そのステージ内の最初のアクションからすべてのアクションが再試行されます。失敗したアクションを再試行すると、ステージ内のすべての失敗したアクションが再試行されます。これにより、同じ実行内で以前に成功したアクションの出力アーティファクトは上書きされます。
> アーティファクトは上書きされる場合がありますが、以前に成功したアクションの実行履歴は保持されます。

コンソールを使用してパイプラインを表示している場合、再試行できるステージに [ステージの再試行] ボタンまたは [失敗したアクションの再試行] ボタンが表示されます。

AWS CLI を使用している場合は、 get-pipeline-state コマンドを使用して、アクションが失敗したかどうかを判断できます。

> **Note**
>
> 次の場合は、ステージを再試行できないことがあります。
>
> - ステージ内のすべてのアクションが成功したため、ステージが失敗ステータスになっていない。
> - ステージが失敗した後、パイプライン全体の構造が変更された。
> - ステージで別の再試行がすでに進行中です。

トピック

- ステージ再試行に関する考慮事項
- 失敗したステージを手動で再試行する
- ステージ障害時の自動再試行を設定する

ステージ再試行に関する考慮事項

ステージ再試行に関する考慮事項は以下のとおりです。

- ステージ障害時の自動再試行は、1 回の再試行に対してのみ設定できます。
- ステージ障害時の自動再試行は、Source アクションを含むすべてのアクションに対して設定できます。

失敗したステージを手動で再試行する

コンソールまたは CLI を使用して、失敗したステージを手動で再試行できます。

「ステージ障害時の自動再試行を設定する」で説明しているように、障害時に自動的に再試行するようにステージを設定することもできます。

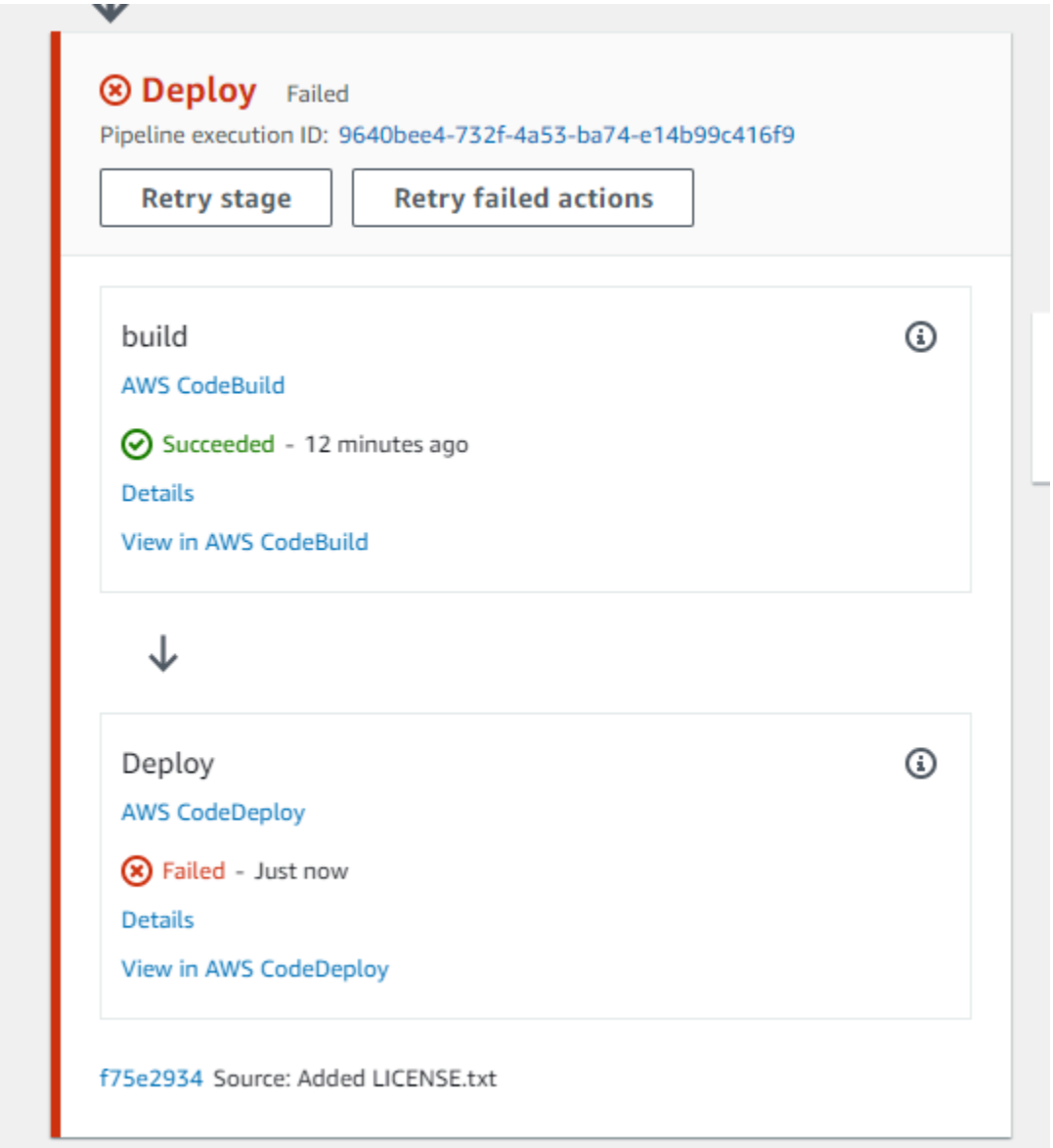
失敗したステージを手動で再試行する (コンソール)

失敗したステージまたはステージ内の失敗したアクションを再試行するには

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

AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。

2. [Name] で、パイプラインの名前を選択します。
3. 失敗したアクションのあるステージを見つけ、次のいずれかを選択します。
   - ステージ内のすべてのアクションを再試行するには、[ステージの再試行] を選択します。
   - ステージ内の失敗したアクションのみを再試行するには、[失敗したアクションの再試行] を選択します。



再試行したアクションがすべて正常に完了した場合、パイプラインは続行されます。

失敗したステージを手動で再試行する (CLI)

失敗したステージまたはステージ内の失敗したアクションを再試行するには - CLI

を使用してすべてのアクションまたは失敗したすべてのアクション AWS CLI を再試行するには、次のパラメータを指定して retry-stage-execution コマンドを実行します。

```
--pipeline-name <value>
--stage-name <value>
--pipeline-execution-id <value>
--retry-mode ALL_ACTIONS/FAILED_ACTIONS
```

> **Note**
>
> `retry-mode` に使用できる値は `FAILED_ACTIONS` と `ALL_ACTIONS` です。

1. ターミナル (Linux、macOS、UNIX) またはコマンドプロンプト (Windows) で、retry-stage-execution コマンドを実行します。次の例では、`MyPipeline` という名前のパイプラインに対して実行しています。

   ```
   aws codepipeline retry-stage-execution --pipeline-name MyPipeline --stage-name
    Deploy --pipeline-execution-id b59babff-5f34-EXAMPLE --retry-mode FAILED_ACTIONS
   ```

   出力は実行 ID を返します。

   ```
   {
       "pipelineExecutionId": "b59babff-5f34-EXAMPLE"
   }
   ```

2. JSON 入力ファイルを使用してコマンドを実行することもできます。まず、パイプライン、失敗したアクションを含むステージ、そのステージでの最新のパイプラインの実行を識別する JSON ファイルを作成します。その後、retry-stage-execution コマンドに `--cli-input-json` パラメータを指定して実行します。JSON ファイルに必要な詳細を取得するには、get-pipeline-state コマンドを使用するのが最も簡単です。

   a. ターミナル (Linux、macOS、UNIX) またはコマンドプロンプト (Windows) で、パイプラインの get-pipeline-state コマンドを実行します。例えば、「MyFirstPipeline」という名前のパイプラインに対しては、以下のようなコマンドを入力します。

```
aws codepipeline get-pipeline-state --name MyFirstPipeline
```

コマンドに対する応答には、各ステージのパイプラインの状態情報が含まれます。以下の例では、応答は [Staging] ステージで 1 つ以上のアクションが失敗したことを示しています。

```
{
    "updated": 1427245911.525,
    "created": 1427245911.525,
    "pipelineVersion": 1,
    "pipelineName": "MyFirstPipeline",
    "stageStates": [
        {
            "actionStates": [...],
            "stageName": "Source",
            "latestExecution": {
                "pipelineExecutionId": "9811f7cb-7cf7-SUCCESS",
                "status": "Succeeded"
            }
        },
        {
            "actionStates": [...],
            "stageName": "Staging",
            "latestExecution": {
                "pipelineExecutionId": "3137f7cb-7cf7-EXAMPLE",
                "status": "Failed"
            }
        }
    ]
}
```

b. プレーンテキストエディタで、JSON 形式で以下の情報を記録するファイルを作成します。

- 失敗したアクションを含むパイプラインの名前
- 失敗したアクションを含むステージの名前
- ステージでの最新のパイプラインの実行 ID
- 再試行モード

先ほどの MyFirstPipeline の例では、ファイルは以下のようになります。

```
{
    "pipelineName": "MyFirstPipeline",
    "stageName": "Staging",
    "pipelineExecutionId": "3137f7cb-7cf7-EXAMPLE",
    "retryMode": "FAILED_ACTIONS"
}
```

c. **retry-failed-actions.json** のような名前でファイルを保存します。

d. retry-stage-execution コマンドを実行したときに作成したファイルを呼び出します。例:

> **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 codepipeline retry-stage-execution --cli-input-json file://retry-failed-actions.json
```

e. 再試行の結果を表示するには、CodePipeline コンソールを開いてから、失敗したアクションが含まれるパイプラインを選択するか、もう一度 get-pipeline-state コマンドを使用します。詳細については、「CodePipeline でパイプラインと詳細を表示する」を参照してください。

ステージ障害時の自動再試行を設定する

ステージ障害時の自動再試行を設定できます。ステージは 1 回の再試行を行い、失敗したステージの再試行ステータスを [パイプラインを表示する] ページに表示します。

再試行モードを設定するには、自動再試行の対象を、失敗したステージ内のすべてのアクションにするか、失敗したアクションのみにするかを指定します。

ステージ障害時の自動再試行を設定する (コンソール)

コンソールを使用して、自動的に再試行するようにステージを設定できます。

自動的に再試行するようにステージを設定する (コンソール)

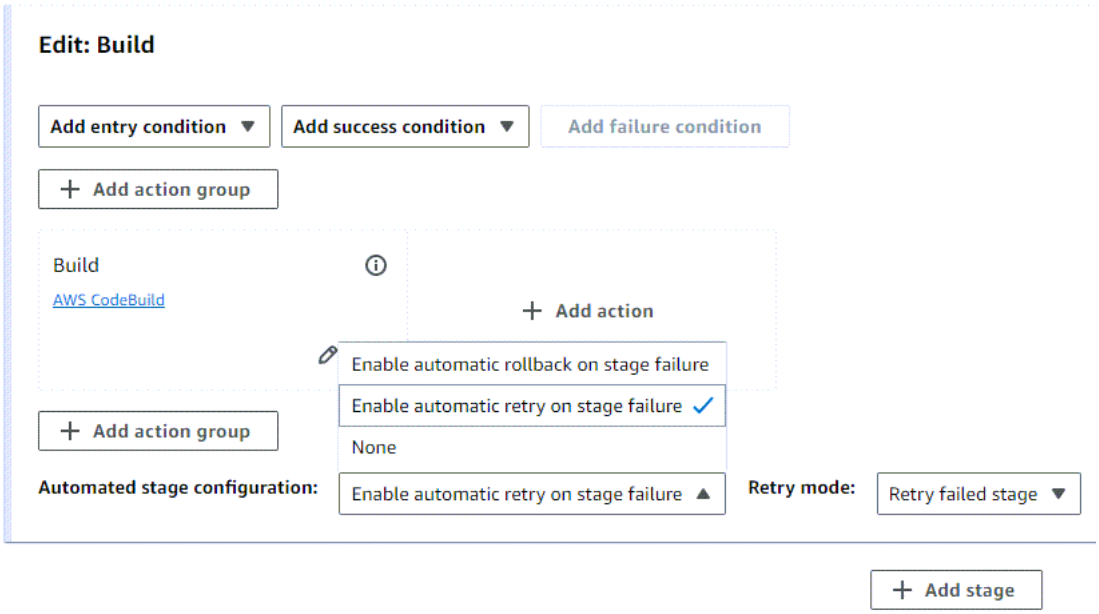
1. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

AWS アカウントに関連付けられているすべてのパイプラインの名前とステータスが表示されます。

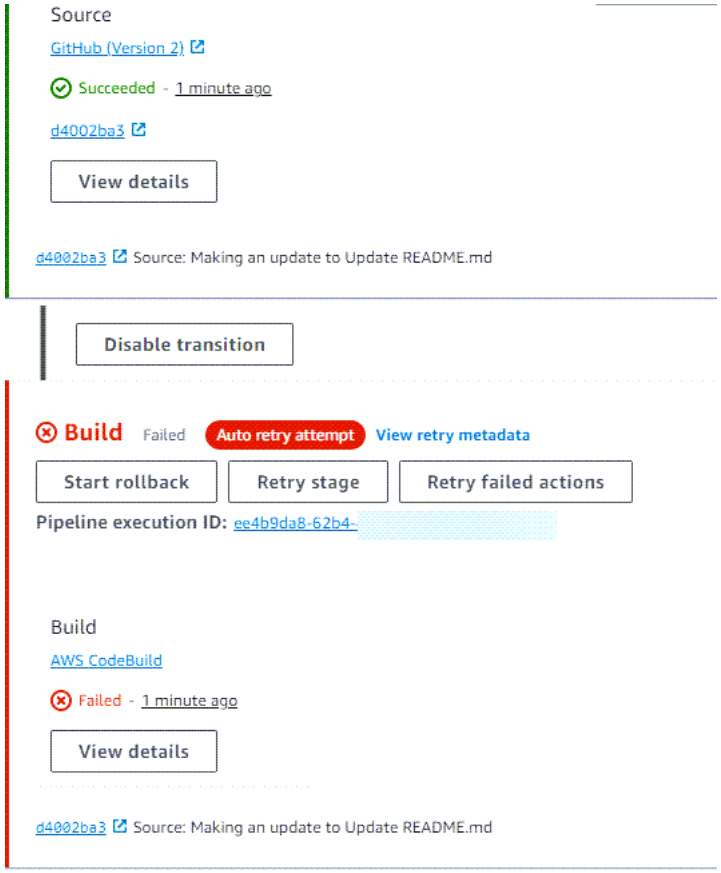
2. [名前] で、編集するパイプラインの名前を選択します。
3. パイプライン詳細ページで、[編集] を選択します。
4. [編集] ページで、編集するアクションに関連する [ステージを編集する] を選択します。
5. [自動ステージ設定]、[ステージ障害時の自動再試行を有効にする] の順に選択します。パイプラインに変更を保存します。
6. [自動ステージ設定] で、次のいずれかの再試行モードを選択します。

   - ステージ内のすべてのアクションを再試行するように指定するには、[失敗したステージを再試行] を選択します。
   - ステージ内の失敗したアクションのみを再試行するように指定するには、[失敗したアクションを再試行] を選択します。

   パイプラインに変更を保存します。



7. パイプラインの実行後に、ステージの障害が発生した場合は、自動再試行が行われます。以下の例は、自動的に再試行されたビルドステージを示しています。



8. 再試行の詳細を表示するには、[詳細を表示] を選択します。ウィンドウが表示されます。



自動的に再試行するようにステージを設定する (CLI)

を使用して、障害時に自動的に再試行するようにステージ AWS CLI を設定するには、 コマンドを使用して、 パイプライン、ステージ、アクションを作成するおよび で説明されているようにパイプラインを作成または更新しますCodePipeline でパイプラインを編集する。

- ターミナル (Linux、macOS、Unix) またはコマンドプロンプト (Windows) を開きます。次に AWS CLI を使用し、パイプライン構造の失敗条件を指定して update-pipeline コマンドを実行します。次の例では、S3Deploy という名前のステージの自動再試行を設定します。

```
{
                "name": "S3Deploy",
                "actions": [
                    {
                        "name": "s3deployaction",
                        "actionTypeId": {
                            "category": "Deploy",
                            "owner": "AWS",
                            "provider": "S3",
                            "version": "1"
                        },
                        "runOrder": 1,
                        "configuration": {
                            "BucketName": "static-website-bucket",
                            "Extract": "false",
                            "ObjectKey": "SampleApp.zip"
                        },
                        "outputArtifacts": [],
                        "inputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "region": "us-east-1"
                    }
                ],
                 "onFailure": {
                    "result": "RETRY",
                    "retryConfiguration": {
                        "retryMode": "ALL_ACTIONS",
                    },
            }
```

自動的に再試行するようにステージを設定する (CloudFormation)

CloudFormation を使用して、障害発生時の自動再試行用にステージを設定するには、OnFailureステージライフサイクルパラメータを使用します。RetryConfiguration パラメータを使用して再試行モードを設定します。

```
OnFailure:
     Result: RETRY
     RetryConfiguration:
         RetryMode: ALL_ACTIONS
```

- 次のスニペットに示すように、テンプレートを更新します。次の例では、Release という名前のステージの自動再試行を設定します。

```
AppPipeline:
  Type: AWS::CodePipeline::Pipeline
  Properties:
    RoleArn:
      Ref: CodePipelineServiceRole
    Stages:
      -
        Name: Source
        Actions:
          -
            Name: SourceAction
            ActionTypeId:
              Category: Source
              Owner: AWS
              Version: 1
              Provider: S3
            OutputArtifacts:
              -
                Name: SourceOutput
            Configuration:
              S3Bucket:
                Ref: SourceS3Bucket
              S3ObjectKey:
                Ref: SourceS3ObjectKey
            RunOrder: 1
      -
        Name: Release
        Actions:
```

```
        -
          Name: ReleaseAction
          InputArtifacts:
            -
              Name: SourceOutput
          ActionTypeId:
            Category: Deploy
            Owner: AWS
            Version: 1
            Provider: CodeDeploy
          Configuration:
            ApplicationName:
              Ref: ApplicationName
            DeploymentGroupName:
              Ref: DeploymentGroupName
          RunOrder: 1
     OnFailure:
         Result: RETRY
         RetryConfiguration:
            RetryMode: ALL_ACTIONS
  ArtifactStore:
    Type: S3
    Location:
      Ref: ArtifactStoreS3Location
    EncryptionKey:
      Id: arn:aws:kms:useast-1:ACCOUNT-ID:key/KEY-ID
      Type: KMS
  DisableInboundStageTransitions:
    -
      StageName: Release
      Reason: "Disabling the transition until integration tests are completed"
  Tags:
    - Key: Project
      Value: ProjectA
    - Key: IsContainerBased
      Value: 'true'
```

障害時にステージを再試行するように設定する方法の詳細については、「CloudFormation ユーザーガイド」で「StageDeclaration」の「OnFailure」を参照してください。

ステージロールバックの設定

ステージは、そのステージで成功した実行にロールバックできます。失敗時にステージをロールバックするように事前設定することも、手動でステージをロールバックすることもできます。ロールバック操作により、新しい実行が開始されます。ロールバック用に選択したターゲットパイプライン実行は、ソースリビジョンと変数を取得するために使用します。

実行のタイプ (標準またはロールバックのいずれか) は、パイプラインの履歴、パイプラインの状態、パイプライン実行の詳細に表示されます。

トピック

- ロールバックに関する考慮事項
- ステージを手動でロールバックする
- ステージの自動ロールバックを設定する
- 実行リストでロールバックステータスを表示する
- ロールバックステータスの詳細を表示する

ロールバックに関する考慮事項

ステージのロールバックに関する考慮事項は以下のとおりです。

- ソースステージをロールバックすることはできません。
- パイプラインは、以前の実行が現在のパイプライン構造バージョンで開始されている場合にのみ、以前の実行にロールバックできます。
- ロールバック実行タイプであるターゲット実行 ID にロールバックすることはできません。
- CodePipeline は、ロールバック先の実行の変数とアーティファクトを使用します。

ステージを手動でロールバックする

コンソールまたは CLI を使用して、ステージを手動でロールバックできます。パイプラインは、以前の実行が現在のパイプライン構造バージョンで開始されている場合にのみ、以前の実行にロールバックできます。

「ステージの自動ロールバックを設定する」で説明しているように、失敗時に自動的にロールバックするようにステージを設定することもできます。

ステージを手動でロールバックする (コンソール)

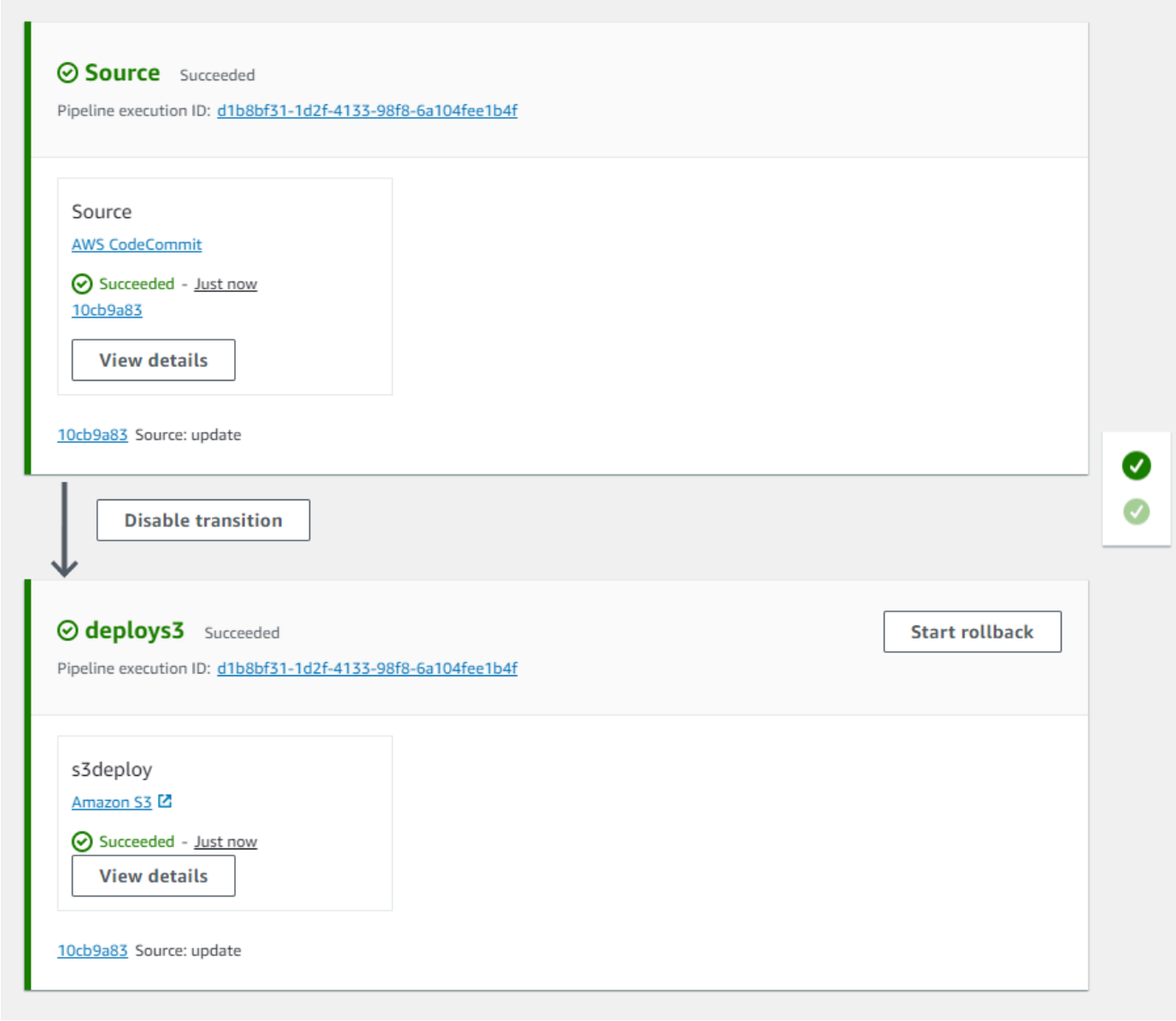
コンソールを使用して、ステージをターゲットパイプライン実行に手動でロールバックできます。ステージをロールバックすると、Rollback ラベルがコンソールのパイプライン可視化に表示されます。

ステージを手動でロールバックする (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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

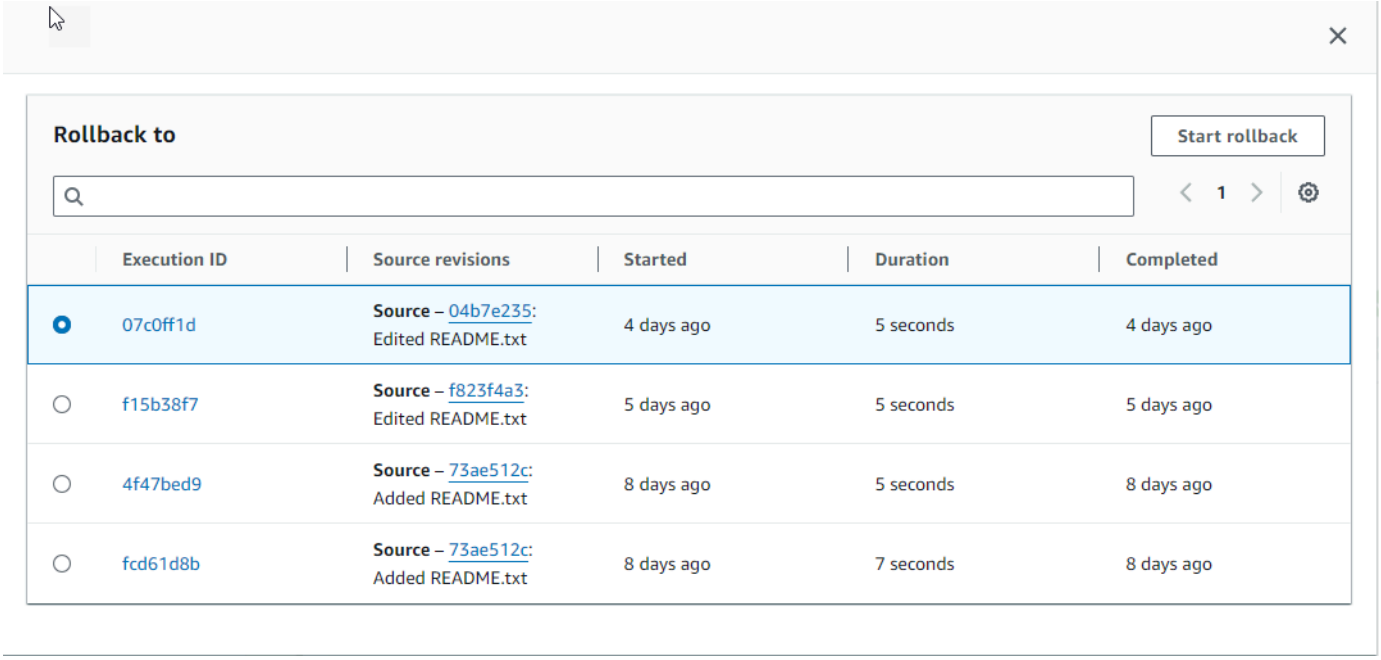
   AWS アカウントに関連付けられているすべてのパイプラインの名前とステータスが表示されます。

2. [名前] で、ロールバックするステージのパイプラインの名前を選択します。

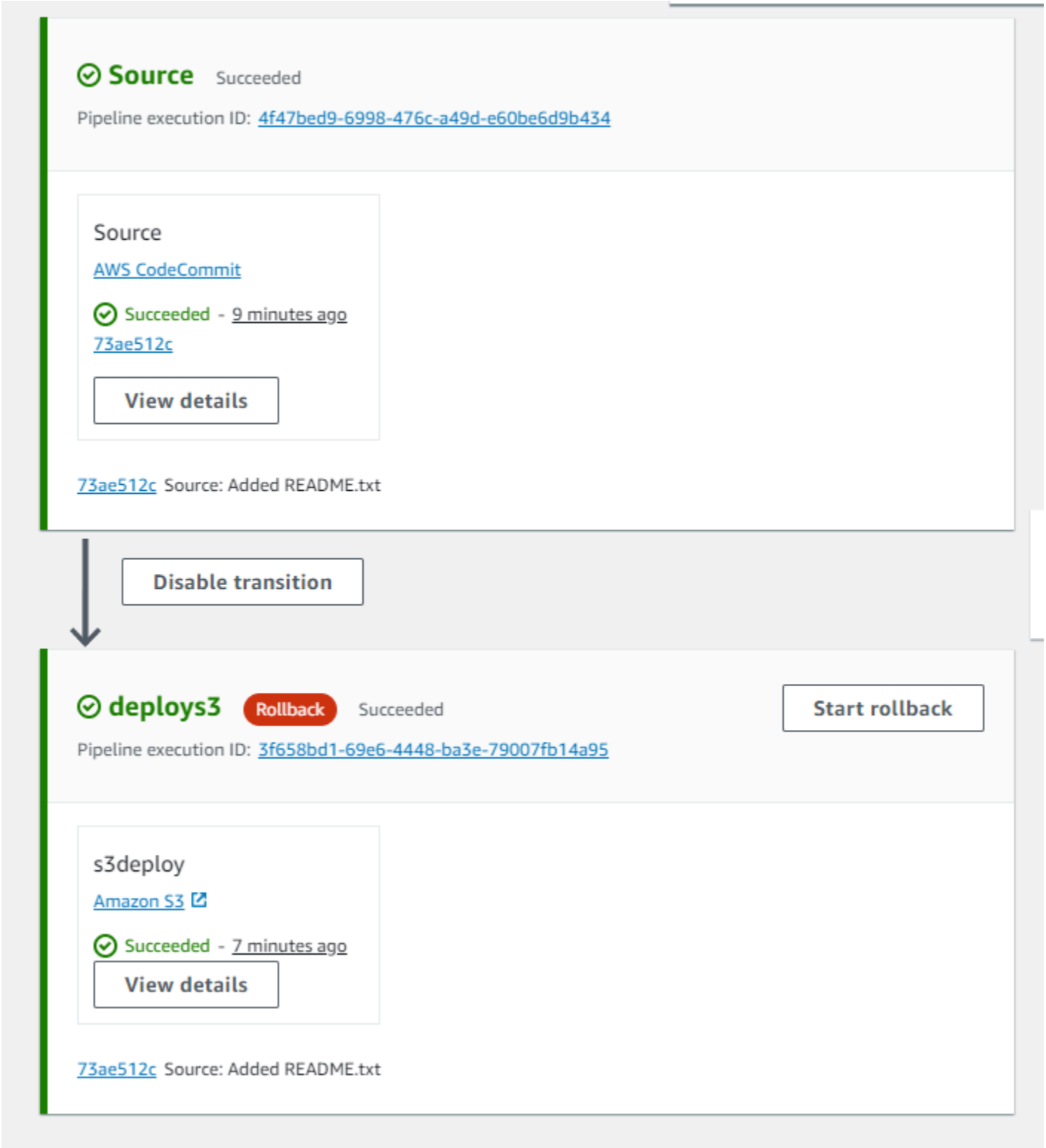


3. ステージで、[ロールバックを開始] を選択します。[次にロールバック:] ページが表示されます。
4. ステージをロールバックする先のターゲット実行を選択します。

> **Note**
>
> 利用可能なターゲットパイプライン実行のリストは、2024 年 2 月 1 日以降の現在のパイプラインバージョンのすべての実行になります。



次の図は、ステージをロールバックした後の新しい実行 ID の例を示しています。



ステージを手動でロールバックする (CLI)

を使用してステージ AWS CLI を手動でロールバックするには、 `rollback-stage` コマンドを使用します。

また、「ステージを手動でロールバックする」で説明しているように、ステージを手動でロールバックすることもできます。

> **Note**
>
> 利用可能なターゲットパイプライン実行のリストは、2024 年 2 月 1 日以降の現在のパイプラインバージョンのすべての実行になります。

ステージを手動でロールバックするには (CLI)

1. 手動ロールバックの CLI コマンドには、ステージで以前に成功したパイプライン実行の実行 ID が必要です。ターゲットパイプライン実行 ID を指定して取得するには、list-pipeline-executions コマンドでステージの成功した実行を返すフィルターを使用します。ターミナル (Linux、macOS、または Unix) またはコマンドプロンプト (Windows) を開き、 AWS CLI を使用して`list-pipeline-executions`コマンドを実行し、パイプラインの名前とステージで正常に実行するためのフィルターを指定します。この例では、MyFirstPipeline という名前のパイプラインの実行と、deploys3 という名前のステージで成功した実行が出力に一覧表示されます。

   ```
   aws codepipeline list-pipeline-executions --pipeline-name MyFirstPipeline --filter
    succeededInStage={stageName=deploys3}
   ```

   出力で、ロールバック用に指定する、以前に成功した実行の実行 ID をコピーします。次のステップで、これをターゲット実行 ID として使用します。

2. ターミナル (Linux、macOS、Unix) またはコマンドプロンプト (Windows) を開きます。次に AWS CLI を使用し、パイプラインの名前、ステージの名前、ロールバック先のターゲット実行を指定して `rollback-stage` コマンドを実行します。例えば、*`MyFirstPipeline`* という名前のパイプラインの Deploy という名前のステージをロールバックするには、次のように指定します。

   ```
   aws codepipeline rollback-stage --pipeline-name MyFirstPipeline --stage-name Deploy
    --target-pipeline-execution-id bc022580-4193-491b-8923-9728dEXAMPLE
   ```

   出力は、ロールバックされた新しい実行の実行 ID を返します。これは、指定したターゲット実行のソースリビジョンとパラメータを使用する別の ID です。

ステージの自動ロールバックを設定する

パイプラインのステージは、失敗時に自動的にロールバックするように設定できます。ステージは、失敗すると、最後の成功した実行にロールバックされます。パイプラインは、以前の実行が現在のパイプライン構造バージョンで開始されている場合にのみ、以前の実行にロールバックできます。自動ロールバック設定はパイプライン定義の一部であるため、パイプラインステージは、パイプラインステージでパイプラインが正常に実行された後にのみ自動ロールバックされます。

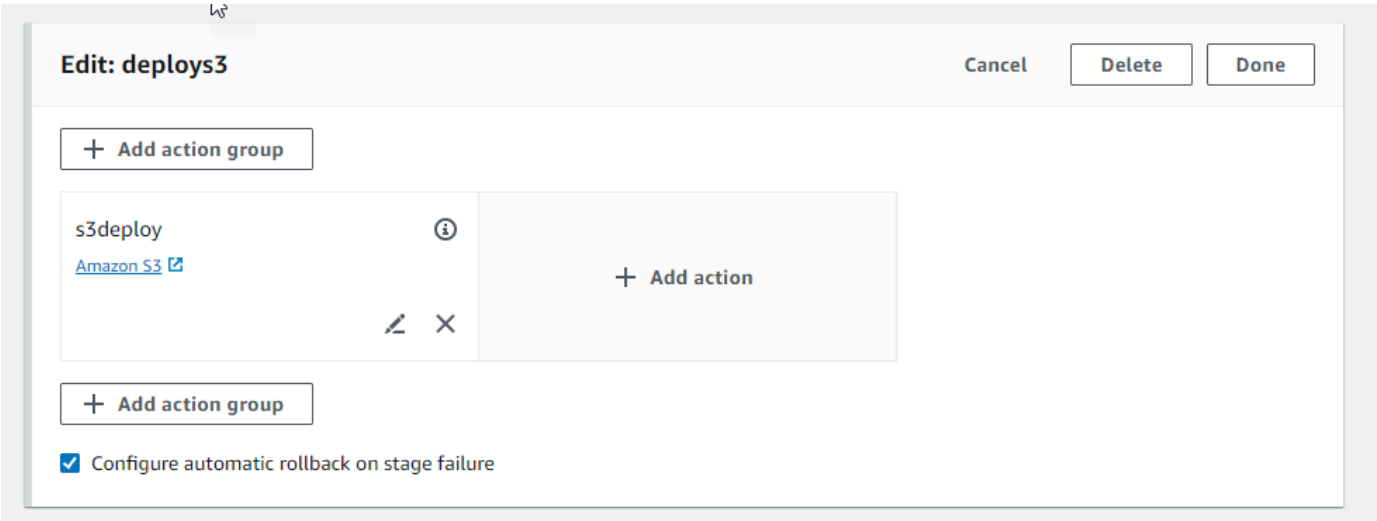
ステージの自動ロールバックを設定する (コンソール)

ステージは、指定した以前の成功した実行にロールバックできます。詳細については、「CodePipeline API ガイド」の「RollbackStage」を参照してください。

ステージの自動ロールバックを設定する (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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

   AWS アカウントに関連付けられているすべてのパイプラインの名前とステータスが表示されます。
2. [名前] で、編集するパイプラインの名前を選択します。
3. パイプライン詳細ページで、[編集] を選択します。
4. [編集] ページで、編集するアクションに関連する [ステージを編集する] を選択します。
5. [自動ステージ設定]、[ステージ障害時の自動ロールバックを設定] の順に選択します。パイプラインに変更を保存します。



ステージの自動ロールバックを設定する (CLI)

を使用して、最後に成功した実行に自動的にロールバックするように失敗したステージ AWS CLI を設定するには、 コマンドを使用して、 パイプライン、ステージ、アクションを作成するおよび で説明されているようにパイプラインを作成または更新しますCodePipeline でパイプラインを編集する。

- ターミナル (Linux、macOS、Unix) またはコマンドプロンプト (Windows) を開きます。次に AWS CLI を使用し、パイプライン構造の失敗条件を指定して update-pipeline コマンドを実行します。次の例では、 という名前のステージの自動ロールバックを設定しますS3Deploy。

```
{
                "name": "S3Deploy",
                "actions": [
                    {
                        "name": "s3deployaction",
                        "actionTypeId": {
                            "category": "Deploy",
                            "owner": "AWS",
                            "provider": "S3",
                            "version": "1"
                        },
                        "runOrder": 1,
                        "configuration": {
                            "BucketName": "static-website-bucket",
                            "Extract": "false",
                            "ObjectKey": "SampleApp.zip"
                        },
                        "outputArtifacts": [],
                        "inputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "region": "us-east-1"
                    }
                ],
                "onFailure": {
                    "result": "ROLLBACK"
                }
            }
```

ステージロールバックの失敗条件の設定の詳細については、「CodePipeline API リファレンス」の「FailureConditions」を参照してください。

ステージの自動ロールバックを設定する (CloudFormation)

CloudFormation を使用して、障害時に自動的にロールバックするようにステージを設定するには、`OnFailure`パラメータを使用します。失敗すると、ステージは自動的に最新の成功した実行にロールバックされます。

```
OnFailure:
     Result: ROLLBACK
```

- 次のスニペットに示すように、テンプレートを更新します。次の例では、 という名前のステージの自動ロールバックを設定します`Release`。

```
AppPipeline:
  Type: AWS::CodePipeline::Pipeline
  Properties:
    RoleArn:
      Ref: CodePipelineServiceRole
    Stages:
      -
        Name: Source
        Actions:
          -
            Name: SourceAction
            ActionTypeId:
              Category: Source
              Owner: AWS
              Version: 1
              Provider: S3
            OutputArtifacts:
              -
                Name: SourceOutput
            Configuration:
              S3Bucket:
                Ref: SourceS3Bucket
              S3ObjectKey:
                Ref: SourceS3ObjectKey
            RunOrder: 1
```

```
      -
        Name: Release
        Actions:
          -
            Name: ReleaseAction
            InputArtifacts:
              -
                Name: SourceOutput
            ActionTypeId:
              Category: Deploy
              Owner: AWS
              Version: 1
              Provider: CodeDeploy
            Configuration:
              ApplicationName:
                Ref: ApplicationName
              DeploymentGroupName:
                Ref: DeploymentGroupName
            RunOrder: 1
       OnFailure:
            Result: ROLLBACK
    ArtifactStore:
      Type: S3
      Location:
        Ref: ArtifactStoreS3Location
      EncryptionKey:
        Id: arn:aws:kms:useast-1:ACCOUNT-ID:key/KEY-ID
        Type: KMS
    DisableInboundStageTransitions:
      -
        StageName: Release
        Reason: "Disabling the transition until integration tests are completed"
    Tags:
      - Key: Project
        Value: ProjectA
      - Key: IsContainerBased
        Value: 'true'
```

ステージロールバックの失敗条件の設定の詳細については、「CloudFormation ユーザーガイド」で「StageDeclaration」の「OnFailure」を参照してください。

実行リストでロールバックステータスを表示する

ロールバック実行のステータスとターゲット実行 ID を表示できます。

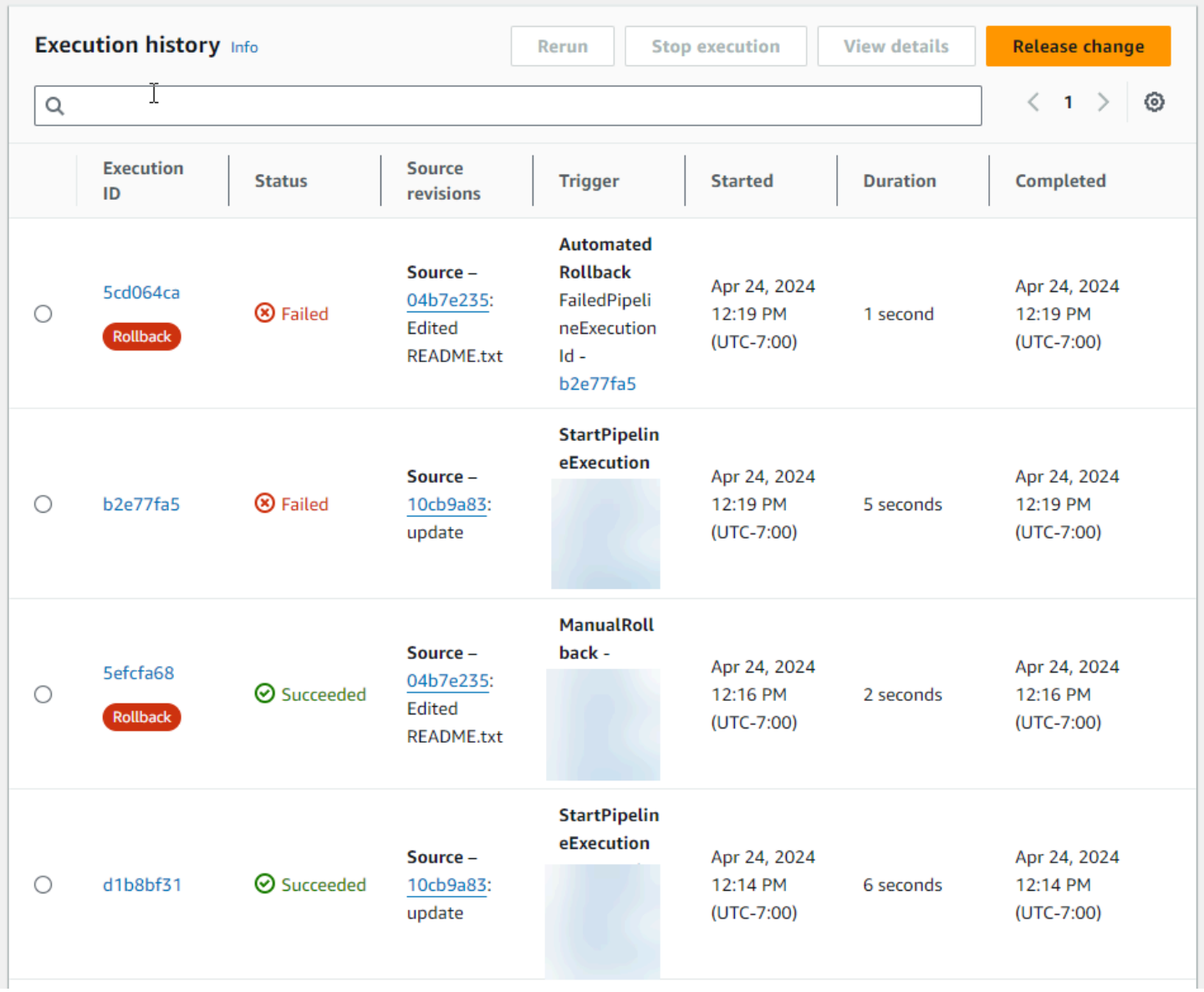
実行のリストでロールバックステータスを表示する (コンソール)

コンソールを使用して、実行リストでロールバック実行のステータスとターゲット実行 ID を表示できます。

実行のリストでロールバック実行ステータスを表示する (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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

   AWS アカウント に関連付けられているすべてのパイプラインの名前とステータスが表示されます。
2. [名前] で、表示するパイプラインの名前を選択します。
3. [履歴] を選択します。実行のリストに Rollback ラベルが表示されます。



詳細を表示する実行 ID を選択します。

`list-pipeline-executions` を使用してロールバックステータスを表示する (CLI)

CLI を使用して、ロールバック実行のステータスとターゲット実行 ID を表示できます。

- ターミナル (Linux、macOS、Unix) またはコマンドプロンプト (Windows) を開き、 AWS CLI を使用して list-pipeline-executions のコマンドを実行します。

  ```
  aws codepipeline list-pipeline-executions --pipeline-name MyFirstPipeline
  ```

  このコマンドは、パイプラインに関連付けられたすべての完了した実行のリストを返します。

次の例は、ロールバック実行によって開始された *MyFirstPipeline* という名前のパイプラインについて返されたデータを示しています。

```
{
    "pipelineExecutionSummaries": [
        {
            "pipelineExecutionId": "eb7ebd36-353a-4551-90dc-18ca5EXAMPLE",
            "status": "Succeeded",
            "startTime": "2024-04-16T09:00:28.185000+00:00",
            "lastUpdateTime": "2024-04-16T09:00:29.665000+00:00",
            "sourceRevisions": [
                {
                    "actionName": "Source",
                    "revisionId": "revision_ID",
                    "revisionSummary": "Added README.txt",
                    "revisionUrl": "console-URL"
                }
            ],
            "trigger": {
                "triggerType": "ManualRollback",
                "triggerDetail": "{arn:aws:sts::<account_ID>:assumed-role/<role>"}"
            },
            "executionMode": "SUPERSEDED",
            "executionType": "ROLLBACK",
            "rollbackMetadata": {
                "rollbackTargetPipelineExecutionId":
 "f15b38f7-20bf-4c9e-94ed-2535eEXAMPLE"
            }
        },
        {
            "pipelineExecutionId": "fcd61d8b-4532-4384-9da1-2aca1EXAMPLE",
            "status": "Succeeded",
            "startTime": "2024-04-16T08:58:56.601000+00:00",
            "lastUpdateTime": "2024-04-16T08:59:04.274000+00:00",
            "sourceRevisions": [
                {
                    "actionName": "Source",
                    "revisionId": "revision_ID",
                    "revisionSummary": "Added README.txt",
                    "revisionUrl": "console_URL"
                }
            ],
```

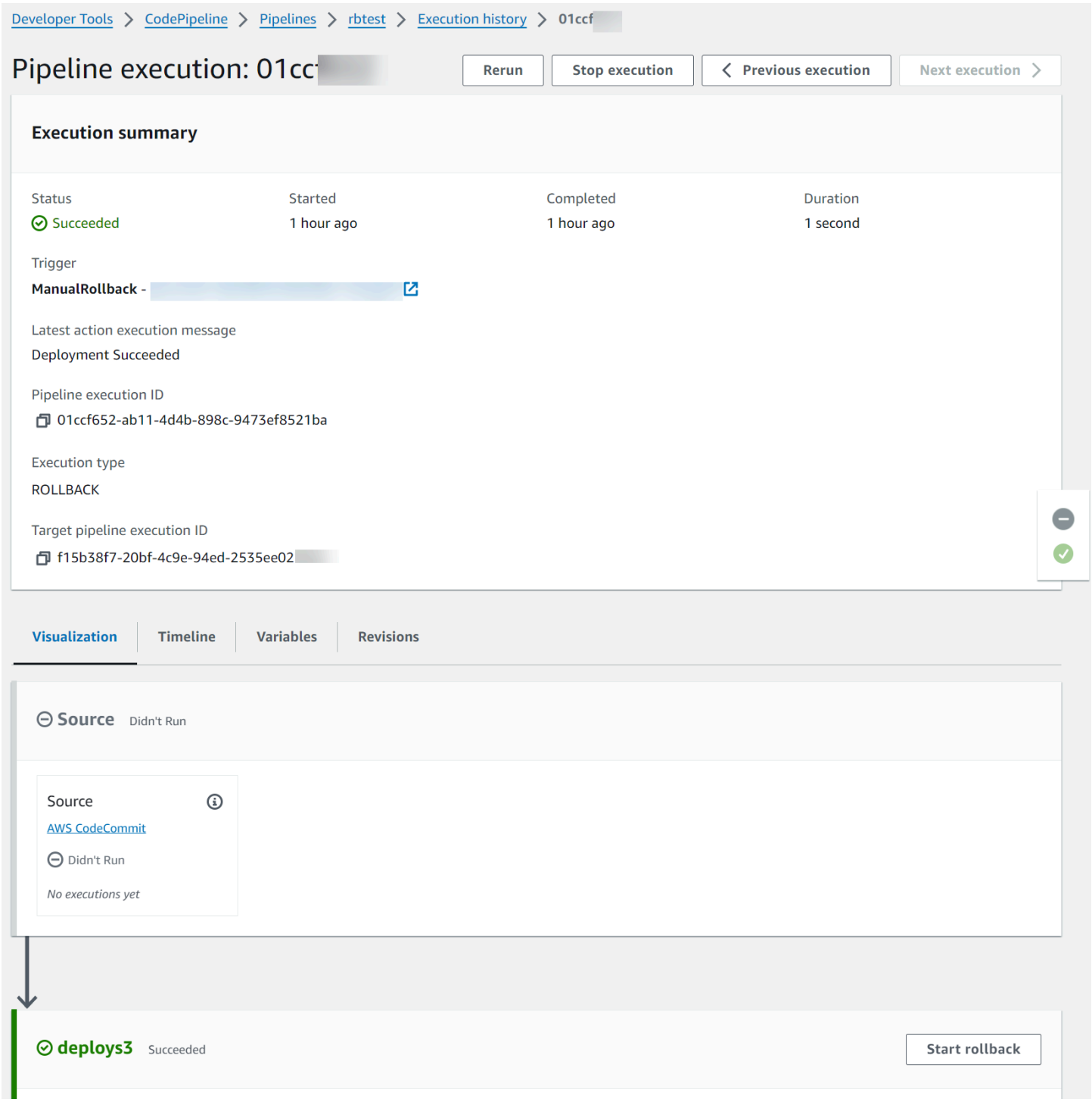
```
            "trigger": {
                "triggerType": "StartPipelineExecution",
                "triggerDetail": "arn:aws:sts::<account_ID>:assumed-role/<role>"
            },
            "executionMode": "SUPERSEDED"
        },
        {
            "pipelineExecutionId": "5cd064ca-bff7-425f-8653-f41d9EXAMPLE",
            "status": "Failed",
            "startTime": "2024-04-24T19:19:50.781000+00:00",
            "lastUpdateTime": "2024-04-24T19:19:52.119000+00:00",
            "sourceRevisions": [
                {
                    "actionName": "Source",
                    "revisionId": "<revision_ID>",
                    "revisionSummary": "Edited README.txt",
                    "revisionUrl": "<revision_URL>"
                }
            ],
            "trigger": {
                "triggerType": "AutomatedRollback",
                "triggerDetail": "{\"FailedPipelineExecutionId\":
\"b2e77fa5-9285-4dea-ae66-4389EXAMPLE\"}"
            },
            "executionMode": "SUPERSEDED",
            "executionType": "ROLLBACK",
            "rollbackMetadata": {
                "rollbackTargetPipelineExecutionId": "5efcfa68-d838-4ca7-
a63b-4a743EXAMPLE"
            }
         },
```

ロールバックステータスの詳細を表示する

ロールバック実行のステータスとターゲット実行 ID を表示できます。

詳細ページにロールバックステータスを表示する (コンソール)

コンソールを使用して、ロールバック実行のステータスとターゲットパイプライン実行 ID を表示できます。



get-pipeline-execution を使用してロールバックの詳細を表示する (CLI)

ロールバックされたパイプライン実行は、パイプライン実行を取得するための出力に表示されます。

- パイプラインの詳細を表示するには、パイプラインの一意の名前を指定して、get-pipeline-execution コマンドを実行します。例えば、*MyFirstPipeline* という名前のパイプラインの詳細を表示するには、以下のように入力します:

```
aws codepipeline get-pipeline-execution --pipeline-name MyFirstPipeline --pipeline-
execution-id 3f658bd1-69e6-4448-ba3e-79007EXAMPLE
```

このコマンドは、パイプラインの構造を返します。

次の例は、*MyFirstPipeline* という名前のパイプラインの一部について返されたデータを示しています。ロールバック実行 ID とメタデータが表示されています。

```
{
    "pipelineExecution": {
        "pipelineName": "MyFirstPipeline",
        "pipelineVersion": 6,
        "pipelineExecutionId": "2004a94e-8b46-4c34-a695-c8d20EXAMPLE",
        "status": "Succeeded",
        "artifactRevisions": [
            {
                "name": "SourceArtifact",
                "revisionId": "<ID>",
                "revisionSummary": "Added README.txt",
                "revisionUrl": "<console_URL>"
            }
        ],
        "trigger": {
            "triggerType": "ManualRollback",
            "triggerDetail": "arn:aws:sts::<account_ID>:assumed-role/<role>"
        },
        "executionMode": "SUPERSEDED",
        "executionType": "ROLLBACK",
        "rollbackMetadata": {
            "rollbackTargetPipelineExecutionId": "4f47bed9-6998-476c-a49d-
e60beEXAMPLE"
        }
    }
}
```

get-pipeline-state を使用してロールバック状態を表示する (CLI)

ロールバックされたパイプライン実行は、パイプライン状態を取得するための出力に表示されます。

- パイプラインの詳細を表示するには、パイプラインの一意の名前を指定して、get-pipeline-state コマンドを実行します。例えば、*MyFirstPipeline* という名前のパイプラインの状態の詳細を表示するには、次のように入力します。

```
aws codepipeline get-pipeline-state --name MyFirstPipeline
```

次の例は、ロールバック実行タイプで返されたデータを示しています。

```
{
    "pipelineName": "MyFirstPipeline",
    "pipelineVersion": 7,
    "stageStates": [
        {
            "stageName": "Source",
            "inboundExecutions": [],
            "inboundTransitionState": {
                "enabled": true
            },
            "actionStates": [
                {
                    "actionName": "Source",
                    "currentRevision": {
                        "revisionId": "<Revision_ID>"
                    },
                    "latestExecution": {
                        "actionExecutionId": "13bbd05d-
b439-4e35-9c7e-887cb789b126",
                        "status": "Succeeded",
                        "summary": "update",
                        "lastStatusChange": "2024-04-24T20:13:45.799000+00:00",
                        "externalExecutionId": "10cbEXAMPLEID"
                    },
                    "entityUrl": "console-url",
                    "revisionUrl": "console-url"
                }
            ],
            "latestExecution": {
                "pipelineExecutionId": "cf95a8ca-0819-4279-ae31-03978EXAMPLE",
```

```
                "status": "Succeeded"
            }
        },
        {
            "stageName": "deploys3",
            "inboundExecutions": [],
            "inboundTransitionState": {
                "enabled": true
            },
            "actionStates": [
                {
                    "actionName": "s3deploy",
                    "latestExecution": {
                        "actionExecutionId":
 "3bc4e3eb-75eb-45b9-8574-8599aEXAMPLE",
                        "status": "Succeeded",
                        "summary": "Deployment Succeeded",
                        "lastStatusChange": "2024-04-24T20:14:07.577000+00:00",
                        "externalExecutionId": "mybucket/SampleApp.zip"
                    },
                    "entityUrl": "console-URL"
                }
            ],
            "latestExecution": {
                "pipelineExecutionId": "fdf6b2ae-1472-4b00-9a83-1624eEXAMPLE",
                "status": "Succeeded",
                "type": "ROLLBACK"
            }
        }
    ],
    "created": "2024-04-15T21:29:01.635000+00:00",
    "updated": "2024-04-24T20:12:24.480000+00:00"
}
```

ステージの条件を設定する

パイプライン実行で特定の変数をチェックするなどのステージ条件を指定し、条件の結果としてステージをスキップしたり、ステージを失敗させたりすることができます。実行中のステージ条件をチェックするようにパイプラインを設定できます。これにより、ステージに対するチェックを指定し、特定の条件が満たされたときにステージをどのように進めるかを指定します。条件には、CodePipeline のルールのリストで使用できる 1 つ以上のルールが含まれます。条件内のすべてのルールが成功すると、条件は満たされます。条件が満たされない場合に、指定した結果が適用されるように、条件を設定できます。

各条件内には、一連の順序付けられたルールがあり、セットとしてまとめて評価されます。したがって、条件内の 1 つのルールが失敗すると、条件は失敗します。ルール条件は、パイプラインのランタイムに上書きできます。

条件は、特定の式タイプで使用します。条件ごとに利用可能な特定の結果として、次のようなオプションがあります。

- 入力 - チェックするための条件。条件を満たすと、ステージへの入力が許可されます。ルールに適用される結果のオプションは、失敗またはスキップです。
- 失敗時 - 失敗したときにステージをチェックするための条件。ルールに適用される結果のオプションは、ロールバックです。
- 成功時 - ステージが成功したときにステージをチェックするための条件。ルールに適用される結果のオプションは、ロールバックまたは失敗です。

条件は、条件のタイプごとに一連のルールでサポートされます。

条件のタイプごとに、条件によって設定された特定のアクションがあります。アクションは、成功または失敗した条件チェックの結果です。例えば、入力の条件 (入力条件) がアラーム (ルール) に遭遇すると、チェックは成功し、結果 (アクション) としてステージへの入力がブロックされます。

AWS CodePipeline コンソールまたは を使用して AWS CLI 、ステージまたはステージ内のアクションを手動でロールバックまたは再試行することもできます。「ステージの条件を設定する」を参照してください。

トピック

- ステージ条件のユースケース
- ステージ条件に設定する結果に関する考慮事項

- ステージ条件に設定するルールに関する考慮事項
- 入力条件の作成
- 失敗時の条件の作成
- 成功時の条件の作成
- ステージ条件の削除
- ステージ条件の上書き

ステージ条件のユースケース

ステージ条件には、パイプラインでリリースと変更の安全性を設定するための複数のユースケースがあります。ステージ条件のユースケースの例を以下に示します。

- 入力条件を使用して、CloudWatch アラーム状態をチェックする条件を定義します。これにより、本番環境が正常な状態でない場合、変更がブロックされます。
- 入力条件を使用して待機時間を 60 分とし、ステージ内のすべてのアクションが正常に完了したときに評価する条件を定義します。CloudWatch アラームが 60 分以内に ALARM 状態になった場合は変更をロールバックします。
- 成功時の条件を使用して条件を定義することにより、ステージが正常に完了した場合に、現在の時刻がデプロイウィンドウ内であるかどうかをルールで確認し、ルールが成功したら、デプロイするように設定します。

ステージ条件に設定する結果に関する考慮事項

ステージ条件に関する考慮事項は次のとおりです。

- OnFailure 条件では、ステージの自動再試行を使用できません。
- ロールバックを結果とする条件を設定する場合、ステージがロールバックする先の以前の実行が、現在のパイプライン構造バージョン内で利用可能な場合にのみ、ステージはロールバックできます。
- ロールバックを結果とする条件を設定する場合、ロールバック実行タイプであるターゲット実行 ID にロールバックすることはできません。
- 条件が失敗した場合に結果としてスキップを使用してステージをスキップする入力条件の場合、サポートされるルールは `LambdaInvoke` と `VariableCheck` のみです。
- [スキップ済み] ステータスのステージに対しては、ステージの手動再試行を実行できません。

- [スキップ済み] ステータスのステージに対しては、手動ロールバックを実行できません。
- 条件に結果としてスキップが設定されている場合、条件を上書きすることはできません。
- スキップの結果を除き、パイプライン実行を開始するときにステージ条件を上書きできます。上書きが適用されるステージ条件の場合、実行は次の表に示すように実行されます。

タイプ	失敗時の条件に設定されている結果	ステージのステータス	上書き動作
入力	失敗	進行中	ステージは進行します。
入力	スキップ	スキップ済み	該当なし。
OnFailure	ロールバック	失敗	ステージは失敗しました。
OnSuccess	ロールバック	成功	ステージは進行します。
OnSuccess	失敗	失敗	ステージは進行します。

ステージ条件に設定するルールに関する考慮事項

ステージ条件で使用できるルールに関する考慮事項は以下のとおりです。

- `LambdaInvoke` ルールでは、まずルールで使用する Lambda 関数を設定する必要があります。ルールを設定するときに、Lambda 関数 ARN を指定できるよう準備しておきます。
- `CloudWatchAlarm` ルールの場合、まず、ルールで使用する CloudWatch Events イベントを設定する必要があります。ルールを設定するときに、イベント ARN を指定できるよう準備しておきます。

入力条件の作成

コンソールまたは CLI を使用して、ステージの入力条件を設定できます。条件ごとに対応するルールと結果を設定します。ロールバック結果では、以前の実行が現在のパイプライン構造バージョンで開始されている場合にのみ、パイプラインは以前の実行にロールバックできます。

以下の手順では、モニタールールを使用する入力条件の例を示します。

詳細については、「CodePipeline API ガイド」の「条件」、「RuleTypeId」、「RuleExecution」を参照してください。

入力条件の作成 - CloudWatchAlarm ルールの例 (コンソール)

ステージの入力条件は、条件が満たされたときにステージで実行するルールおよび結果と共に設定できます。

入力条件を設定する (コンソール)

1. リソースの作成やリソースに指定するルール (AWS CloudWatchAlarm など) の ARN の取得など、すべての前提条件を満たします。
2. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

   AWS アカウント に関連付けられているすべてのパイプラインの名前とステータスが表示されます。
3. [名前] で、編集するパイプラインの名前を選択します。
4. パイプライン詳細ページで、[編集] を選択します。
5. [編集] ページで、編集するアクションに関連する [ステージを編集する] を選択します。
6. [入力条件を追加] を選択します。[ステージ前の入力条件] カードに、この条件で使用できる [失敗] オプションが表示されます。
7. [ルールを追加] を選択し、次の操作を行います。

   a. [ルール名] に、ルールの名前を入力します。この例では、`MyAlarmRule` と入力します。

   b. [ルールプロバイダー] で、条件に追加する事前設定済みのルールプロバイダーを選択します。この例では、AWS [CloudWatchAlarm] を選択し、次の手順を実行します。

   c. [リージョン] で、条件のリージョンを選択するか、デフォルトのままにします。

   d. [アラーム名] で、ルールに使用する CloudWatch リソースを選択します。アカウントでリソースを既に作成している必要があります。

   e. (オプション) [待機時間] に、アラームの最初の評価時にアラームが ALARM 状態になった場合に CodePipeline が待機する時間を入力します。ルールの最初のチェック時にアラームが OK 状態の場合、ルールはすぐに成功します。

   f. (オプション) モニタリングする特定のアラーム状態を入力し、必要に応じてロールの ARN を入力します。

g. ステージの編集が完了したら、[完了] を選択します。パイプラインの編集ページで、[保存] を選択します。

8. 実行後、結果を確認します。

スキップ結果と VariableCheck ルールを使用した入力条件の作成 (コンソール)

ステージのエントリ条件を設定して、エントリ条件が満たされない場合にステージがスキップされるようにできます。条件が失敗した場合、結果はエンゲージし、ステージはスキップされます。ステージがスキップされると、ステージのステータスは [スキップ済み] となり、アクションのステータスは [実行しませんでした] となります。スキップ結果を使用したステージ条件に関する考慮事項については、「ステージ条件に設定する結果に関する考慮事項」を参照してください。

次の例では、変数チェックルールによって値が一致していないことが検出されて、ビルドステージはスキップされます。

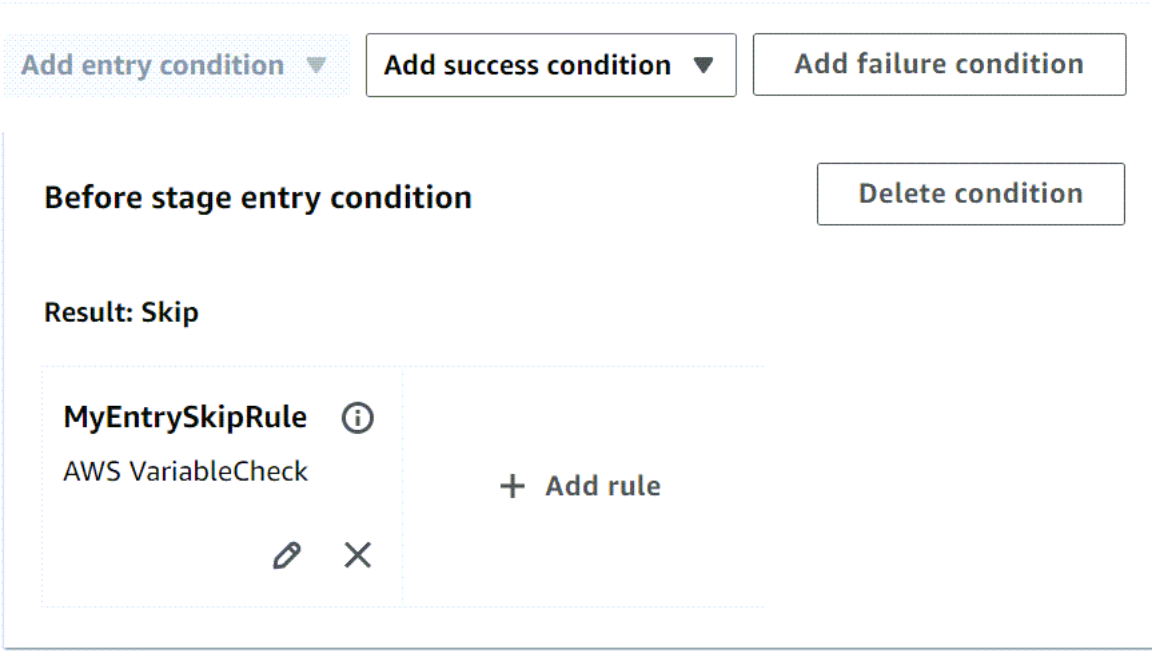
スキップ結果を使用して入力条件を設定する (コンソール)

1. リソースの作成やリソースに指定するルール (AWS CloudWatchAlarm など) の ARN の取得など、すべての前提条件を満たします。
2. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

   AWS アカウント に関連付けられているすべてのパイプラインの名前とステータスが表示されます。
3. [名前] で、編集するパイプラインの名前を選択します。
4. パイプライン詳細ページで、[編集] を選択します。
5. [編集] ページで、編集するアクションに関連する [ステージを編集する] を選択します。
6. [入力条件を追加] を選択し、[スキップ] を結果として選択します。
7. [ルールを追加] を選択し、次の操作を行います。

   a. [ルール名] に、ルールの名前を入力します。この例では、`MyAlarmRule` と入力します。

   b. [ルールプロバイダー] で、条件に追加する事前設定済みのルールプロバイダーを選択します。この例では、[VariableCheck] を選択し、次の手順を実行します。

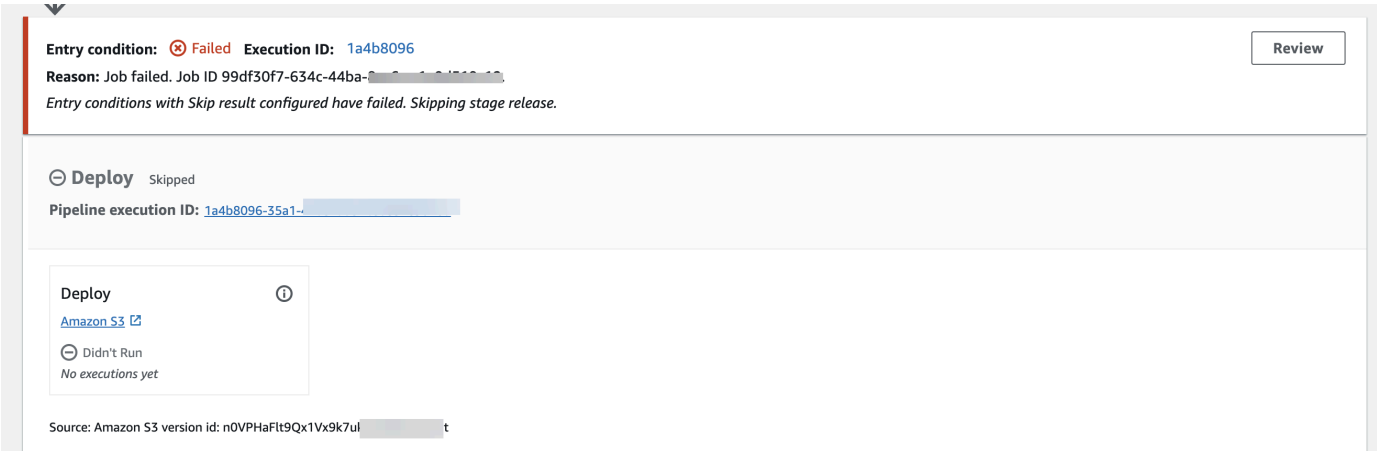


c. [リージョン] で、条件のリージョンを選択するか、デフォルトのままにします。

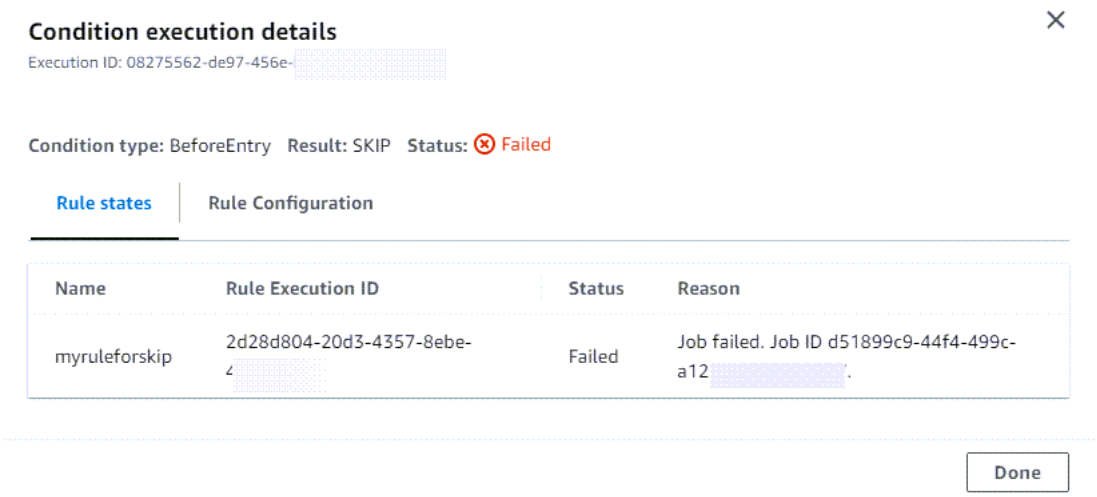
d. [変数] で、GitHub (GitHub アプリ経由) ソースアクションを持つパイプラインの `#{SourceVariables.FullRepositoryName}` など、比較する変数を選択します。リポジトリ名を入力し、[等しい] などの演算子を選択します。

e. ステージの編集が完了したら、[完了] を選択します。パイプラインの編集ページで、[保存] を選択します。

8. 実行後、結果を確認します。



9. 詳細を確認するには、[レビュー] を選択します。次の例の詳細は、条件の設定済みの結果が [スキップ] であり、上書きできないことを示しています。条件が満たされていないため、ルールのステータスは [失敗しました] になります。



入力条件の作成 (CLI)

を使用してエントリ条件 AWS CLI を設定するには、 コマンドを使用して、パイプライン、ステージ、アクションを作成する「」および「」で説明されているようにパイプラインを作成または更新しますCodePipeline でパイプラインを編集する。

条件とルールを設定する (CLI)

- ターミナル (Linux、macOS、Unix) またはコマンドプロンプト (Windows) を開きます。次に AWS CLI を使用し、パイプライン構造の失敗条件を指定して update-pipeline コマンドを実行します。次の例では、 という名前のステージのエントリ条件を設定しますDeploy。

```
{
    "name": "Deploy",
    "actions": [
        {
            "name": "Deploy",
            "actionTypeId": {
                "category": "Deploy",
                "owner": "AWS",
                "provider": "S3",
                "version": "1"
            },
            "runOrder": 1,
            "configuration": {
                "BucketName": "MyBucket",
                "Extract": "false",
```

```
                "ObjectKey": "object.xml"
            },
            "outputArtifacts": [],
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-east-1",
            "namespace": "DeployVariables"
        }
    ],
    "beforeEntry": {
        "conditions": [
            {
                "result": "FAIL",
                "rules": [
                    {
                        "name": "MyAlarmRule",
                        "ruleTypeId": {
                            "category": "Rule",
                            "owner": "AWS",
                            "provider": "CloudWatchAlarm",
                            "version": "1"
                        },
                        "configuration": {
                            "AlarmName": "CWAlarm",
                            "WaitTime": "1"
                        },
                        "inputArtifacts": [],
                        "region": "us-east-1"
                    }
                ]
            }
        ]
    }
}
```

ステージのロールバックの成功条件を設定する方法の詳細については、「CodePipeline API リファレンス」の「SuccessConditions」を参照してください。

入力条件の作成 (CFN)

CloudFormation を使用してエントリ条件を設定するには、 beforeEntryパラメータを使用します。入力時に、ステージはルールを実行し、結果を適用します。

```
beforeEntry:
     Result: FAIL
```

- 次のスニペットに示すように、テンプレートを更新します。次の例では、MyMonitorRule という名前のルールを使用して入力条件を設定します。

```
Name: Deploy
Actions:
- Name: Deploy
  ActionTypeId:
    Category: Deploy
    Owner: AWS
    Provider: S3
    Version: '1'
  RunOrder: 1
  Configuration:
    BucketName: MyBucket
    Extract: 'false'
    ObjectKey: object.xml
  OutputArtifacts: []
  InputArtifacts:
  - Name: SourceArtifact
  Region: us-east-1
  Namespace: DeployVariables
BeforeEntry:
  Conditions:
  - Result: FAIL
    Rules:
    - Name: MyMonitorRule
      RuleTypeId:
        Category: Rule
        Owner: AWS
        Provider: CloudWatchAlarm
        Version: '1'
      Configuration:
        AlarmName: CWAlarm
        WaitTime: '1'
```

```
      InputArtifacts: []
      Region: us-east-1
```

beforeEntry 条件の設定の詳細については、「CloudFormation ユーザーガイド」で「StageDeclaration」の「AWS::CodePipeline::Pipeline BeforeEntryConditions」を参照してください。

失敗時の条件の作成

ステージの失敗時の条件は、コンソールまたは CLI を使用して設定できます。条件ごとに対応するルールと結果を設定します。ロールバック結果では、以前の実行が現在のパイプライン構造バージョンで開始されている場合にのみ、パイプラインは以前の実行にロールバックできます。

失敗時の条件の作成 (コンソール)

ステージの失敗時の条件は、条件が満たされたときにステージで実行するルールおよび結果と共に設定できます。

失敗時の条件を設定する (コンソール)

1. リソースの作成やリソースに指定するルール (LambdaInvoke など) の ARN の取得など、すべての前提条件を満たします。
2. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

   AWS アカウント に関連付けられているすべてのパイプラインの名前とステータスが表示されます。
3. [名前] で、編集するパイプラインの名前を選択します。
4. パイプライン詳細ページで、[編集] を選択します。
5. [編集] ページで、編集するアクションに関連する [ステージを編集する] を選択します。
6. [失敗条件を追加] を選択します。[失敗条件] カードに、この条件で使用できる [ロールバック] オプションが表示されます。
7. [ルールを追加] を選択し、次の操作を行います。

   a. [ルール名] に、ルールの名前を入力します。この例では、MyLambdaRule と入力します。

   b. [ルールプロバイダー] で、条件に追加する事前設定済みのルールプロバイダーを選択します。この例では、[AWS LambdaInvoke] を選択し、次の手順を実行します。

c. [リージョン] で、条件のリージョンを選択するか、デフォルトのままにします。

d. [入力アーティファクト] で、ソースアーティファクトを選択します。

e. [関数名] で、ルールに使用する Lambda リソースを選択します。アカウントでリソースを既に作成している必要があります。

f. (オプション) [ユーザーパラメータ] で、追加設定のパラメータを表すペアを入力します。

g. (オプション) [ロールの ARN] で、ロールの ARN を入力します (設定済みである場合)。

h. (オプション) [タイムアウト (分)] に、タイムアウトするまでにルールが待機する時間を分単位で入力します。

i. ステージの編集が完了したら、[完了] を選択します。パイプラインの編集ページで、[保存] を選択します。

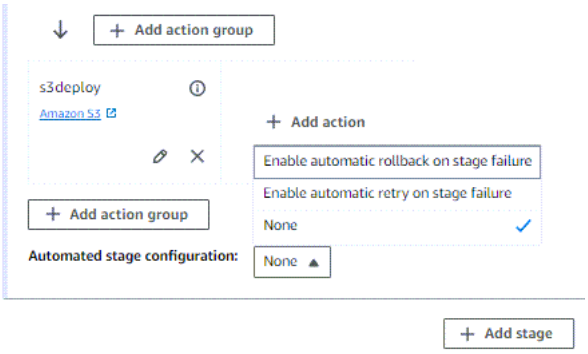
再試行結果の例を使用した OnFailure 条件の作成 (コンソール)

ステージの OnFailure 条件を設定して、入力条件が満たされない場合にステージを再試行できます。この結果の一環として、再試行モードを設定し、失敗したアクションを再試行するか、失敗したステージを再試行するかを指定します。

再試行結果を使用して OnFailure 条件を設定する (コンソール)

1. リソースの作成やリソースに指定するルール (AWS CloudWatchAlarm など) の ARN の取得など、すべての前提条件を満たします。
2. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

   AWS アカウント に関連付けられているすべてのパイプラインの名前とステータスが表示されます。
3. [名前] で、編集するパイプラインの名前を選択します。
4. パイプライン詳細ページで、[編集] を選択します。
5. [編集] ページで、編集するアクションに関連する [ステージを編集する] を選択します。
6. ステージの下部の [自動ステージ設定] で、[ステージ障害時の自動再試行を有効にする] を選択します。[再試行モード] で、[失敗したステージを再試行] または [失敗したアクションを再試行] を選択します。



7. OnFailure 条件を追加することを選択し、[ルールを追加] を選択して条件のルールを入力します。

   a. [ルール名] に、ルールの名前を入力します。この例では、`MyAlarmRule` と入力します。

   b. [ルールプロバイダー] で、条件に追加する事前設定済みのルールプロバイダーを選択します。この例では、[CloudWatchAlarm] を選択し、次の手順を実行します。

   c. [リージョン] で、条件のリージョンを選択するか、デフォルトのままにします。

   d. [アラーム名] で、アラートの設定済みリソースを選択します。

   e. ステージの編集が完了したら、[完了] を選択します。パイプラインの編集ページで、[保存] を選択します。

8. 実行後、結果を確認します。

失敗時の条件の作成 (CLI)

を使用して障害発生時の条件 AWS CLI を設定するには、 コマンドを使用して、パイプライン、ステージ、アクションを作成する「」および「」で説明されているパイプラインを作成または更新しますCodePipeline でパイプラインを編集する。

条件とルールを設定する (CLI)

- ターミナル (Linux、macOS、Unix) またはコマンドプロンプト (Windows) を開きます。次に AWS CLI を使用し、パイプライン構造の失敗条件を指定して `update-pipeline` コマンドを実行します。次の例では、 という名前のステージの障害発生時の条件を設定しますDeploy。

```
{
    "name": "Deploy",
    "actions": [
        {
            "name": "Deploy",
```

```
            "actionTypeId": {
                "category": "Deploy",
                "owner": "AWS",
                "provider": "S3",
                "version": "1"
            },
            "runOrder": 1,
            "configuration": {
                "BucketName": "MyBucket",
                "Extract": "false",
                "ObjectKey": "object.xml"
            },
            "outputArtifacts": [],
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-east-1",
            "namespace": "DeployVariables"
        }
    ],
    "onFailure": {
        "conditions": [
            {
                "result": "ROLLBACK",
                "rules": [
                    {
                        "name": "MyLambdaRule",
                        "ruleTypeId": {
                            "category": "Rule",
                            "owner": "AWS",
                            "provider": "LambdaInvoke",
                            "version": "1"
                        },
                        "configuration": {
                            "FunctionName": "my-function"
                        },
                        "inputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "region": "us-east-1"
```

```
                    }
                ]
            }
        ]
    }
}
```

失敗条件の設定の詳細については、「CodePipeline API リファレンス」の「FailureConditions」を参照してください。

失敗時の条件の作成 (CFN)

CloudFormation を使用して障害発生時の条件を設定するには、 `OnFailure`パラメータを使用します。失敗すると、ステージはルールを実行し、結果を実行します。

```
OnFailure:
     Result: ROLLBACK
```

- 次のスニペットに示すように、テンプレートを更新します。次の例では、`MyMonitorRule` という名前のルールを使用して OnFailure 条件を設定します。

```
name: Deploy
actions:
- name: Deploy
  actionTypeId:
    category: Deploy
    owner: AWS
    provider: S3
    version: '1'
  runOrder: 1
  configuration:
    BucketName: MyBucket
    Extract: 'false'
    ObjectKey: object.xml
  outputArtifacts: []
  inputArtifacts:
  - name: SourceArtifact
  region: us-east-1
  namespace: DeployVariables
OnFailure:
  conditions:
```

```
  - result: ROLLBACK
    rules:
    - name: MyMonitorRule
      ruleTypeId:
        category: Rule
        owner: AWS
        provider: CloudWatchAlarm
        version: '1'
      configuration:
        AlarmName: AlarmOnHelloWorldInvocation
        AlarmStates: ALARM
        WaitTime: '1'
      inputArtifacts: []
      region: us-east-1
```

失敗条件の設定の詳細については、「CloudFormation ユーザーガイド」で「StageDeclaration」の「OnFailure」を参照してください。

成功時の条件の作成

ステージの成功時の条件は、コンソールまたは CLI を使用して設定できます。条件ごとに対応するルールと結果を設定します。ロールバック結果では、以前の実行が現在のパイプライン構造バージョンで開始されている場合にのみ、パイプラインは以前の実行にロールバックできます。

以下の手順では、デプロイウィンドウルールを使用する成功時の条件の例を示します。

詳細については、「CodePipeline API ガイド」の「条件」、「RuleTypeId」、「RuleExecution」を参照してください。

成功時の条件の作成 (コンソール)

ステージの成功時の条件は、条件が満たされたときにステージで実行するルールおよび結果と共に設定できます。

成功時の条件を設定する (コンソール)

1. AWS LambdaRule など、リソースが提供されているルールのリソースと ARN の作成などの前提条件を完了します。
2. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

AWS アカウント に関連付けられているすべてのパイプラインの名前とステータスが表示されます。

3. [名前] で、編集するパイプラインの名前を選択します。
4. パイプライン詳細ページで、[編集] を選択します。
5. [編集] ページで、編集するアクションに関連する [ステージを編集する] を選択します。
6. [成功条件を追加] を選択します。[ステージの成功条件] カードが表示されます。この条件タイプの利用可能な結果として表示される [ロールバック] または [失敗] オプションを選択します。
7. [ルールを追加] を選択し、次の操作を行います。

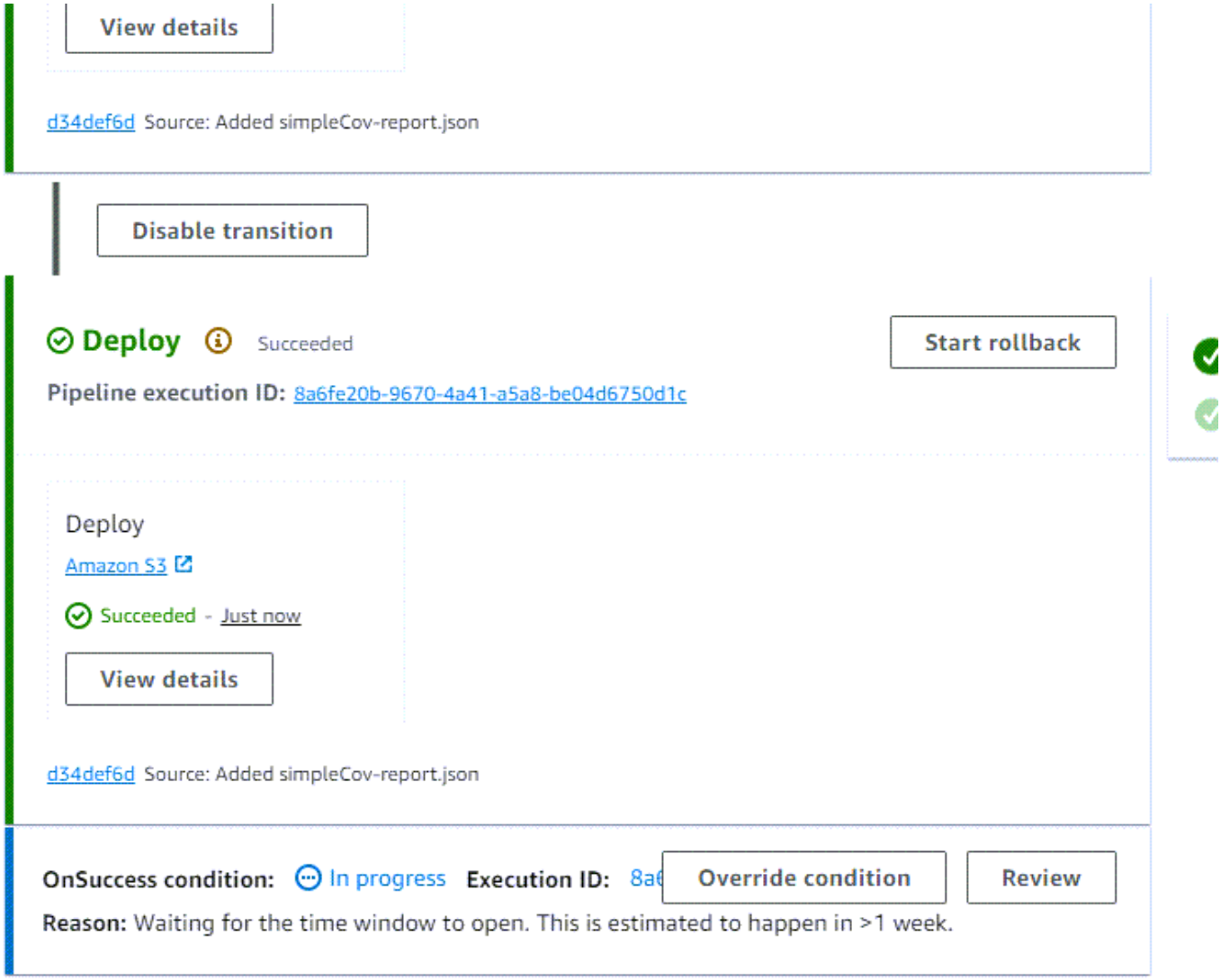
   a. [ルール名] に、条件の名前を入力します。この例では、`MyDeploymentRule` と入力します。

   b. [ルールプロバイダー] で、条件に追加する事前設定済みのルールを選択します。この例では、[AWS DeploymentWindow] を選択し、次の手順を実行します。

   c. [リージョン] で、条件のリージョンを選択するか、デフォルトのままにします。

   d. [cron] で、デプロイウィンドウの cron 式を入力します。cron 式は、デプロイを許可する曜日と時刻を定義します。cron 式のリファレンス情報については、「cron 式と rate 式を使用してルールをスケジュールする」を参照してください。

   e. (オプション) [タイムゾーン] で、デプロイウィンドウのタイムゾーンを入力します。
8. 実行後、結果を確認します。



成功時の条件の作成 (CLI)

を使用して成功時の条件 AWS CLI を設定するには、 コマンドを使用して、パイプライン、ステージ、アクションを作成する「」および「」で説明されているパイプラインを作成または更新しますCodePipeline でパイプラインを編集する。

条件とルールを設定する (CLI)

- ターミナル (Linux、macOS、Unix) またはコマンドプロンプト (Windows) を開きます。次に AWS CLI を使用し、パイプライン構造の失敗条件を指定して `update-pipeline` コマンドを実

行します。次の例では、 という名前のステージに成功時の条件を設定します。Deployルールの名前は ですMyDeploymentRule。

```
{
    "name": "Deploy",
    "actions": [
        {
            "name": "Deploy",
            "actionTypeId": {
                "category": "Deploy",
                "owner": "AWS",
                "provider": "S3",
                "version": "1"
            },
            "runOrder": 1,
            "configuration": {
                "BucketName": "MyBucket",
                "Extract": "false",
                "ObjectKey": "object.xml"
            },
            "outputArtifacts": [],
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-east-1",
            "namespace": "DeployVariables"
        }
    ],
    "onSuccess": {
       "conditions": [
            {
                "result": "FAIL",
                "rules": [
                    {
                        "name": "MyAlarmRule",
                        "ruleTypeId": {
                            "category": "Rule",
                            "owner": "AWS",
                            "provider": "CloudWatchAlarm",
                            "version": "1"
                        },
```

```
                        "configuration": {
                            "AlarmName": "CWAlarm",
                            "WaitTime": "1"
                        },
                        "inputArtifacts": [],
                        "region": "us-east-1"
                    }
                ]
            }
        ]
    }
}
```

成功条件の設定の詳細については、「CodePipeline API リファレンス」の「SuccessConditions」を参照してください。

成功時の条件を作成する (CFN)

CloudFormation を使用して成功時の条件を設定するには、 OnSuccessパラメータを使用します。成功時に、ステージはルールを実行し、結果を適用します。

```
OnSuccess:
     Result: ROLLBACK
```

- 次のスニペットに示すように、テンプレートを更新します。次の例では、MyDeploymentWindowRule という名前のルールを使用して OnSuccess 条件を設定します。

```
name: Deploy
actions:
- name: Deploy
  actionTypeId:
    category: Deploy
    owner: AWS
    provider: S3
    version: '1'
  runOrder: 1
  configuration:
    BucketName: MyBucket
    Extract: 'false'
    ObjectKey: object.xml
```

```
  outputArtifacts: []
  inputArtifacts:
  - name: SourceArtifact
  region: us-east-1
  namespace: DeployVariables
onSuccess:
   conditions:
  - result: FAIL
    rules:
    - name: MyMonitorRule
      ruleTypeId:
        category: Rule
        owner: AWS
        provider: CloudWatchAlarm
        version: '1'
      configuration:
        AlarmName: CWAlarm
        WaitTime: '1'
      inputArtifacts: []
      region: us-east-1
```

ステージロールバックの失敗条件の設定の詳細については、「CloudFormation ユーザーガイド」で「StageDeclaration」の「OnFailure」を参照してください。

ステージ条件の削除

パイプラインに設定したステージ条件を削除できます。

ステージ条件を削除するには

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

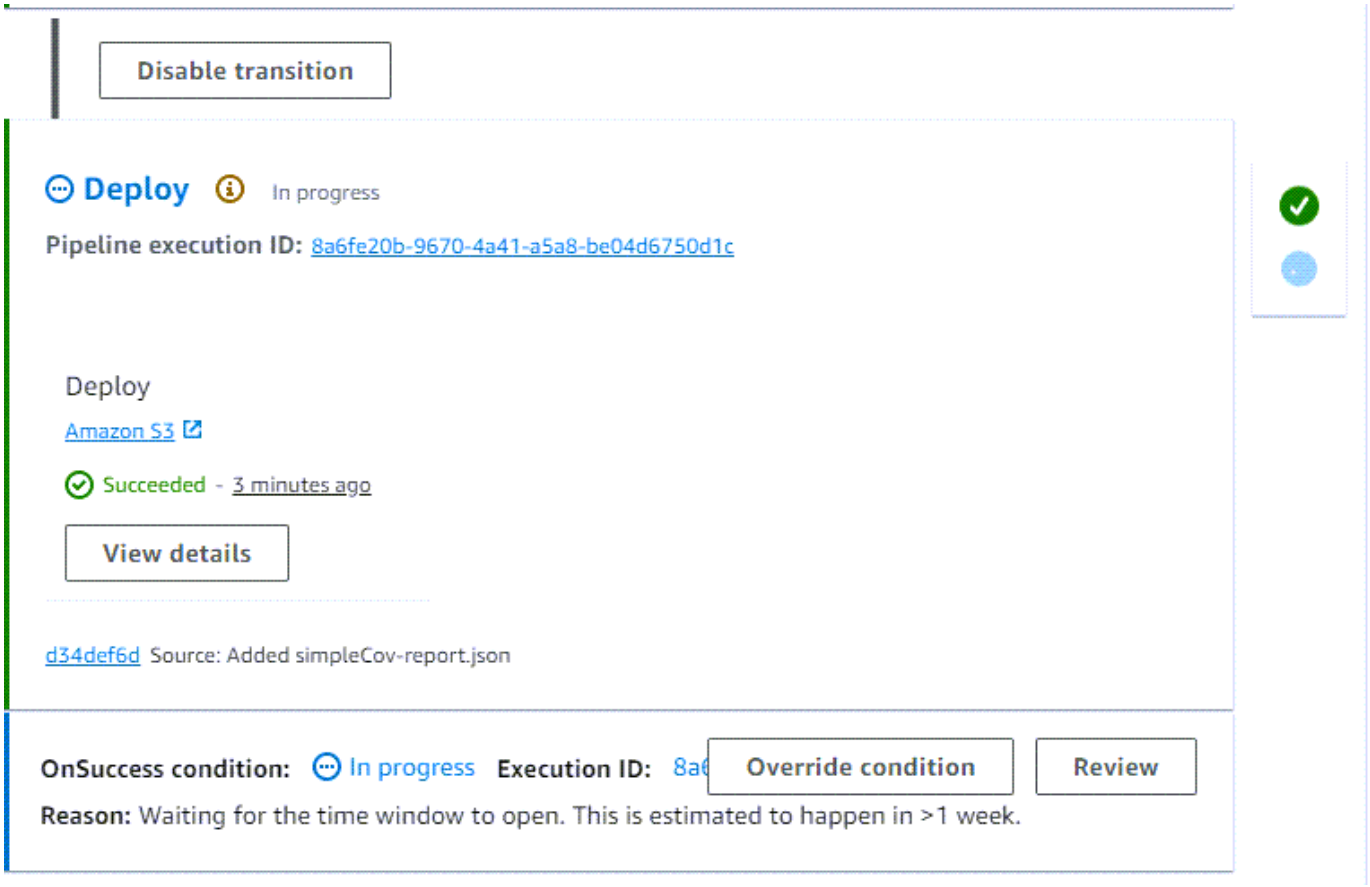
   に関連付けられているすべてのパイプラインの名前とステータス AWS アカウント が表示されます。
2. [名前] で、編集するパイプラインの名前を選択します。
3. パイプライン詳細ページで、[編集] を選択します。
4. [編集] ページで、編集する条件に関連する [ステージを編集する] を選択します。
5. 削除する条件の横にある、[条件を削除] を選択します。

ステージ条件の上書き

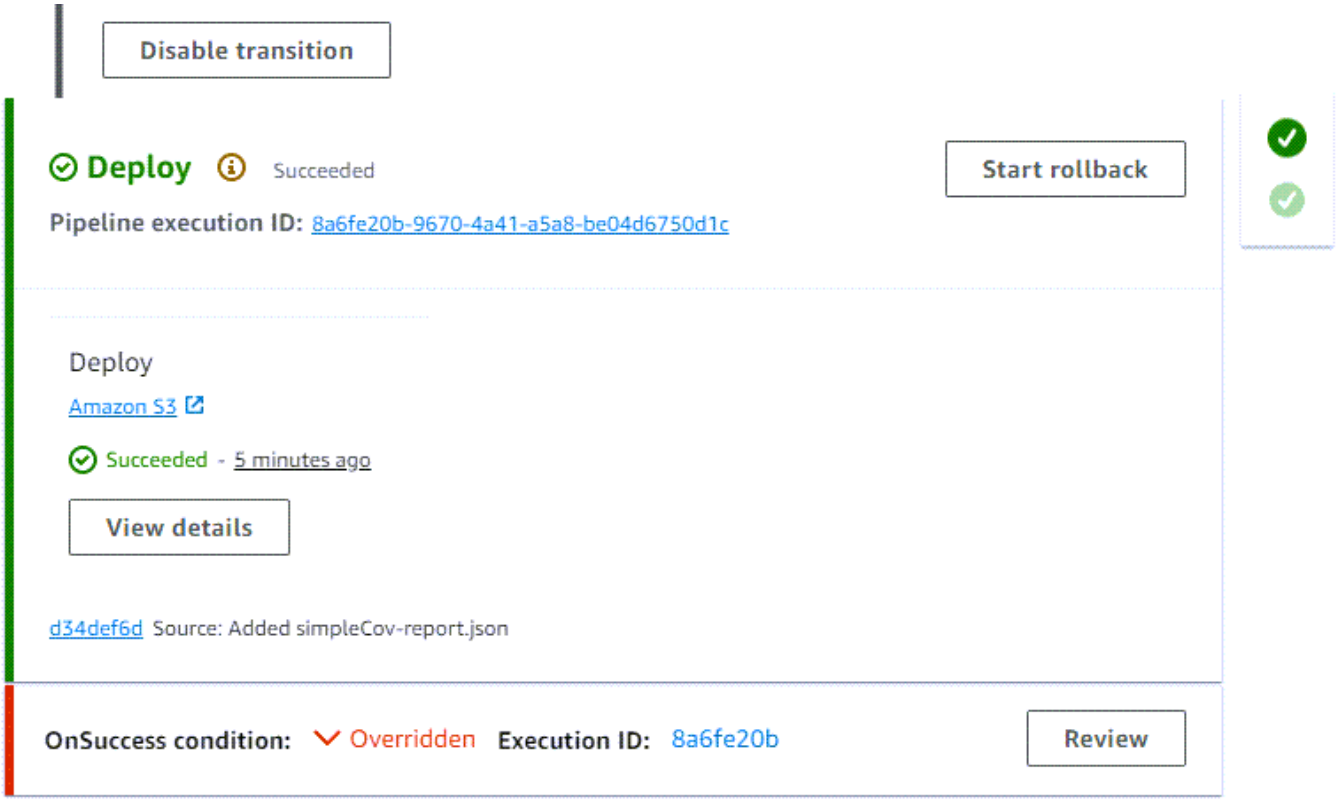
パイプラインに設定したステージ条件を上書きできます。コンソールで、ステージとルールの実行中に、ステージ条件を上書きすることを選択できます。これにより、ステージが実行されます。

ステージ条件を上書きするには

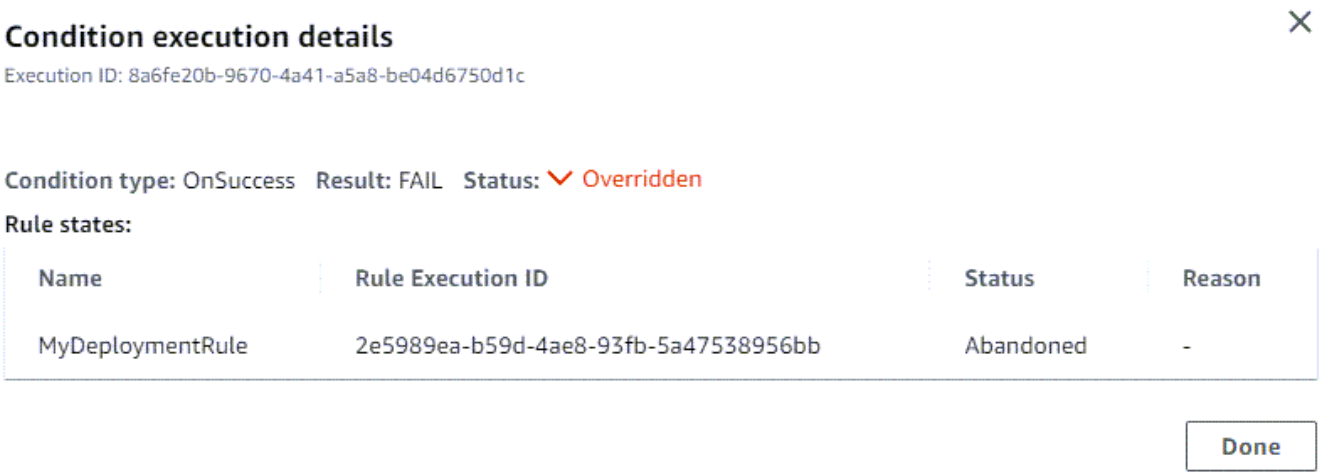
1. この例では、実行しているパイプラインステージに 1 つの条件が表示されています。[上書き] ボタンが有効になっています。



2. 上書きする条件の横にある [上書き] を選択します。



3. 詳細を確認するには、[レビュー] を選択します。次の例の詳細は、条件の設定済みの結果が [失敗] であり、上書きされたことを示しています。上書きにより、ルールステータスは [破棄済み] と表示されています。



アクションタイプ、カスタムアクション、および承認アクションを使用する

では AWS CodePipeline、アクションはパイプラインのステージのシーケンスの一部です。また、そのステージのアーティファクトで実行されるタスクです。パイプラインのアクションは、ステージ設定の定義に従い、指定された順序で順番に、または同時に実行されます。

CodePipeline では、6 種類のアクションをサポートしています。

- ソース
- ビルド
- テスト
- デプロイ
- 承認
- Invoke

アクションタイプに基づいてパイプラインに統合できる AWS のサービス およびパートナー製品およびサービスについては、「」を参照してくださいCodePipeline アクションタイプとの統合。

トピック

- アクションタイプの使用
- CodePipeline でカスタムアクションを作成および追加する
- CodePipeline でカスタムアクションにタグ付けする
- CodePipeline のパイプラインで AWS Lambda 関数を呼び出す
- 手動の承認アクションをステージに追加する
- CodePipeline でクロスリージョンアクションを追加する
- 変数の操作

アクションタイプの使用

アクションタイプは、プロバイダとして AWS CodePipelineでサポートされているインテグレーションモデルのいずれかを使用して顧客用に作成する、事前構成済みのアクションです。

アクションタイプをリクエスト、表示、および更新できます。所有者としてアカウントに対してアクションタイプが作成されている場合は、 AWS CLI を使用してアクションタイプのプロパティと構造を表示または更新できます。アクションタイプのプロバイダーまたは所有者である場合、顧客はアクションを選択し、CodePipeline で使用可能になった後にそのアクションをパイプラインに追加できます。

> **Note**
>
> `custom` フィールド内の `owner` でアクションを作成して、ジョブワーカーで実行します。インテグレーションモデルでは作成しません。カスタムアクションの詳細については、「CodePipeline でカスタムアクションを作成および追加する」を参照してください。

アクションタイプのコンポーネント

次のコンポーネントがアクションタイプを構成します。

- アクションタイプ ID - ID はカテゴリ、所有者、プロバイダー、およびバージョンで構成されます。次の例は、`ThirdParty` の所有者、`Test` のカテゴリ、`TestProvider` というプロバイダー、`1` のバージョンのアクションタイプ ID であることを示しています。

```
{
    "Category": "Test",
    "Owner": "ThirdParty",
    "Provider": "TestProvider",
    "Version": "1"
},
```

- 実行者設定 - アクションの作成時に指定されたインテグレーションモデルまたはアクションエンジン。アクションタイプの実行者を指定するときは、次の 2 つのタイプのいずれかを選択します。
  - Lambda: アクションタイプの所有者は、インテグレーションを Lambda 関数として書き込みます。Lambda 関数は、アクションで使用可能なジョブがあるたびに CodePipeline によって呼び出されます。
  - ジョブワーカー: アクションタイプの所有者は、カスタマーパイプラインで利用可能なジョブをポーリングするジョブワーカーとしてインテグレーションを書き込みます。その後ジョブワーカーはジョブを実行し、CodePipeline API を使用してジョブ結果を CodePipeline に送り返します。

> **Note**
>
> ジョブワーカーインテグレーションモデルは、推奨されるインテグレーションモデルではありません。

- 入力および出力アーティファクト: アクションタイプの所有者がアクションの顧客に対して指定するアーティファクトの制限。
- アクセス許可: サードパーティーのアクションタイプにアクセスできる顧客を指定するアクセス許可戦略。使用可能なアクセス許可戦略は、アクションタイプで選択したインテグレーションモデルによって異なります。
- URL: アクションタイプの所有者の設定ページなど、顧客が操作できるリソースへのディープリンク。

トピック

- アクションタイプをリクエストする
- 使用可能なアクションタイプをパイプラインに追加する (コンソール)
- アクションタイプを表示する
- アクションタイプを更新する

アクションタイプをリクエストする

サードパーティープロバイダーから新しい CodePipeline アクションタイプがリクエストされると、アクションタイプが CodePipeline でアクションタイプの所有者に対して作成され、所有者はアクションタイプを管理および表示できます。

アクションタイプは、プライベートアクションまたは公開アクションのいずれかです。アクションタイプの作成時、アクションタイプはプライベートになります。アクションタイプを公開アクションに変更するようリクエストするには、CodePipeline サービスチームにお問い合わせください。

CodePipeline チームのアクション定義ファイル、実行者リソース、およびアクションタイプのリクエストを作成する前に、インテグレーションモデルを選択する必要があります。

ステップ 1: インテグレーションモデルを選択する

インテグレーションモデルを選択し、そのモデルの構成を作成します。インテグレーションモデルを選択したら、インテグレーションリソースを構成する必要があります。

- Lambda インテグレーションモデルの場合、Lambda 関数を作成し、許可を追加します。インテグレータである Lambda 関数に許可を追加して、CodePipeline サービスプリンシパル `codepipeline.amazonaws.com` を使用して呼び出す許可を CodePipeline サービスに提供します。アクセス許可は、 CloudFormation またはコマンドラインを使用して追加できます。
  - CloudFormationを使用して許可を追加する例

```
  CodePipelineLambdaBasedActionPermission:
    Type: 'AWS::Lambda::Permission'
    Properties:
      Action: 'lambda:invokeFunction'
      FunctionName: {"Fn::Sub": "arn:${AWS::Partition}:lambda:${AWS::Region}:
${AWS::AccountId}:function:function-name"}
      Principal: codepipeline.amazonaws.com
```

  - コマンドラインのドキュメント
- ジョブワーカーインテグレーションモデルの場合、ジョブワーカーが CodePipeline API を使用してジョブをポーリングすることが許可されたアカウントのリストを使用してインテグレーションを作成します。

ステップ 2: アクションタイプ定義ファイルを作成する

JSON を使用して、アクションタイプ定義ファイルでアクションタイプを定義します。ファイルには、アクションカテゴリ、アクションタイプの管理に使用されるインテグレーションモデル、および構成プロパティが含まれます。

> **Note**
>
> パブリックアクションの作成後は、`properties` のアクションタイププロパティを `optional` から `required` へ変更することはできません。`owner` を変更することもできません。

アクションタイプ定義ファイルパラメータの詳細については、「ActionTypeDeclaration」および [CodePipeline API リファレンス] の「UpdateActionType」を参照してください。

アクションタイプ定義ファイルには 8 つのセクションがあります。

- description: 更新するアクションタイプの説明。
- executor: Lambda または job worker のサポートされているインテグレーションモデルで作成されたアクションタイプの実行者に関する情報。実行者タイプに基づき、jobWorkerExecutorConfiguration または lambdaExecutorConfiguration のどちらかのみを提供できます。
  - configuration: 選択したインテグレーションモデルに基づいたアクションタイプの構成のリソース。Lambda インテグレーションモデルの場合は、Lambda 関数 ARN を使用します。ジョブワーカーインテグレーションモデルの場合は、ジョブワーカーが実行されるアカウントまたはアカウントのリストを使用します。
  - jobTimeout: ジョブのタイムアウト (秒単位)。アクションの実行は、複数のジョブで構成できます。これは 1 つのジョブに対するタイムアウトであり、アクションの実行全体に対するタイムアウトではありません。

    > **Note**
    >
    > Lambda インテグレーションモデルの最大タイムアウトは 15 分です。

  - policyStatementsTemplate: アクションの実行を正常に実行するために必要な CodePipeline カスタマーアカウント内の許可を指定するポリシーステートメント。
  - type: Lambda または JobWorker のアクションタイプの作成と更新に使用されるインテグレーションモデル。
- id: アクションタイプのカテゴリ、所有者、プロバイダー、およびバージョン ID。
  - category: ソース、ビルド、デプロイ、テスト、呼び出し、承認のステージで実行できるアクションの種類。
  - provider: プロバイダ会社や製品名など、呼び出されるアクションタイプのプロバイダ。プロバイダ名は、アクションタイプの作成時に指定されます。
  - owner: AWS または ThirdParty の呼び出されているアクションの作成者。
  - version: アクションタイプのバージョン設定に使用する文字列。最初のバージョンでは、バージョン番号を 1 に設定します。
- inputArtifactDetails: パイプラインの前のステージから期待されるアーティファクトの数。
- outputArtifactDetails: アクションタイプステージの結果から期待されるアーティファクトの数。
- permissions: アクションタイプを使用する許可のあるアカウントを識別する詳細。

- properties: プロジェクトタスクを完了するために必要なパラメータ。
  - description: ユーザーに表示されるプロパティの説明。
  - optional: 設定プロパティがオプションであるかどうか。
  - noEcho: 顧客が入力したフィールド値をログから除外するかどうか。もし true の場合、GetPipeline API リクエストで返された時に値が編集されます。
  - key 設定プロパティがキーであるかどうか。
  - queryable: プロパティがポーリングで使用されるかどうか。アクションタイプには、1 つだけ問い合わせ可能なプロパティを設定できます。1 つ設定されている場合、そのプロパティは必須でなければならず、シークレットであってはなりません。
  - name: ユーザーに表示されるプロパティ名。
- urls: CodePipeline がユーザーに表示する URL のリスト。
  - entityUrlTemplate: 設定ページなど、アクションタイプの外部リソースへの URL。
  - executionUrlTemplate: アクションの最新の実行の詳細への URL。
  - revisionUrlTemplate: CodePipeline コンソールに表示される、顧客が外部アクションの設定を更新、または変更できるページへの URL。
  - thirdPartyConfigurationUrl: ユーザーが外部サービスにサインアップし、そのサービスによって提供されるアクションの初期設定を実行できるページへの URL。

次のコードは、アクションタイプ定義ファイルの例を示しています。

```
{
   "actionType": {
      "description": "string",
      "executor": {
         "configuration": {
            "jobWorkerExecutorConfiguration": {
               "pollingAccounts": [ "string" ],
               "pollingServicePrincipals": [ "string" ]
            },
            "lambdaExecutorConfiguration": {
               "lambdaFunctionArn": "string"
            }
         },
         "jobTimeout": number,
         "policyStatementsTemplate": "string",
         "type": "string"
      },
```

```
      "id": {
         "category": "string",
         "owner": "string",
         "provider": "string",
         "version": "string"
      },
      "inputArtifactDetails": {
         "maximumCount": number,
         "minimumCount": number
      },
      "outputArtifactDetails": {
         "maximumCount": number,
         "minimumCount": number
      },
      "permissions": {
         "allowedAccounts": [ "string" ]
      },
      "properties": [
         {
            "description": "string",
            "key": boolean,
            "name": "string",
            "noEcho": boolean,
            "optional": boolean,
            "queryable": boolean
         }
      ],
      "urls": {
         "configurationUrl": "string",
         "entityUrlTemplate": "string",
         "executionUrlTemplate": "string",
         "revisionUrlTemplate": "string"
      }
   }
}
```

ステップ 3: CodePipeline にインテグレーションを登録する

アクションタイプを CodePipeline に登録するには、CodePipeline サービスチームにお問い合わせください。

CodePipeline サービスチームにより、サービスコードベースを変更して、新しいアクションタイプのインテグレーションが登録されます。CodePipeline は 公開アクション と プライベートアクショ

ン の 2 つの新しいアクションを登録します。プライベートアクションをテストに使用し、準備ができたら、顧客トラフィックを処理する公開アクションを起動します。

Lambda インテグレーションのリクエストを登録するには

- 次のフォームを使用して CodePipeline サービスチームにリクエストを送信します。

```
This issue will track the onboarding of [Name] in CodePipeline.


[Contact engineer] will be the primary point of contact for this integration.

Name of the action type as you want it to appear to customers: Example.com Testing

Initial onboard checklist:

1. Attach an action type definition file in JSON format. This includes the schema
 for the action type

2. A list of test accounts for the allowlist which can access the new action type
 [{account, account_name}]

3. The Lambda function ARN

4. List of AWS ##### where your action will be available

5. Will this be available as a public action?
```

ジョブワーカーインテグレーションのリクエストを登録するには

- 次のフォームを使用して CodePipeline サービスチームにリクエストを送信します。

```
This issue will track the onboarding of [Name] in CodePipeline.

[Contact engineer] will be the primary point of contact for this integration.


Name of the action type as you want it to appear to customers: Example.com Testing

Initial onboard checklist:
```

```
1. Attach an action type definition file in JSON format. This includes the schema
 for the action type.

2. A list of test accounts for the allowlist which can access the new action type
 [{account, account_name}]

3. URL information:
Website URL: https://www.example.com/%TestThirdPartyName%/%TestVersionNumber%

Example URL pattern where customers will be able to review their configuration
 information for the action: https://www.example.com/%TestThirdPartyName%/
%customer-ID%/%CustomerActionConfiguration%

Example runtime URL pattern: https://www.example.com/%TestThirdPartyName%/
%customer-ID%/%TestRunId%

4. List of AWS ##### where your action will be available

5. Will this be available as a public action?
```

ステップ 4: 新しいインテグレーションを起動する

新しいインテグレーションを一般的に使用する準備ができたら、CodePipeline サービスチームにお問い合わせください。

使用可能なアクションタイプをパイプラインに追加する (コンソール)

アクションタイプをパイプラインに追加して、テストできるようにします。新しいパイプラインを作成するか、既存のパイプラインを編集することでこれが可能になります。

> **Note**
>
> アクションタイプがソース、ビルド、またはデプロイカテゴリのアクションの場合は、パイプラインを作成することで追加できます。アクションタイプがテストカテゴリにある場合は、既存のパイプラインを編集して追加する必要があります。

CodePipeline コンソールから既存のパイプラインにアクションタイプを追加するには

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

2. パイプラインのリストで、アクションタイプを追加したいパイプラインを選択します。
3. パイプラインの概要ビューページで、[編集] を選択します。
4. ステージの編集を選択します。アクションタイプを追加したいステージで、[Add action group (アクショングループの追加)] を選択します。[アクションの編集] ページが表示されます。
5. [アクションの編集] ページで、[Action name (アクション名)] にアクションの名前を入力します。これは、パイプラインのステージに表示される名前です。
6. [アクションプロバイダ] で、リストからアクションタイプを選択します。

   リスト内の値は、アクションタイプ定義ファイルで指定された `provider` に基づいています。
7. [入力アーティファクト] で、次の形式でアーティファクト名を入力します。

   `Artifactname``::`*`FileName`*

   許容される最小数量と最大数量は、アクションタイプ定義ファイルで指定される `inputArtifactDetails` に基づいて定義されます。
8. [<Action_Name> への接続] を選択します。

   ブラウザウィンドウが開き、アクションタイプ用に作成したウェブサイトに接続します。
9. 顧客としてウェブサイトにログインし、顧客がアクションタイプを使用するための手順を完了します。ステップはアクションカテゴリ、ウェブサイト、設定によって異なりますが、通常、アクションの編集ページに顧客を返す完了アクションが含まれます。
10. CodePipeline の [アクションの編集] ページで、アクションの追加設定フィールドが表示されます。表示されるフィールドは、アクション定義ファイルで指定した設定プロパティです。アクションタイプに合わせてカスタマイズされたフィールドに情報を入力します。

    例えば、アクション定義ファイルで `Host` という名前のプロパティが指定されている場合、ホスト のラベルが付いたフィールドが、お客様のアクション用に [アクションの編集] ページに表示されます。
11. [出力アーティファクト] で、次の形式でアーティファクト名を入力します。

    `Artifactname``::`*`FileName`*

    許容される最小数量と最大数量は、アクションタイプ定義ファイルで指定される `outputArtifactDetails` に基づいて定義されます。
12. [完了] を選択して、パイプラインの詳細ページに戻ります。

> **Note**
>
> 顧客は、オプションで CLI を使用してアクションタイプをパイプラインに追加できます。

13. アクションをテストするには、パイプラインのソースステージで指定されたソースに変更をコミットするか、[パイプラインを手動で開始する] の手順に従います。

アクションタイプでパイプラインを作成するには、パイプライン、ステージ、アクションを作成するのステップに従い、テストをするステージの数だけアクションタイプを選択します。

アクションタイプを表示する

CLI を使用して、アクションタイプを表示できます。get-action-type コマンドを使用して、インテグレーションモデルを使用して作成されたアクションタイプを表示します。

アクションタイプを表示するには

1. 入力 JSON ファイルを作成し、ファイルの名前を `file.json` にします。アクションタイプ ID を次の例に示すように JSON 形式で追加します。

```
{
    "category": "Test",
    "owner": "ThirdParty",
    "provider": "TestProvider",
    "version": "1"
}
```

2. ターミナルウィンドウまたはコマンドラインで、get-action-type コマンドを実行します。

```
aws codepipeline get-action-type --cli-input-json file://file.json
```

このコマンドは、アクションタイプのアクション定義の出力を返します。この例では、Lambda インテグレーションモデルで作成されたアクションタイプを表示します。

```
{
    "actionType": {
        "executor": {
            "configuration": {
```

```
                "lambdaExecutorConfiguration": {
                    "lambdaFunctionArn": "arn:aws:lambda:us-west-2:<account-
id>:function:my-function"
                }
            },
            "type": "Lambda"
        },
        "id": {
            "category": "Test",
            "owner": "ThirdParty",
            "provider": "TestProvider",
            "version": "1"
        },
        "inputArtifactDetails": {
            "minimumCount": 0,
            "maximumCount": 1
        },
        "outputArtifactDetails": {
            "minimumCount": 0,
            "maximumCount": 1
        },
        "permissions": {
            "allowedAccounts": [
                "<account-id>"
            ]
        },
        "properties": []
    }
}
```

アクションタイプを更新する

CLI を使用して、インテグレーションモデルで作成されたアクションタイプを編集できます。

公開アクションタイプの場合、所有者を更新することはできず、オプションのプロパティを必須事項に変更することはできません。また、新しいオプションプロパティのみを追加できます。

1. get-action-type コマンドを使用して、アクションタイプの構造を取得します。構造をコピーします。

2. 入力 JSON ファイルを作成し、ファイルの名前を action.json にします。前のステップでコピーしたアクションタイプ構造を、そこに貼り付けます。変更したいパラメータを更新します。オプションのパラメータを追加することもできます。

   入力ファイルのパラメータの詳細については、ステップ 2: アクションタイプ定義ファイルを作成する の「アクション定義ファイルの説明」を参照してください。

   次の例では、Lambda インテグレーションモデルで作成されたアクションタイプの例を更新する方法を表示します。この例では、以下の変更が発生します。

   - provider の名前を TestProvider1 に変更します。
   - 900 秒のジョブタイムアウト制限を追加します。
   - Host という名前のアクション設定プロパティを追加します。これは、アクションを使用する顧客に表示されます。

```
{
    "actionType": {
        "executor": {
            "configuration": {
                "lambdaExecutorConfiguration": {
                    "lambdaFunctionArn": "arn:aws:lambda:us-west-2:<account-
id>:function:my-function"
                }
            },
            "type": "Lambda",
            "jobTimeout": 900
        },
        "id": {
            "category": "Test",
            "owner": "ThirdParty",
            "provider": "TestProvider1",
            "version": "1"
        },
        "inputArtifactDetails": {
            "minimumCount": 0,
            "maximumCount": 1
        },
        "outputArtifactDetails": {
            "minimumCount": 0,
            "maximumCount": 1
        },
        "permissions": {
```

```
            "allowedAccounts": [
                "account-id"
            ]
        },
        "properties": {
         "description": "Owned build action parameter description",
         "optional": true,
         "noEcho": false,
         "key": true,
         "queryable": false,
         "name": "Host"
         }
    }
}
```

3. ターミナルまたはコマンドラインで、update-action-type コマンドを実行します。

```
aws codepipeline update-action-type --cli-input-json file://action.json
```

このコマンドは、更新されたパラメータに適合するアクションタイプの出力を返します。

CodePipeline でカスタムアクションを作成および追加する

AWS CodePipeline には、自動リリースプロセスのリソースのビルド、テスト、デプロイの設定に役立つアクションが多数含まれています。社内で開発したビルドプロセスやテストスイート等、デフォルトアクションに含まれていないアクティビティがリリースプロセスに含まれる場合、その目的のためにカスタムアクションを作成し、パイプラインに含めることができます。を使用して AWS CLI 、AWS アカウントに関連付けられたパイプラインにカスタムアクションを作成できます。

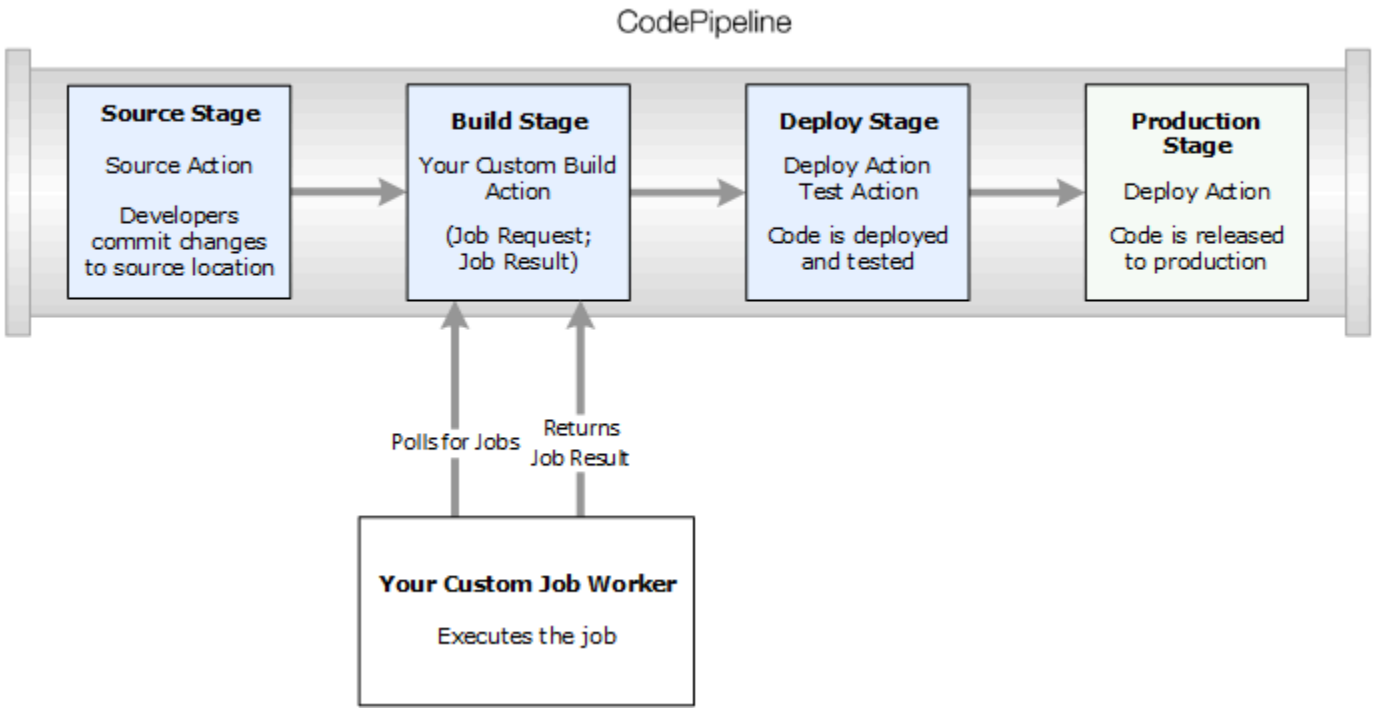
次のアクションカテゴリのカスタム AWS CodePipeline アクションを作成できます。

- 項目を構築または変換するカスタムビルドアクション
- 項目を 1 つ以上のサーバー、ウェブサイトまたはリポジトリにデプロイするカスタムデプロイアクション
- 自動テストを設定して実行するカスタムテストアクション
- 関数を実行するカスタム呼び出しアクション

カスタムアクションを作成する際、このカスタムアクションのジョブリクエストの CodePipeline をポーリングするジョブワーカーを作成し、ジョブを実行してステータス結果を CodePipeline に返す

必要があります。 このジョブワーカーは、CodePipeline のパブリックエンドポイントにアクセスできる限り、どのコンピュータまたはリソースにあっても構いません。簡単にアクセスおよびセキュリティを管理するために、ジョブワーカーを Amazon EC2 インスタンスにホストすることを考慮してください。

次の図では、カスタム構築アクションを含むパイプラインの高レベルビューを示します:



パイプラインにステージの一部としてカスタムアクションが含まれる場合、パイプラインはジョブリクエストを作成します。カスタムジョブワーカーはそのリクエストを検出し、そのジョブを実行します (この例では、サードパーティー構築ソフトウェアを使用するカスタムプロセス)。アクションが完了すると、ジョブワーカーは成功結果または失敗結果を返します。成功結果を受け取ると、パイプラインはリビジョンと次のアクションのアーティファクトを提供します。失敗が返された場合、パイプラインはリビジョンを次のアクションに渡しません。

> **Note**
>
> これらの手順では、すでにCodePipeline の使用開始のステップを完了していることを前提としています。

トピック

- カスタムアクションを作成する

- カスタムアクションのジョブワーカーを作成する
- パイプラインにカスタムアクションを追加する

カスタムアクションを作成する

を使用してカスタムアクションを作成するには AWS CLI

1. テキストエディタを開き、アクションカテゴリ、アクションプロバイダー、およびカスタムアクションで必要な設定を含む、カスタムアクションの JSON ファイルを作成します。例えば、1 つのプロパティのみを必要とするカスタムビルドアクションを作成する場合、JSON ファイルは次のようになります。

```
{
    "category": "Build",
    "provider": "My-Build-Provider-Name",
    "version": "1",
    "settings": {
        "entityUrlTemplate": "https://my-build-instance/job/{Config:ProjectName}/",
        "executionUrlTemplate": "https://my-build-instance/job/
{Config:ProjectName}/lastSuccessfulBuild/{ExternalExecutionId}/"
    },
    "configurationProperties": [{
        "name": "ProjectName",
        "required": true,
        "key": true,
        "secret": false,
        "queryable": false,
        "description": "The name of the build project must be provided when this
 action is added to the pipeline.",
        "type": "String"
    }],
    "inputArtifactDetails": {
        "maximumCount": integer,
        "minimumCount": integer
    },
    "outputArtifactDetails": {
        "maximumCount": integer,
        "minimumCount": integer
    },
    "tags": [{
      "key": "Project",
```

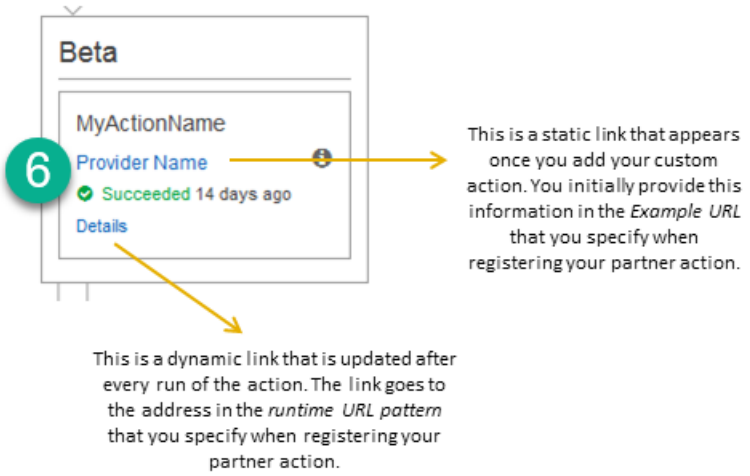
```
      "value": "ProjectA"
    }]
}
```

この例では、カスタムアクションで Project タグキーと ProjectA 値を含めることで、カスタムアクションにタグ付けを追加します。CodePipeline の リソースのタグ付けの詳細については、リソースのタグ付け を参照してください。

2 つのプロパティ entityUrlTemplate および executionUrlTemplate が JSON ファイルに含まれています。設定プロパティが必須で、かつシークレットではない限り、{Config:*name*} 形式に従って、URL テンプレート内のカスタムアクションの設定プロパティで名前を参照できます。例えば、上の例では、entityUrlTemplate の値は設定プロパティ *ProjectName* を参照します。

- entityUrlTemplate: アクションのサービスプロバイダーに関する情報を提供する静的リンク。この例では、ビルドシステムには、各ビルドプロジェクトへの静的リンクが含まれます。リンク形式は、ビルドプロバイダー (または、テストなど別のアクションタイプを作成する場合はその他のサービスプロバイダー) に応じて変わります。このリンク形式を指定し、カスタムアクションが追加されたときに、ユーザーはこのリンクを選択してブラウザを開き、ビルドプロジェクト (またはテスト環境) の詳細を提供するウェブサイト上のページに移動できるようにする必要があります。
- executionUrlTemplate: アクションの現在の実行または最新の実行に関する情報で更新される動的リンク。カスタムジョブワーカーがジョブのステータス (成功、失敗、進行中など) を更新するときに、リンクを完了するために使用される externalExecutionId も提供されます。このリンクを使用して、アクションの実行に関する詳細を提供できます。

例えば、パイプラインでアクションを表示すると、次の 2 つのリンクが表示されます。



1

この静的リンクは、カスタムアクションを追加し、`entityUrlTemplate` でアドレスを指した後で表示されます (カスタムアクションを作成するときに指定します)。

2

この動的なリンクは、アクションを実行し、`executionUrlTemplate` でアドレスを指すたびに更新されます (カスタムアクションを作成するときに指定します)。

これらのリンクタイプ、および `RevisionURLTemplate` と `ThirdPartyURL` の詳細については、「CodePipeline API リファレンス」の「ActionTypeSettings」および「CreateCustomActionType」を参照してください。アクション構造の要件とアクションの作成方法の詳細については、「CodePipeline パイプライン構造リファレンス」を参照してください。

2. JSON ファイルを保存し、簡単に覚えることができる名前 (例えば*`MyCustomAction`* など) を付けます。
3. AWS CLIをインストールしたコンピュータで、ターミナルセッション (Linux、OS X、Unix) またはコマンドプロンプト (Windows) を開きます。
4. AWS CLI を使用して aws codepipeline create-custom-action-type コマンドを実行し、先ほど作成した JSON ファイルの名前を指定します。

   例えば、ビルドカスタムアクションを作成するには以下のようにします。

> **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 codepipeline create-custom-action-type --cli-input-json
 file://MyCustomAction.json
```

5. このコマンドは、作成したカスタムアクションの構造全体、および追加された JobList アクション設定プロパティを返します。パイプラインにカスタムアクションを追加するときは、JobList を使用して、プロバイダーからのプロジェクトのうちジョブをポーリングできるものを指定できます。これを設定しない場合、カスタムジョブワーカーがジョブをポーリングするときに、使用可能なすべてのジョブが返されます。

   例えば、前のコマンドは、次のような構造を返します。

```
{
    "actionType": {
        "inputArtifactDetails": {
            "maximumCount": 1,
            "minimumCount": 1
       },
       "actionConfigurationProperties": [
            {
                "secret": false,
                "required": true,
                "name": "ProjectName",
                "key": true,
                "description": "The name of the build project must be provided when
 this action is added to the pipeline."
            }
        ],
        "outputArtifactDetails": {
            "maximumCount": 0,
            "minimumCount": 0
        },
        "id": {
            "category": "Build",
            "owner": "Custom",
            "version": "1",
```

```
            "provider": "My-Build-Provider-Name"
        },
        "settings": {
            "entityUrlTemplate": "https://my-build-instance/job/
{Config:ProjectName}/",
            "executionUrlTemplate": "https://my-build-instance/job/mybuildjob/
lastSuccessfulBuild/{ExternalExecutionId}/"
        }
    }
}
```

> **Note**
>
> create-custom-action-type コマンドの出力の一部として、 `id` セクションには `"owner": "Custom"` が含まれます。CodePipeline は、カスタムアクションタイプの所有者として自動的に `Custom` を割り当てます。create-custom-action-type コマンドまたは update-pipeline コマンドを使用する場合、この値を割り当てまたは変更することはできません。

カスタムアクションのジョブワーカーを作成する

カスタムアクションは、カスタムアクションのジョブリクエストに CodePipeline をポーリングし、ジョブを実行して、 CodePipeline にステータス結果を返すジョブワーカーが必要です。 ジョブワーカーは、 CodePipeline のパブリックエンドポイントにアクセスできる限り、どのコンピュータまたはリソースにあっても構いません。

ジョブワーカーを設計する方法は複数あります。次のセクションでは、 CodePipeline のカスタムジョブワーカーを開発するための、実践的なガイダンスを提供します。

トピック

- ジョブワーカー用にアクセス許可管理戦略を選択して設定する
- カスタムアクションのジョブワーカーを開発する
- カスタムジョブワーカーのアーキテクチャと例

ジョブワーカー用にアクセス許可管理戦略を選択して設定する

CodePipeline でカスタムアクションのカスタムジョブワーカーを開発するには、ユーザーやアクセス権限管理を統合するための戦略が必要になります。

最も簡単な戦略は、 IAM インスタンスロールで Amazon EC2 インスタンスを作成することでカスタムジョブワーカーに必要なインフラストラクチャを追加することです。これは、統合に必要なリソースを簡単にスケールアップすることを可能にします。との組み込み統合を使用して AWS 、カスタムジョブワーカーと CodePipeline 間のやり取りを簡素化できます。

Amazon EC2 インスタンスをセットアップする

1. Amazon EC2 の詳細を参照し、統合に適しているかどうかを判断します。詳細については、「Amazon EC2 - 仮想サーバーのホスティング」を参照してください。
2. Amazon EC2 インスタンスの作成を開始します。詳細については、「Amazon EC2 Linux インスタンスの開始方法」を参照してください。

他に考慮すべき戦略は、ID フェデレーションと IAM を使用した既存の ID プロバイダーシステムおよびリソースとの統合です。この戦略は、お客様がすでに企業 ID プロバイダーを持っているか、ウェブ ID プロバイダーを使用するユーザーをサポートできるよう設定されている場合に、特に便利です。ID フェデレーションを使用すると、IAM ユーザーを作成または管理することなく、CodePipeline などの AWS リソースへの安全なアクセスを許可できます。パスワードのセキュリティ要件や認証情報の更新に機能やポリシーを活用できます。サンプルアプリケーションをお客様自身の設計のテンプレートとして使用できます。

ID フェデレーションをセットアップするには

1. IAM 認証フェデレーションの詳細について学習します。詳細については、「フェデレーションの管理」を参照してください。
2. 「一時的なアクセス権を付与するシナリオ」の例を参照して、カスタムアクションのニーズに最適な一時アクセスのシナリオを確認します。
3. インフラストラクチャに関連する ID フェデレーションのコード例を確認します。例えば、以下の参照先をご覧ください。
   - Active Directory ユースケースのための ID フェデレーションのサンプルアプリケーション
4. ID フェデレーションの設定を開始します。詳細については、「IAM ユーザーガイド」の「ID プロバイダーとフェデレーション」を参照してください。

カスタムアクションとジョブワーカーを実行する AWS アカウント ときに、 で使用する次のいずれかを作成します。

ユーザーが の AWS 外部で を操作する場合は、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、、 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AWS「ローカル開発のための口グイン」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「ローカル開発用の口グイン」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。

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、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
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 とツールの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。

次は、カスタムジョブワーカーで使用するために作成する可能性があるポリシーの例です。このポリシーは例に過ぎず、そのまま提供されています。

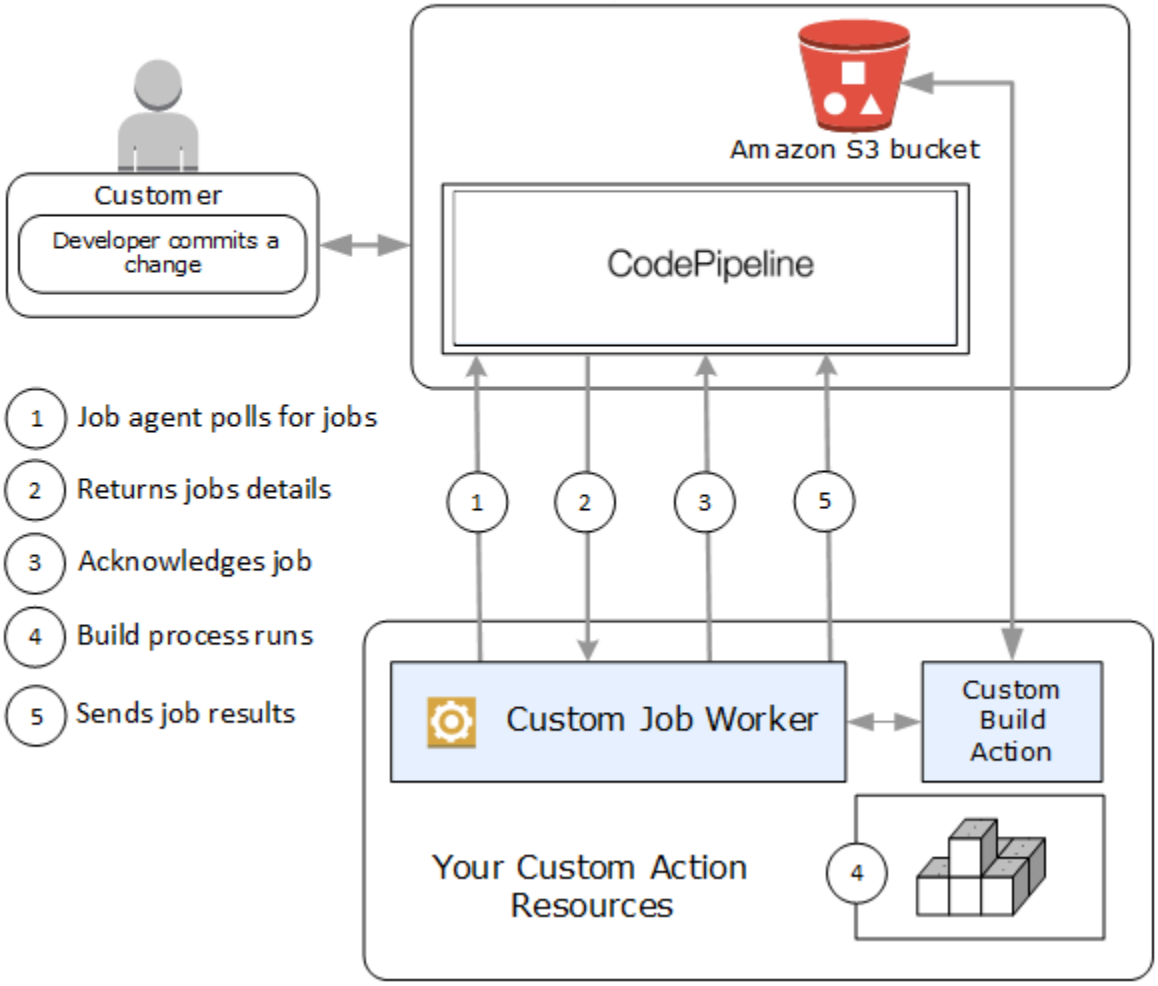
JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codepipeline:PollForJobs",
        "codepipeline:AcknowledgeJob",
        "codepipeline:GetJobDetails",
        "codepipeline:PutJobSuccessResult",
        "codepipeline:PutJobFailureResult"
      ],
      "Resource": [
        "arn:aws:codepipeline:us-east-2::actionType:custom/
Build/MyBuildProject/1/"
      ]
    }
  ]
}
```

> **Note**
>
> AWSCodePipelineCustomActionAccess マネージドポリシーの使用を検討してください。

カスタムアクションのジョブワーカーを開発する

アクセス権限管理戦略を選択した後、ジョブワーカーが CodePipeline とどう相互作用するかを考慮した方が良いでしょう。次の概要図は、ビルドプロセスのカスタムアクションおよびジョブワーカーのワークフローを示します。



1. ジョブワーカーは、PollForJobs を使用するジョブに CodePipeline をポーリングします。
2. パイプラインがソースステージでの変更によってトリガーされる際 (例えば、開発者が変更をコミットした際)、自動リリースプロセスが開始します。プロセスは、カスタムアクションが設定されたステージまで継続します。このステージのアクションに達すると、 CodePipeline はジョブをキューにいれます。このジョブは、ジョブワーカーがステータスを取得するために PollForJobs を再度呼び出すと、表示されます。PollForJobs からジョブ詳細を取得し、ジョブワーカーに渡します。
3. ジョブワーカーが AcknowledgeJob CodePipeline にジョブの確認応答を送信します。CodePipeline は、ジョブワーカーがジョブを継続中であることを示す確認 (InProgress) を返します。または、複数のジョブワーカーがジョブをポーリングしていて、別のジョブワーカーがジョブを登録済みである場合は、エラー InvalidNonceException が返されます。確認 InProgress の後、 CodePipeline　は結果が返されるまで待機します。

4. ジョブワーカーはリビジョンでカスタムアクションを開始し、その後にアクションが実行されます。他のアクションとともに、カスタムアクションは結果をジョブワーカーに返します。ビルドカスタムアクションの例では、アクションが Amazon S3 バケットからアーティファクトを引き出し、構築して、構築されたアーティファクトを Amazon S3 バケットに正常にプッシュします。
5. そのアクションが実行されている間、ジョブワーカーは、継続トークン (ジョブワーカーによって生成されたジョブの状態のシリアル化、例えば JSON 形式のビルド識別子や Amazon S3 オブジェクトキー) を使用して `PutJobSuccessResult` を呼び出すことができ、さらに `ExternalExecutionId` 情報 (`executionUrlTemplate` のリンクの入力に使用される) も呼び出すことができます。これは、進行中に特定のアクション詳細への有効なリンクとともにパイプラインのコンソールビューを更新します。必要ではありませんが、ユーザーがカスタムアクションの実行中にそのステータスを確認することを可能にするため、ベストプラクティスです。

   `PutJobSuccessResult` が呼び出されると、ジョブは完了したと見なされます。継続トークンが含まれる新しいジョブが CodePipeline に作成されます。このジョブは、ジョブワーカーが再度 `PollForJobs` を呼び出すと表示されます。この新しいジョブは、アクションの状態を確認するために使用でき、継続トークンを伴って返すか、アクションが完了すると、継続トークンを伴わずに返します。

   > **Note**
   >
   > ジョブワーカーがカスタムアクションの処理をすべて行っている場合、ジョブワーカーの処理を少なくとも 2 つのステップに分割することを考慮した方が良いでしょう。最初のステップでは、アクションの詳細ページを確立します。詳細ページを作成すると、ジョブワーカーの状態をシリアル化し、サイズ制限の対象である継続トークンとして返します。(AWS CodePipeline のクォータを参照してください)。例えば、継続トークンとして使用する文字列に、アクションの状態を書き込むことができます。ジョブワーカーの処理の 2 番目のステップ (およびその後のステップ) が、アクションの実際の作業を実行します。最終ステップは、最終ステップの継続トークンなしで成功または失敗を CodePipeline に返します。

   継続トークンの使用方法の詳細については、 `PutJobSuccessResult` の仕様のCodePipeline API を参照してください。
6. カスタムアクションが完了すると、ジョブワーカーは 2 つの内 1 つの API を呼び出すことでカスタムアクションの結果を CodePipeline に返します:

- カスタムアクションの実行が成功したことを示す、継続トークンなしの `PutJobSuccessResult` 。
- `PutJobFailureResult` のカスタムアクションが成功しなかったことを示す

結果によって、パイプラインは次のアクションに継続 (成功) するか、停止 (失敗) します。

カスタムジョブワーカーのアーキテクチャと例

高レベルワークフローを綿密に計画した後、ジョブワーカーを作成できます。最終的にはカスタムアクションの仕様がジョブワーカーに必要なものを決定しますが、カスタムアクションのジョブワーカーの多くは以下の機能を含みます:

- `PollForJobs` を使用して CodePipeline からジョブをポーリングする。
- ジョブを確認し、 CodePipeline に `AcknowledgeJob`、`PutJobSuccessResult` および `PutJobFailureResult`を使用して結果を返す。
- パイプラインの Amazon S3 バケットからアーティファクトを取得する、またはアーティファクトを配置する。Amazon S3 バケットからアーティファクトをダウンロードするには、署名バージョン 4 の署名 (Sig V4) を使用する Amazon S3 クライアントを作成する必要があります。Sig V4 は必須です AWS KMS。

  Amazon S3 バケットにアーティファクトをアップロードするには、さらに Amazon S3 リクエスト `PutObject` が暗号化を使用するよう設定する必要があります。現在、 暗号化では AWS Key Management Service (AWS KMS) のみがサポートされています。 AWS KMS を使用します AWS KMS keys。 AWS マネージドキー または カスタマーマネージドキーを使用してアーティファクトをアップロードするかどうかを知るには、カスタムジョブワーカーがジョブデータを調べ、暗号化キープロパティを確認する必要があります。プロパティが設定されている場合は、設定時にそのカスタマーマネージドキー ID を使用する必要があります AWS KMS。key プロパティが null の場合は、 AWS マネージドキーを使用します。CodePipeline は、特に設定 AWS マネージドキー されていない限り、 を使用します。

  Java または .NET で AWS KMS パラメータを作成する方法を示す例については、「SDK を使用して Amazon S3 AWS Key Management Service で を指定する AWS SDKs」を参照してください。CodePipeline 用の Amazon S3 バケットの詳細については、CodePipeline の概念 を参照してください。

カスタムジョブワーカーのさらに複雑な例が GitHub から入手可能です。このサンプルは、オープンソースコードであり、現状のまま提供されています。

- CodePipeline のサンプルジョブワーカー : GitHub リポジトリからサンプルをダウンロードしてください。

パイプラインにカスタムアクションを追加する

ジョブワーカーを作成したら、新しいアクションを作成してパイプラインの作成ウィザードの使用時に選択するか、既存のパイプラインを編集してカスタムアクションを追加するか AWS CLI、、、SDKs、または APIs を使用して、カスタムアクションをパイプラインに追加できます。

> **Note**
>
> ビルドまたはデプロイアクションであれば、パイプラインの作成ウィザードでカスタムアクションを含むパイプラインを作成できます。カスタムアクションがテスト カテゴリーにある場合は、既存のパイプラインを編集して追加する必要があります。

トピック

- 既存のパイプラインにカスタムアクションを追加する (CLI)

既存のパイプラインにカスタムアクションを追加する (CLI)

を使用して AWS CLI 、既存のパイプラインにカスタムアクションを追加できます。

1. ターミナルセッション (Linux、macOSまたはUnix) またはコマンドプロンプト (Windows) を開き、get-pipeline コマンドを実行して、編集するパイプライン構造を JSON ファイルにコピーします。例えば、**MyFirstPipeline** という名前のパイプラインの場合は、以下のコマンドを入力します。

   ```
   aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
   ```

   このコマンドは何も返しませんが、作成したファイルは、コマンドを実行したディレクトリにあります。

2. 任意のテキストエディタで JSON ファイルを開き、ファイルの構造を変更して、カスタムアクションを既存のステージに追加します。

> **Note**
>
> そのステージでアクションを別のアクションと並行して実行する場合は、そのアクションと同じ `runOrder` 値を割り当てます。

例えば、Build という名前のステージを追加し、パイプラインの構造を変更してそのステージにビルドカスタムアクションを追加する場合は次のように JSON を変更して、デプロイステージの前にビルドステージを追加します。

```
,
    {
      "name": "MyBuildStage",
      "actions":  [
             {
               "inputArtifacts": [
               {
                  "name": "MyApp"
                }
                  ],
                   "name": "MyBuildCustomAction",
                   "actionTypeId": {
                       "category": "Build",
                       "owner": "Custom",
                       "version": "1",
                       "provider": "My-Build-Provider-Name"
                   },
                   "outputArtifacts": [
                       {
                         "name": "MyBuiltApp"
                       }
                   ],
                   "configuration": {
                       "ProjectName": "MyBuildProject"
                   },
                   "runOrder": 1
               }
           ]
       },
       {
           "name": "Staging",
```

```
            "actions": [
                    {
                        "inputArtifacts": [
                            {
                                "name": "MyBuiltApp"
                            }
                        ],
                        "name": "Deploy-CodeDeploy-Application",
                        "actionTypeId": {
                            "category": "Deploy",
                            "owner": "AWS",
                            "version": "1",
                            "provider": "CodeDeploy"
                        },
                        "outputArtifacts": [],
                        "configuration": {
                            "ApplicationName": "CodePipelineDemoApplication",
                            "DeploymentGroupName": "CodePipelineDemoFleet"
                        },
                        "runOrder": 1
                    }
                ]
            }
    ]
}
```

3. 変更を適用するには、以下のように、パイプライン JSON ファイルを指定して、 update-pipeline コマンドを実行します。

> **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 codepipeline update-pipeline --cli-input-json file://pipeline.json
```

このコマンドは、編集したパイプラインの構造全体を返します。

4. CodePipeline コンソールを開き、編集したパイプラインの名前を選択します。

そのパイプラインには、行った変更が示されます。ソース場所を次に変更すると、修正した構造のパイプラインによりそのリビジョンが実行されます。

CodePipeline でカスタムアクションにタグ付けする

タグは、 AWS リソースに関連付けられたキーと値のペアです。コンソールまたは CLI を使用して、CodePipelineでカスタムアクションにタグを適用できます。CodePipeline リソースのタグ付け、使用例、タグのキーと値の制約、サポートされているリソースの種類については、リソースのタグ付け を参照してください。

カスタムアクションのタグの値を追加、削除、更新することができます。各カスタムアクションには、最大 50 個のタグを追加できます。

トピック

- カスタムアクションにタグを追加する
- カスタムアクションのタグを表示する
- カスタムアクションのタグを編集する
- カスタムアクションからタグを削除する

カスタムアクションにタグを追加する

を使用してカスタムアクションにタグ AWS CLI を追加するには、次の手順に従います。作成時にカスタムアクションにタグを追加するには、「CodePipeline でカスタムアクションを作成および追加する」を参照してください。

以下のステップでは、 AWS CLI の最新版を既にインストールしているか、最新版に更新しているものと想定します。詳細については、「AWS Command Line Interfaceのインストール」を参照してください。

ターミナルまたはコマンドラインで、タグを追加するカスタムアクションの Amazon リソースネーム (ARN)、および追加するタグのキーと値を指定して tag-resource コマンドを実行します。1 つのカスタムアクションに複数のタグを追加できます。例えば、2 つのタグがあるカスタムアクションにタグ付けするには、*TestActionType* というタグキーで *UnitTest* というタグ値を、また *ApplicationName* というタグキーで *MyApplication* というタグ値を指定します。

```
aws codepipeline tag-resource --resource-arn arn:aws:codepipeline:us-west-2:account-
id:actiontype:Owner/Category/Provider/Version --tags key=TestActionType,value=UnitTest
 key=ApplicationName,value=MyApplication
```

成功した場合、このコマンドは何も返しません。

カスタムアクションのタグを表示する

を使用してカスタムアクションの AWS タグ AWS CLI を表示するには、次の手順に従います。タグが追加されていない場合、返されるリストは空になります。

ターミナルまたはコマンドラインで、list-tags-for-resource コマンドを実行します。たとえば、ARN arn:aws:codepipeline:*us-west-2*:*account-id*:actiontype:*Owner*/*Category*/*Provider*/*Version* を持つカスタムアクションのタグキーとタグ値のリストを表示するには、次のように入力します。

```
aws codepipeline list-tags-for-resource --resource-arn arn:aws:codepipeline:us-
west-2:account-id:actiontype:Owner/Category/Provider/Version
```

成功した場合、このコマンドは次のような情報を返します。

```
{
    "tags": {
        "TestActionType": "UnitTest",
        "ApplicationName": "MyApplication"
    }
}
```

カスタムアクションのタグを編集する

を使用してカスタムアクションのタグ AWS CLI を編集するには、次の手順に従います。既存のキーの値を変更したり、別のキーを追加できます。次のセクションに示すように、カスタムアクションからタグを削除することもできます。

端末またはコマンドラインで、tag-resource コマンドを実行して、タグを更新するカスタムアクションの Amazon リソースネーム (ARN) を指定し、タグキーとタグ値を指定します。

```
aws codepipeline tag-resource --resource-arn arn:aws:codepipeline:us-
west-2:account-id:actiontype:Owner/Category/Provider/Version --tags
 key=TestActionType,value=IntegrationTest
```

カスタムアクションからタグを削除する

を使用してカスタムアクションからタグ AWS CLI を削除するには、次の手順に従います。関連付けられているリソースからタグを削除すると、そのタグが削除されます。

> **Note**
>
> カスタムアクションを削除すると、削除されたカスタムアクションからすべてのタグの関連付けが削除されます。カスタムアクションを削除する前にタグを削除する必要はありません。

ターミナルまたはコマンド行で、タグを削除するカスタムアクションの ARN と削除するタグのタグキーを指定して、untag-resource コマンドを実行します。たとえば、タグキー *TestActionType* があるカスタムアクションでタグを削除するには、次のように入力します。

```
aws codepipeline untag-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:actiontype:Owner/Category/Provider/Version --tag-keys TestActionType
```

成功した場合、このコマンドは何も返しません。カスタムアクションに関連付けられているタグを確認するには、list-tags-for-resource コマンドを実行します。

CodePipeline のパイプラインで AWS Lambda 関数を呼び出す

AWS Lambda はサーバーのプロビジョニングや管理をする必要がなく、コードを実行できるコンピューティングサービスです。Lambda 関数を作成し、アクションとしてパイプラインに追加できます。Lambda は、ほぼ全てのタスクを実行する関数の書き込みができるため、パイプラインの操作方法をカスタマイズできます。

> **Important**
>
> CodePipeline が Lambda に送信する JSON イベントをログに記録しないでください。これにより、CloudWatch Logs にユーザー認証情報が記録される可能性があるためです。CodePipeline ロールは JSON イベントを使用して、一時的な認証情報を `artifactCredentials` フィールドの Lambda に渡します。イベント例については、「JSON イベントの例」を参照してください。

パイプラインで Lambda 関数を使用する方法は次の通りです。

- を使用してパイプラインの 1 つのステージでオンデマンドでリソースを作成し CloudFormation 、別のステージでリソースを削除するには。

- CNAME 値をスワップする Lambda 関数 AWS Elastic Beanstalk を使用して、ダウンタイムのないアプリケーションバージョンを にデプロイします。
- Amazon ECS Docker インスタンスにデプロイするため。
- AMI スナップショットを作成することで、リソースをデプロイまたは作成する前にバックアップするため。
- IRC クライアントにメッセージを投稿する等、サードパーティー製品によってパイプラインに統合を追加するため。

> **Note**
>
> Lambda 関数を作成して実行すると、 AWS アカウントに料金が発生する可能性があります。詳細については、「料金」を参照してください。

このトピックでは、 AWS CodePipeline AWS Lambda と に精通しており、パイプライン、関数、およびそれらが依存する IAM ポリシーとロールを作成する方法を知っていることを前提としています。このトピックでは、以下の方法を示します。

- ウェブページが正常にデプロイされたかをテストする Lambda 関数を作成する。
- CodePipeline および Lambda 実行ロール、ならびにパイプラインの一部として関数を実行するために必要な権限を設定する。
- パイプラインを編集して Lambda 関数をアクションとして追加する。
- 手動で変更をリリースすることでアクションをテストする。

> **Note**
>
> CodePipeline でクロスリージョン Lambda 呼び出しアクションを使用する場合、PutJobSuccessResult と PutJobFailureResult を使用した Lambda 実行のステータスは、CodePipeline が存在する AWS リージョンではなく、Lambda 関数が存在するリージョンに送信する必要があります。

このトピックには、CodePipeline で Lambda 関数を使用する柔軟性を示すためのサンプル関数が含まれています。

- Basic Lambda function

- CodePipeline で使用する基本的な Lambda 関数を作成する。
- アクションの [詳細] リンクの CodePipeline に成功または失敗の結果を返す。

- AWS CloudFormation テンプレートを使用するサンプル Python 関数
  - 複数の設定値を関数に渡すために JSON でエンコードされたユーザーパラメータを使用する (`get_user_params`)。
  - アーティファクトバケットで、.zip アーティファクトと相互作用する (`get_template`)。
  - 長時間実行の非同期処理を監視する継続トークンを使用する (`continue_job_later`)。これにより、15 分のランタイム (Lambda の制限) を超えてもアクションを続行し、関数を成功させることができます。

各サンプル関数には、ロールに追加する必要がある権限についての情報が含まれます。の制限の詳細については AWS Lambda、「 AWS Lambda デベロッパーガイド」の「 の制限」を参照してください。

> **Important**
>
> このトピックに含まれるサンプルコード、ロール、およびポリシーは単なる例であり、現状のまま提供されます。

トピック

- ステップ 1: パイプラインを作成する
- ステップ 2 : Lambda 関数を作成する
- ステップ 3: CodePipeline コンソールでパイプラインに Lambda 関数を追加する
- ステップ 4 : Lambda 関数でパイプラインをテストする
- ステップ 5: 次のステップ
- JSON イベントの例
- 追加のサンプル関数

ステップ 1: パイプラインを作成する

このステップでは、後で Lambda 関数を追加するパイプラインを作成します。これは、「CodePipeline チュートリアル」で作成したものと同じパイプラインです。このパイプラインがま

だアカウントに設定されていて、Lambda 関数を作成するのと同じリージョンにある場合、このステップは省略できます。

パイプラインを作成するには

1. チュートリアル: シンプルなパイプラインを作成する (S3 バケット) の最初の 3 つのステップに従って、Amazon S3 バケット、CodeDeploy リソース、2 ステージパイプラインを作成します。インスタンスタイプに応じて Amazon Linux のオプションを選択します。パイプラインには任意の名前を使用できますが、このトピックの手順では MyLambdaTestPipeline を使用します。
2. パイプラインのステータスページの CodeDeploy アクション で、[詳細] を選択します。デプロイグループのデプロイの詳細ページで、リストからインスタンス ID を選択します。
3. Amazon EC2 コンソールのインスタンスの [詳細] タブで、[パブリック IPv4 アドレス] の IP アドレス (**192.0.2.4** など) をコピーします。このアドレスを AWS Lambdaで関数のターゲットとして使用します。

> **Note**
>
> CodePipeline のデフォルトサービスロールポリシーには、関数を呼び出すのに必要な Lambda アクセス権限が含まれています。ただし、デフォルトサービスロールを変更、または別のものを選択した場合、ロールのポリシーが `lambda:InvokeFunction` および `lambda:ListFunctions` 権限を許可していることを確認してください。そうしない場合、Lambda アクションを含むパイプラインは失敗します。

ステップ 2 : Lambda 関数を作成する

このステップでは、HTTP リクエストを生成し、ウェブページ上のテキスト行をチェックする Lambda 関数を作成します。このステップの一環として、IAM ポリシーおよび Lambda 実行ロールを作成する必要があります。詳細については、[AWS Lambda デベロッパーガイド]の [権限モデル」を参照してください。

実行ロールを作成するには

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/iam/ で IAM コンソールを開きます。
2. [Policies] を選択してから、[Create Policy] を選択します [JSON] タブを選択して、次のポリシーをフィールドに貼り付けます。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:*"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:logs:*:*:*"
    },
    {
      "Action": [
        "codepipeline:PutJobSuccessResult",
        "codepipeline:PutJobFailureResult"
        ],
        "Effect": "Allow",
        "Resource": "*"
     }
  ]
}
```

3. [ポリシーの確認] を選択します。
4. [ポリシーの確認] ページで、[名前] に、ポリシー名 (**CodePipelineLambdaExecPolicy** など) を入力します。[説明] で **Enables Lambda to execute code** を入力します。

   [ポリシーを作成] を選択します。

   > **Note**
   >
   > これらは、Lambda 関数が CodePipeline および Amazon CloudWatch とやり取りするために必要な最小限のアクセス権限です。このポリシーを拡張して、他の AWS リソースとやり取りする関数を許可する場合は、これらの Lambda 関数に必要なアクションを許可するようにこのポリシーを変更する必要があります。

5. ポリシーダッシュボードページで、[ロール]、[ロールの作成] の順に選択します。
6. [ロールの作成] ページで、[AWS のサービス] を選択します。[Lambda] を選択し、[Next: Permissions (次へ: アクセス許可)] を選択します。

7. [アクセス許可ポリシーをアタッチする] のページで、[CodePipelineLambdaExecPolicy] の横にあるチェックボックスを選択して [次へ: タグ] を選択します。[次へ: レビュー] を選択します。
8. [確認] ページの、[ロール名] で名前を入力し、[ロールの作成] を選択します。

CodePipeline で使用するサンプル Lambda 関数を作成するには

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/lambda/ で AWS Lambda コンソールを開きます。
2. [関数] ページで、[関数の作成] を選択します。

> **Note**
>
> [Lambda] のページの代わりに [Welcome] のページが表示された場合は、[今すぐ始める] を選択します。

3. [関数の作成] ページで、[一から作成] を選択します。[関数の名前] に、Lambda 関数の名前を入力します (例 : **MyLambdaFunctionForAWSCodePipeline**)。[ランタイム] で、[Node.js 20.x] を選択します。
4. [Role (ロール)] で、[既存のロールを選択] を選択します。[Existing role (既存のロール)] でロールを選択し、[Create function (関数の作成)] を選択します。

   作成した関数の詳細ページが開きます。
5. 次のコードを関数コードボックスに貼り付けます。

> **Note**
>
> CodePipeline.job キーの下にあるイベントオブジェクトには、ジョブの詳細が含まれます。CodePipeline が Lambda に返す JSON イベントの完全な例については、JSON イベントの例 を参照してください。

```
import { CodePipelineClient, PutJobSuccessResultCommand,
 PutJobFailureResultCommand } from "@aws-sdk/client-codepipeline";
import http from 'http';
import assert from 'assert';

export const handler = (event, context) => {
```

```
   const codepipeline = new CodePipelineClient();

   // Retrieve the Job ID from the Lambda action
   const jobId = event["CodePipeline.job"].id;

   // Retrieve the value of UserParameters from the Lambda action configuration in
CodePipeline, in this case a URL which will be
   // health checked by this function.
   const url =
event["CodePipeline.job"].data.actionConfiguration.configuration.UserParameters;

   // Notify CodePipeline of a successful job
   const putJobSuccess = async function(message) {
       const command = new PutJobSuccessResultCommand({
           jobId: jobId
       });
        try {
            await codepipeline.send(command);
            context.succeed(message);
        } catch (err) {
            context.fail(err);
        }
   };

   // Notify CodePipeline of a failed job
   const putJobFailure = async function(message) {
       const command = new PutJobFailureResultCommand({
           jobId: jobId,
           failureDetails: {
               message: JSON.stringify(message),
               type: 'JobFailed',
               externalExecutionId: context.awsRequestId
           }
       });
       await codepipeline.send(command);
       context.fail(message);
   };

   // Validate the URL passed in UserParameters
   if(!url || url.indexOf('http://') === -1) {
       putJobFailure('The UserParameters field must contain a valid URL address to
test, including http:// or https://');
       return;
```

```
    }

    // Helper function to make a HTTP GET request to the page.
    // The helper will test the response and succeed or fail the job accordingly
    const getPage = function(url, callback) {
        var pageObject = {
            body: '',
            statusCode: 0,
            contains: function(search) {
                return this.body.indexOf(search) > -1;
            }
        };
        http.get(url, function(response) {
            pageObject.body = '';
            pageObject.statusCode = response.statusCode;

            response.on('data', function (chunk) {
                pageObject.body += chunk;
            });

            response.on('end', function () {
                callback(pageObject);
            });

            response.resume();
        }).on('error', function(error) {
            // Fail the job if our request failed
            putJobFailure(error);
        });
    };

    getPage(url, function(returnedPage) {
        try {
            // Check if the HTTP response has a 200 status
            assert(returnedPage.statusCode === 200);
            // Check if the page contains the text "Congratulations"
            // You can change this to check for different text, or add other tests
as required
            assert(returnedPage.contains('Congratulations'));

            // Succeed the job
            putJobSuccess("Tests passed.");
        } catch (ex) {
            // If any of the assertions failed then fail the job
```

```
            putJobFailure(ex);
        }
    });
};
```

6. [Handler (ハンドラ)] はデフォルト値のままにし、[Role (ロール)] もデフォルトの **CodePipelineLambdaExecRole** のままにします。
7. 基本設定のタイムアウトに **20** 秒と入力します。
8. [保存] を選択します。

ステップ 3: CodePipeline コンソールでパイプラインに Lambda 関数を追加する

このステップでは、パイプラインに新しいステージを追加し、そのステージに関数を呼び出す Lambda アクションを追加します。

ステージを追加するには

1. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [Welcome (ようこそ)] ページで、作成したパイプラインを選択します。
3. パイプラインビューページで、[編集] を選択します。
4. [Edit (編集)] ページで、[+ Add stage (ステージの追加)] を選択し、CodeDeploy アクションでデプロイステージの後にステージを追加します。ステージの名前を入力し (たとえば、**LambdaStage**)、[Add stage (ステージの追加)] を選択します。

   > **Note**
   >
   > Lambda アクションを既存のステージに追加することもできます。デモンストレーション用に、ステージでの唯一のアクションとして Lambda 関数を追加し、パイプラインでアーティファクトが進行するにつれてその進行状況を簡単に表示できるようにしています。

5. [+ Add action group (+ アクションの追加)] を選択します。[アクションの編集] の、[アクション名] に、Lambda アクション (例: **MyLambdaAction**) の名前を入力します。[プロバイダ] で、[AWS Lambda] を選択します。[関数名] に Lambda 関数の名前 (例えば、**MyLambdaFunctionForAWSCodePipeline**) を選択または入力します。[ユー

ザーパラメータ] で、先ほどコピーした Amazon EC2 インスタンスの IP アドレス (例: **http://*192.0.2.4***) を指定し、[完了] を選択します。

> **Note**
>
> このトピックでは、IP アドレスを使用していますが、実際のシナリオでは、代わりに登録済みのウェブサイト名 (**http://*www.example.com*** など) を指定できます。のイベントデータとハンドラーの詳細については AWS Lambda、「 AWS Lambda デベロッパーガイド」の「プログラミングモデル」を参照してください。

6. [アクションの編集] ページで、[保存] を選択します。

ステップ 4 : Lambda 関数でパイプラインをテストする

関数をテストするには、パイプラインを通して最新の変更をリリースします。

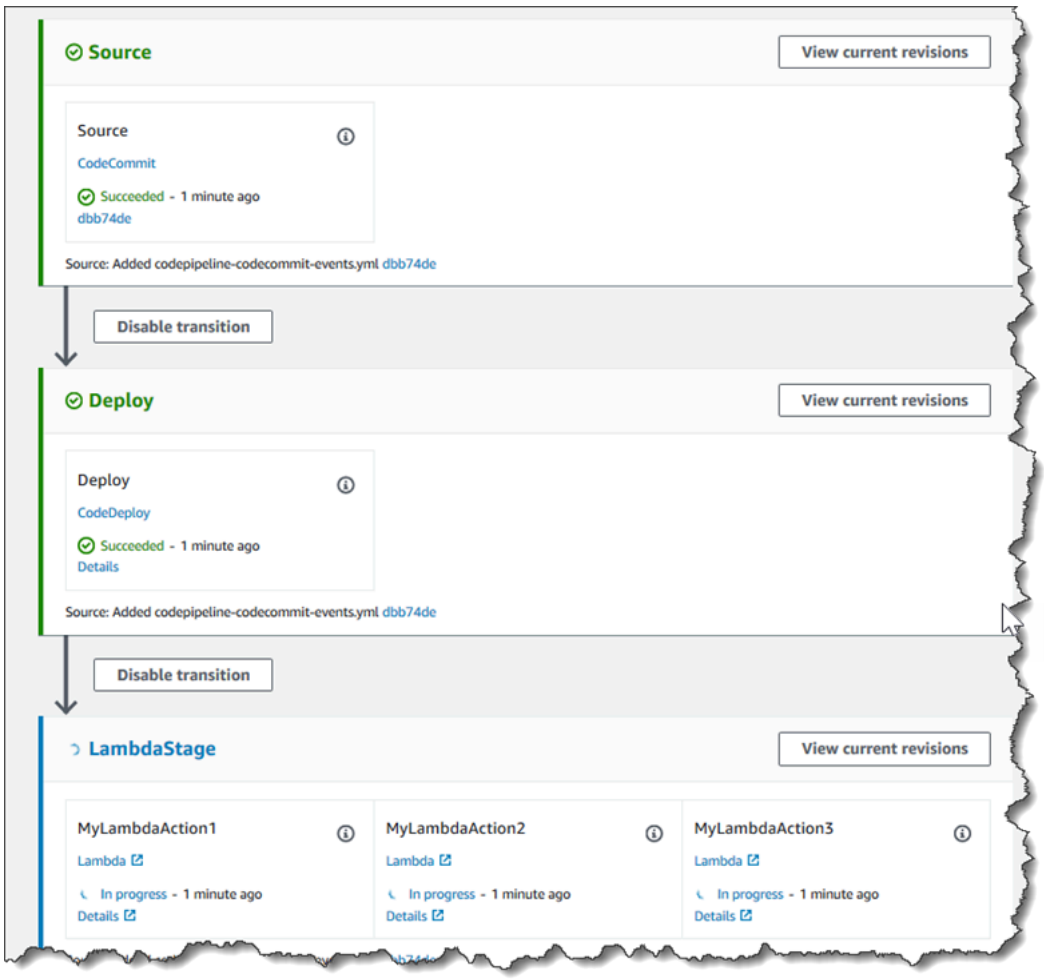
コンソールを使用してパイプラインによりアーティファクトの最新バージョンを実行するには

1. パイプラインの詳細ページで、[Release change] を選択します。これにより、ソースアクションで指定した各ソース場所における最新のリビジョンがパイプラインで実行されます。
2. Lambda アクションが完了したら、[詳細] リンクを選択して、Amazon CloudWatch での関数のログストリーム (イベントの課金期間を含む) を表示します。関数が失敗した場合は、CloudWatch ログからその原因について情報を得られます。

ステップ 5: 次のステップ

Lambda 関数を作成し、アクションとしてパイプラインに追加したので、次を実行できます。

- さらに Lambda アクションをステージに追加して他のウェブページをチェックします。
- Lambda 関数を変更し、別の文字列をチェックします。
- [Lambda 関数を試し]、パイプラインに独自の Lambda 関数を作成して追加します。



Lambda 関数の実験が完了したら、料金が発生しないように、パイプラインから削除し、から削除し AWS Lambda、IAM からロールを削除することを検討してください。詳細については、CodePipeline でパイプラインを編集する、CodePipeline でパイプラインを作成します。およびロールまたはインスタンスプロファイルの削除を参照してください。

JSON イベントの例

以下の例では、CodePipeline によって Lambda に送られたサンプル JSON イベントを示しています。このイベントの構造は、`GetJobDetails API` へのレスポンスと似ていますが、`actionTypeId` および `pipelineContext` データタイプがありません。2 つのアクション設定の詳細、`FunctionName` および `UserParameters` は、JSON イベントと `GetJobDetails` API へのレスポンスの両方に含まれます。`############` の値は例または説明であり、実際の値ではありません。

```
{
    "CodePipeline.job": {
```

```
        "id": "11111111-abcd-1111-abcd-111111abcdef",
        "accountId": "111111111111",
        "data": {
            "actionConfiguration": {
                "configuration": {
                    "FunctionName": "MyLambdaFunctionForAWSCodePipeline",
                    "UserParameters": "some-input-such-as-a-URL"
                }
            },
            "inputArtifacts": [
                {
                    "location": {
                        "s3Location": {
                            "bucketName": "the name of the bucket configured as the
pipeline artifact store in Amazon S3, for example codepipeline-us-east-2-1234567890",
                            "objectKey": "the name of the application, for example
CodePipelineDemoApplication.zip"
                        },
                        "type": "S3"
                    },
                    "revision": null,
                    "name": "ArtifactName"
                }
            ],
            "outputArtifacts": [],
            "artifactCredentials": {
                "secretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
                "sessionToken": "MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w
0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
WF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIw
EAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5
jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBb
WF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMx
HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wgZ8wDQYJKoZIhvcNAQE
BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
ITxOUSQv7c7ugFFDzQGBzZswY6786m86gpEIbb3OhjZnzcvQAaRHhdlQWIMm2nr
AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJIlJ00zbhNYS5f6Guo
EDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
3rrszlaEXAMPLE=",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE"
            },
```

```
            "continuationToken": "A continuation token if continuing job",
            "encryptionKey": {
              "id": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
              "type": "KMS"
            }
        }
    }
}
```

追加のサンプル関数

以下のサンプルLambda 関数は、CodePipeline のパイプラインに利用できる追加の機能を示しています。これらの機能を使うには、各サンプルの概要に示されている通りに、Lambda 実行ロールのポリシーを変更する必要がある場合があります。

トピック

- AWS CloudFormation テンプレートを使用するサンプル Python 関数

AWS CloudFormation テンプレートを使用するサンプル Python 関数

次のサンプルは、指定された CloudFormation テンプレートに基づいてスタックを作成または更新する 関数を示しています。テンプレートによって Amazon S3 バケットが作成されます。コストを最小限に抑えるため、デモンストレーション用に過ぎません。理想的には、バケットに何かアップロードする前に、スタックを削除した方が良いでしょう。バケットにファイルをアップロードすると、スタックを削除する際にバケットを削除することはできません。バケット自体を削除するには、バケット内をすべて手動で削除する必要があります。

この Python サンプルは、Amazon S3 バケットをソースアクションとして使用するパイプラインがあること、またはパイプラインでバージョニングされた Amazon S3 バケットにアクセスできることを前提としています。 CloudFormation テンプレートを作成して圧縮し、.zip ファイルとしてそのバケットにアップロードします。次に、この .zip ファイルをバケットから取得するソースアクションをパイプラインに追加する必要があります。

> **Note**
>
> Amazon S3 がパイプラインのソースプロバイダーである場合、ソースファイルを 1 つの .zip に圧縮し、その .zip をソースバケットにアップロードできます。解凍されたファイルを 1 つ

アップロードすることもできます。ただし、.zip ファイルを想定するダウンストリームアクションは失敗します。

このサンプルは、以下を紹介します:

- 複数の設定値を関数に渡すために JSON でエンコードされたユーザーパラメータの使用 (`get_user_params`)。
- アーティファクトバケットにおける .zip アーティファクトとの相互作用 (`get_template`)。
- 長時間実行の非同期処理を監視する継続トークンの使用 (`continue_job_later`)。これにより、15 分のランタイム (Lambda の制限) を超えてもアクションを続行し、関数を成功させることができます。

このサンプル Lambda 関数を使用するには、このサンプルポリシーに示すように、Lambda 実行ロールのポリシーに CloudFormation、、Amazon S3、および CodePipeline の`Allow`アクセス許可が必要です。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "logs:*"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:logs:*:*:*"
        },
        {
            "Action": [
                "codepipeline:PutJobSuccessResult",
                "codepipeline:PutJobFailureResult"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
```

```
                "cloudformation:DescribeStacks",
                "cloudformation:CreateStack",
                "cloudformation:UpdateStack"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "s3:*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

CloudFormation テンプレートを作成するには、プレーンテキストエディタを開き、次のコードをコピーして貼り付けます。

```
{
  "AWSTemplateFormatVersion" : "2010-09-09",
  "Description" : "CloudFormation template which creates an S3 bucket",
  "Resources" : {
    "MySampleBucket" : {
      "Type" : "AWS::S3::Bucket",
      "Properties" : {
      }
    }
  },
  "Outputs" : {
    "BucketName" : {
      "Value" : { "Ref" : "MySampleBucket" },
      "Description" : "The name of the S3 bucket"
    }
  }
}
```

これを **template.json** という名前の JSON ファイルとして、**template-package** という名前のディレクトリに保存します。このディレクトリと **template-package.zip** という名前のファイルを圧縮して (.zip) ファイルを作成し、圧縮されたファイルをバージョニングされた Amazon S3 バ

ケットにアップロードします。すでにパイプラインに設定したバケットがある場合、それを使用できます。次に、パイプラインを編集して .zip ファイルを取得するソースアクションを追加します。このアクションの出力に *MyTemplate* と名前を付けます。詳細については、「CodePipeline でパイプラインを編集する」を参照してください。

> **Note**
>
> サンプル Lambda 関数は、これらのファイル名と圧縮された構造を想定しています。ただし、このサンプルの代わりに独自の CloudFormation テンプレートを使用できます。独自のテンプレートを使用する場合は、 CloudFormation テンプレートに必要な追加機能を許可するように Lambda 実行ロールのポリシーを変更してください。

以下のコードを Lambda の 関数として追加するには

1. Lambda コンソールを開き、[関数の作成] を選択します。
2. [関数の作成] ページで、[一から作成] を選択します。[関数の名前] に、Lambda 関数の名前を入力します。
3. [ランタイム] で [Python2.7] を選択します。
4. [実行ロールを選択または作成] で、[既存のロールを使用する] を選択します。[Existing role (既存のロール)] でロールを選択し、[Create function (関数の作成)] を選択します。

   作成した関数の詳細ページが開きます。
5. 次のコードを関数コードボックスに貼り付けます。

```
from __future__ import print_function
from boto3.session import Session

import json
import urllib
import boto3
import zipfile
import tempfile
import botocore
import traceback

print('Loading function')

cf = boto3.client('cloudformation')
```

```
code_pipeline = boto3.client('codepipeline')

def find_artifact(artifacts, name):
    """Finds the artifact 'name' among the 'artifacts'

    Args:
        artifacts: The list of artifacts available to the function
        name: The artifact we wish to use
    Returns:
        The artifact dictionary found
    Raises:
        Exception: If no matching artifact is found

    """
    for artifact in artifacts:
        if artifact['name'] == name:
            return artifact

    raise Exception('Input artifact named "{0}" not found in event'.format(name))

def get_template(s3, artifact, file_in_zip):
    """Gets the template artifact

    Downloads the artifact from the S3 artifact store to a temporary file
    then extracts the zip and returns the file containing the CloudFormation
    template.

    Args:
        artifact: The artifact to download
        file_in_zip: The path to the file within the zip containing the template

    Returns:
        The CloudFormation template as a string

    Raises:
        Exception: Any exception thrown while downloading the artifact or unzipping
 it

    """
    tmp_file = tempfile.NamedTemporaryFile()
    bucket = artifact['location']['s3Location']['bucketName']
    key = artifact['location']['s3Location']['objectKey']

    with tempfile.NamedTemporaryFile() as tmp_file:
```

```
        s3.download_file(bucket, key, tmp_file.name)
        with zipfile.ZipFile(tmp_file.name, 'r') as zip:
            return zip.read(file_in_zip)

def update_stack(stack, template):
    """Start a CloudFormation stack update

    Args:
        stack: The stack to update
        template: The template to apply

    Returns:
        True if an update was started, false if there were no changes
        to the template since the last update.

    Raises:
        Exception: Any exception besides "No updates are to be performed."

    """
    try:
        cf.update_stack(StackName=stack, TemplateBody=template)
        return True

    except botocore.exceptions.ClientError as e:
        if e.response['Error']['Message'] == 'No updates are to be performed.':
            return False
        else:
            raise Exception('Error updating CloudFormation stack
 "{0}"'.format(stack), e)

def stack_exists(stack):
    """Check if a stack exists or not

    Args:
        stack: The stack to check

    Returns:
        True or False depending on whether the stack exists

    Raises:
        Any exceptions raised .describe_stacks() besides that
        the stack doesn't exist.

    """
```

```
    try:
        cf.describe_stacks(StackName=stack)
        return True
    except botocore.exceptions.ClientError as e:
        if "does not exist" in e.response['Error']['Message']:
            return False
        else:
            raise e

def create_stack(stack, template):
    """Starts a new CloudFormation stack creation

    Args:
        stack: The stack to be created
        template: The template for the stack to be created with

    Throws:
        Exception: Any exception thrown by .create_stack()
    """
    cf.create_stack(StackName=stack, TemplateBody=template)

def get_stack_status(stack):
    """Get the status of an existing CloudFormation stack

    Args:
        stack: The name of the stack to check

    Returns:
        The CloudFormation status string of the stack such as CREATE_COMPLETE

    Raises:
        Exception: Any exception thrown by .describe_stacks()

    """
    stack_description = cf.describe_stacks(StackName=stack)
    return stack_description['Stacks'][0]['StackStatus']

def put_job_success(job, message):
    """Notify CodePipeline of a successful job

    Args:
        job: The CodePipeline job ID
        message: A message to be logged relating to the job status
```

```
    Raises:
        Exception: Any exception thrown by .put_job_success_result()

    """
    print('Putting job success')
    print(message)
    code_pipeline.put_job_success_result(jobId=job)

def put_job_failure(job, message):
    """Notify CodePipeline of a failed job

    Args:
        job: The CodePipeline job ID
        message: A message to be logged relating to the job status

    Raises:
        Exception: Any exception thrown by .put_job_failure_result()

    """
    print('Putting job failure')
    print(message)
    code_pipeline.put_job_failure_result(jobId=job, failureDetails={'message':
 message, 'type': 'JobFailed'})

def continue_job_later(job, message):
    """Notify CodePipeline of a continuing job

    This will cause CodePipeline to invoke the function again with the
    supplied continuation token.

    Args:
        job: The JobID
        message: A message to be logged relating to the job status
        continuation_token: The continuation token

    Raises:
        Exception: Any exception thrown by .put_job_success_result()

    """

    # Use the continuation token to keep track of any job execution state
    # This data will be available when a new job is scheduled to continue the
 current execution
    continuation_token = json.dumps({'previous_job_id': job})
```

```
    print('Putting job continuation')
    print(message)
    code_pipeline.put_job_success_result(jobId=job,
 continuationToken=continuation_token)

def start_update_or_create(job_id, stack, template):
    """Starts the stack update or create process

    If the stack exists then update, otherwise create.

    Args:
        job_id: The ID of the CodePipeline job
        stack: The stack to create or update
        template: The template to create/update the stack with

    """
    if stack_exists(stack):
        status = get_stack_status(stack)
        if status not in ['CREATE_COMPLETE', 'ROLLBACK_COMPLETE',
 'UPDATE_COMPLETE']:
            # If the CloudFormation stack is not in a state where
            # it can be updated again then fail the job right away.
            put_job_failure(job_id, 'Stack cannot be updated when status is: ' +
 status)
            return

        were_updates = update_stack(stack, template)

        if were_updates:
            # If there were updates then continue the job so it can monitor
            # the progress of the update.
            continue_job_later(job_id, 'Stack update started')

        else:
            # If there were no updates then succeed the job immediately
            put_job_success(job_id, 'There were no stack updates')
    else:
        # If the stack doesn't already exist then create it instead
        # of updating it.
        create_stack(stack, template)
        # Continue the job so the pipeline will wait for the CloudFormation
        # stack to be created.
        continue_job_later(job_id, 'Stack create started')
```

```
def check_stack_update_status(job_id, stack):
    """Monitor an already-running CloudFormation update/create

    Succeeds, fails or continues the job depending on the stack status.

    Args:
        job_id: The CodePipeline job ID
        stack: The stack to monitor

    """
    status = get_stack_status(stack)
    if status in ['UPDATE_COMPLETE', 'CREATE_COMPLETE']:
        # If the update/create finished successfully then
        # succeed the job and don't continue.
        put_job_success(job_id, 'Stack update complete')

    elif status in ['UPDATE_IN_PROGRESS', 'UPDATE_ROLLBACK_IN_PROGRESS',
    'UPDATE_ROLLBACK_COMPLETE_CLEANUP_IN_PROGRESS', 'CREATE_IN_PROGRESS',
    'ROLLBACK_IN_PROGRESS', 'UPDATE_COMPLETE_CLEANUP_IN_PROGRESS']:
        # If the job isn't finished yet then continue it
        continue_job_later(job_id, 'Stack update still in progress')

    else:
        # If the Stack is a state which isn't "in progress" or "complete"
        # then the stack update/create has failed so end the job with
        # a failed result.
        put_job_failure(job_id, 'Update failed: ' + status)

def get_user_params(job_data):
    """Decodes the JSON user parameters and validates the required properties.

    Args:
        job_data: The job data structure containing the UserParameters string which
 should be a valid JSON structure

    Returns:
        The JSON parameters decoded as a dictionary.

    Raises:
        Exception: The JSON can't be decoded or a property is missing.

    """
    try:
```

```
        # Get the user parameters which contain the stack, artifact and file
 settings
        user_parameters = job_data['actionConfiguration']['configuration']
['UserParameters']
        decoded_parameters = json.loads(user_parameters)

    except Exception as e:
        # We're expecting the user parameters to be encoded as JSON
        # so we can pass multiple values. If the JSON can't be decoded
        # then fail the job with a helpful message.
        raise Exception('UserParameters could not be decoded as JSON')

    if 'stack' not in decoded_parameters:
        # Validate that the stack is provided, otherwise fail the job
        # with a helpful message.
        raise Exception('Your UserParameters JSON must include the stack name')

    if 'artifact' not in decoded_parameters:
        # Validate that the artifact name is provided, otherwise fail the job
        # with a helpful message.
        raise Exception('Your UserParameters JSON must include the artifact name')

    if 'file' not in decoded_parameters:
        # Validate that the template file is provided, otherwise fail the job
        # with a helpful message.
        raise Exception('Your UserParameters JSON must include the template file
 name')

    return decoded_parameters

def setup_s3_client(job_data):
    """Creates an S3 client

    Uses the credentials passed in the event by CodePipeline. These
    credentials can be used to access the artifact bucket.

    Args:
        job_data: The job data structure

    Returns:
        An S3 client with the appropriate credentials

    """
    key_id = job_data['artifactCredentials']['accessKeyId']
```

```
    key_secret = job_data['artifactCredentials']['secretAccessKey']
    session_token = job_data['artifactCredentials']['sessionToken']

    session = Session(aws_access_key_id=key_id,
        aws_secret_access_key=key_secret,
        aws_session_token=session_token)
    return session.client('s3',
 config=botocore.client.Config(signature_version='s3v4'))

def lambda_handler(event, context):
    """The Lambda function handler

    If a continuing job then checks the CloudFormation stack status
    and updates the job accordingly.

    If a new job then kick of an update or creation of the target
    CloudFormation stack.

    Args:
        event: The event passed by Lambda
        context: The context passed by Lambda

    """
    try:
        # Extract the Job ID
        job_id = event['CodePipeline.job']['id']

        # Extract the Job Data
        job_data = event['CodePipeline.job']['data']

        # Extract the params
        params = get_user_params(job_data)

        # Get the list of artifacts passed to the function
        artifacts = job_data['inputArtifacts']

        stack = params['stack']
        artifact = params['artifact']
        template_file = params['file']

        if 'continuationToken' in job_data:
            # If we're continuing then the create/update has already been triggered
            # we just need to check if it has finished.
            check_stack_update_status(job_id, stack)
```

```
        else:
            # Get the artifact details
            artifact_data = find_artifact(artifacts, artifact)
            # Get S3 client to access artifact with
            s3 = setup_s3_client(job_data)
            # Get the JSON template file out of the artifact
            template = get_template(s3, artifact_data, template_file)
            # Kick off a stack update or create
            start_update_or_create(job_id, stack, template)

    except Exception as e:
        # If any other exceptions which we didn't expect are raised
        # then fail the job and log the exception message.
        print('Function failed due to exception.')
        print(e)
        traceback.print_exc()
        put_job_failure(job_id, 'Function exception: ' + str(e))

    print('Function complete.')
    return "Complete."
```

6. [ハンドラー] をデフォルト値のままにし、[ロール] 以前に選択または作成した名前 **CodePipelineLambdaExecRole** のままにします。
7. 基本設定のタイムアウトで、デフォルトの 3 秒を **20** に置き換えます。
8. [保存] を選択します。
9. CodePipeline コンソールから、パイプラインを編集して、関数をパイプラインのステージのアクションとして追加します。変更するパイプラインステージの [編集] を選択し、[アクショングループを追加] します。[アクションの編集] ページで、[Action name (アクション名)] にアクションの名前を入力します。[アクションプロバイダー] で、[Lambda] を選択します。

   [アーティファクト入力] で `MyTemplate` を選択します。[UserParameters] で JSON 文字列に次の 3 つのパラメータを指定する必要があります。

   - スタック名
   - CloudFormation テンプレート名とファイルへのパス
   - アーティファクト入力

   中括弧 ({ }) を使用し、パラメータをカンマで区切ります。例えば、*MyTemplate* のパイプラインに、[アーティファクト入力] で *MyTestStack* というスタックを作

成するには、UserParameters に {"stack":"*MyTestStack*","file":"template-package/template.json","artifact":"*MyTemplate*"} を入力します。

> **Note**
>
> [UserParameters] で入力アーティファクトを指定した場合でも、[入力アーティファクト] のアクションに対しては、この入力アーティファクトをやはり指定する必要があります。

10. 変更をパイプラインに保存したら、手動で変更をリリースして、アクションと Lambda 関数をテストします。

手動の承認アクションをステージに追加する

では AWS CodePipeline、必要なアクセス許可を持つ AWS Identity and Access Management ユーザーがアクションを承認または拒否できるように、パイプラインの実行を停止する時点でパイプライン内のステージに承認アクションを追加できます。

アクションが承認されると、パイプラインの実行が再開されます。アクションが拒否されているか、パイプラインの到達および停止のアクションが 7 日間以内に承認または拒否されない場合は、アクションが失敗した時と同じ結果になり、パイプラインの実行は継続されません。

次の理由で手動承認を使用する場合があります。

- リビジョンがパイプラインの次のステージに移行される前に、コードレビューの実施または管理レビューの変更を行う場合
- リリース前に、最新バージョンのアプリケーションで手動の品質保証を実行するか、ビルドアーティファクトの完全性を確認する場合
- 企業のウェブサイトに公開する前に新規テキストまたは更新済みのテキストをレビューしてもらう場合

CodePipeline でのマニュアルの承認アクションに関する設定オプション

CodePipeline には、承認アクションに関して承認者に通知することができる 3 つの設定オプションがあります。

承認通知の発行 パイプラインがアクションで停止されると、Amazon Simple Notification Service のトピックにメッセージが発行されるように、承認アクションを設定できます。Amazon SNS は、ト

ピックにサブスクライブされた各エンドポイントにメッセージを送信します。承認アクションを含むパイプラインと同じ AWS リージョンで作成されたトピックを使用する必要があります。トピックを作成する際には、`MyFirstPipeline-us-east-2-approval` などの形式で、目的を識別しやすい名前を付けることをお勧めします。

承認通知を Amazon SNS トピックに発行する場合は、E メールか SMS の受信者、SQS キュー、HTTP/HTTPS エンドポイント、または Amazon SNS を用いて呼び出す AWS Lambda 関数などの形式から選択することができます。Amazon SNS トピックの通知の詳細については、以下のトピックを参照してください。

- Amazon Simple Notification Service
- Amazon SNS トピックを作成します
- Amazon SQS キューへの Amazon SNS メッセージの送信
- Amazon SNS トピックへのキューのサブスクライブ
- HTTP/HTTPS エンドポイントへの Amazon SNS メッセージの送信
- Amazon SNS 通知を用いた Lambda 関数の呼び出し

承認アクションの通知で生成される JSON データの構造については、「CodePipeline でのマニュアルの承認通知の JSON データ形式」を参照してください。

レビューする URL の指定 承認アクションの設定の一部として、レビューする URL を指定することができます。この URL は、承認者がテストするウェブアプリケーションか、承認リクエストに関する詳細を含むページへのリンクです。また、URL には、Amazon SNS トピックに対して発行される通知が含まれます。承認者は、コンソールまたは CLI を使用して表示することができます。

承認者に対するコメントの入力 承認アクションを作成する場合、コメントを追加して、通知の受取者、またはコンソールか CLI レスポンスのアクションを確認するユーザーに表示することもできます。

設定オプション不要 これらの 3 つのオプションのいずれも設定しないように選択することもできます。たとえば、アクションがレビューできる状態になったことや、アクションを承認するまでパイプラインを停止することを直接通知する場合などに、これらの設定は不要です。

CodePipeline での承認アクションのセットアップおよびワークフローの概要

手動の承認の設定および使用に関する概要は次のとおりです。

1. 承認アクションの承認または拒否に必要な IAM のアクセス許可を組織内の IAM ロールに 1 つ以上付与します。
2. (オプション) Amazon SNS 通知を使用している場合、CodePipeline オペレーションで使用しているサービスロールには、Amazon SNS リソースにアクセスするためのアクセス許可が含まれます。
3. (オプション) Amazon SNS 通知を使用している場合、Amazon SNS のトピックを作成し、1 人以上の受信者または 1 つ以上のエンドポイントを追加します。
4. AWS CLI を使用してパイプラインを作成する場合、または CLI またはコンソールを使用してパイプラインを作成した後、パイプラインのステージに承認アクションを追加します。

   通知を使用している場合は、アクションの設定に Amazon SNS トピックの Amazon リソースネーム (ARN) を含みます。(ARN は Amazon リソースの一意の識別子です。 Amazon SNS トピック ARNs は*#arn:aws:sns:us-east-2:80398EXAMPLE:MyApprovalTopic* のように構造化されています。 詳細については、ARNs) と AWS のサービス 名前空間」を参照してくださいAmazon Web Services 全般のリファレンス。)
5. 承認アクションに達すると、パイプラインは停止します。Amazon SNS トピックの ARN がアクション設定に含まれている場合、通知は Amazon SNS トピックに発行され、メッセージは、コンソールの承認アクションをレビューするリンクと合わせて、トピックの受信者またはサブスクライブされたエンドポイントに送信されます。
6. 承認者は、必要に応じて、ターゲット URL およびコメントを確認します。
7. 承認者は、コンソール、CLI、SDK を使用して、概要コメントを確認し、次のようなレスポンスを送信します。
   - 承認済み: パイプラインの実行が再開されます。
   - 拒否: ステージステータスが「失敗」に変更され、パイプラインの実行は再開されません。

   7 日以内にレスポンスが送信されない場合、アクションは「失敗」とマークされます。

CodePipeline で IAM ユーザーに承認アクセス許可を付与する

組織内の IAM ユーザーが承認アクションを承認または拒否するには、パイプラインのアクセスと承認アクションのステータスの更新を行うアクセス許可が必要です。すべてのパイプラインと、アカウントの承認アクションに対するアクセス許可を付与するには、`AWSCodePipelineApproverAccess` マネージドポリシーを IAM ユーザー、ロール、またはグループにアタッチします。また、アクセス許可を制限するには、IAM ユーザー、ロール、またはグループでアクセスできる各リソースを指定します。

> **Note**
>
> このトピックに記載されているアクセス許可でアクセスできる範囲はかなり制限されています。ユーザー、ロール、グループを有効化して、複数の承認アクションを承認または拒否するには、他の管理ポリシーをアタッチします。CodePipeline で利用できるマネージドポリシーの詳細については、[AWS の 管理ポリシー AWS CodePipeline] を参照してください。

すべてのパイプラインおよび承認アクションに承認アクセス許可を付与する

CodePipeline で承認アクションを実行する必要があるユーザーには、`AWSCodePipelineApproverAccess` マネージドポリシーを使用します。

アクセス許可を付与するにはユーザー、グループ、またはロールにアクセス許可を追加します。

- のユーザーとグループ AWS IAM アイデンティティセンター:

  アクセス許可セットを作成します。「AWS IAM アイデンティティセンター ユーザーガイド」の「アクセス許可セットを作成する」の手順に従ってください。
- IAM 内で、ID プロバイダーによって管理されているユーザー:

  ID フェデレーションのロールを作成します。詳細については IAM ユーザーガイド の サードパーティー ID プロバイダー (フェデレーション) 用のロールを作成する を参照してください。
- IAM ユーザー:
  - ユーザーが担当できるロールを作成します。手順については IAM ユーザーガイド の IAM ユーザーのロールの作成 を参照してください。
  - (お奨めできない方法) ポリシーをユーザーに直接アタッチするか、ユーザーをユーザーグループに追加します。「IAM ユーザーガイド」の「ユーザー (コンソール) へのアクセス許可の追加」の指示に従います。

特定のパイプラインや承認アクションに対する承認アクセス許可を指定します。

CodePipeline で承認アクションを実行する必要があるユーザーには、以下のカスタムポリシーを使用します。以下のポリシーで、ユーザーがアクセスできる個々のリソースを指定します。たとえば、以下のポリシーは、米国東部 (オハイオ) リージョン (us-east-2) の `MyApprovalAction` パイプラインで `MyFirstPipeline` という名前のアクションのみを承認または拒否する権限をユーザーに付与します。

> **Note**
>
> IAM ユーザーが CodePipeline ダッシュボードにアクセスしてこのパイプラインのリストを表示する必要がある場合にのみ、codepipeline:ListPipelines アクセス許可が必要です。コンソールへのアクセスが必要ない場合は、codepipeline:ListPipelines を省略できます。

JSON ポリシーエディタでポリシーを作成するには

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/iam/ で IAM コンソールを開きます。
2. 左側のナビゲーションペインで、[ポリシー] を選択します。

   初めて [ポリシー] を選択する場合には、[管理ポリシーにようこそ] ページが表示されます。今すぐ始める を選択します。
3. ページの上部で、[ポリシーを作成] を選択します。
4. ポリシーエディタ セクションで、JSON オプションを選択します。
5. 次の JSON ポリシードキュメントを入力します。

```
{
   "Version": "2012-10-17",
   "Statement": [
       {
           "Effect": "Allow",
           "Action": [
               "codepipeline:ListPipelines"
           ],
           "Resource": [
               "*"
           ]
       },
       {
           "Effect": "Allow",
           "Action": [
               "codepipeline:GetPipeline",
               "codepipeline:GetPipelineState",
               "codepipeline:GetPipelineExecution"
           ],
```

```
            "Resource": "arn:aws:codepipeline:us-
east-2:80398EXAMPLE:MyFirstPipeline"
        },
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:PutApprovalResult"
            ],
            "Resource": "arn:aws:codepipeline:us-
east-2:80398EXAMPLE:MyFirstPipeline/MyApprovalStage/MyApprovalAction"
        }
    ]
}
```

6. 次へ を選択します。

> **Note**
>
> いつでも Visual と JSON エディタオプションを切り替えることができます。ただし、[ビジュアル] エディタで [次へ] に変更または選択した場合、IAM はポリシーを再構成してビジュアルエディタに合わせて最適化することがあります。詳細については、IAM ユーザーガイド の ポリシーの再構成 を参照してください。

7. 確認と作成 ページで、作成するポリシーの ポリシー名 と 説明 (オプション) を入力します。このポリシーで定義されているアクセス許可 を確認して、ポリシーによって付与されたアクセス許可を確認します。
8. ポリシーを作成 をクリックして、新しいポリシーを保存します。

Amazon SNS アクセス権限を CodePipeline サービスロールに付与する

承認アクションでレビューが必要な際に、Amazon SNS を使用してトピックに通知を発行する場合、CodePipeline オペレーションで使用しているサービスロールでアクセス許可を付与して、Amazon SNS リソースにアクセスできるようにする必要があります。IAM コンソールを使用して、このアクセス許可をサービスロールに追加することができます。

以下のポリシーで、SNS で公開するためのポリシーを指定します。以下のポリシーには、SNSPublish という名前を付けることができます。以下のポリシーをサービスロールにアタッチして使用します。

> **Important**
>
> で使用したのと同じアカウント情報 AWS マネジメントコンソール を使用して にサインインしていることを確認しますCodePipeline の使用開始。

JSON

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

JSON ポリシーエディタでポリシーを作成するには

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/iam/ で IAM コンソールを開きます。
2. 左側のナビゲーションペインで、[ポリシー] を選択します。

   初めて [ポリシー] を選択する場合には、[管理ポリシーにようこそ] ページが表示されます。今すぐ始める を選択します。
3. ページの上部で、[ポリシーを作成] を選択します。
4. ポリシーエディタ セクションで、JSON オプションを選択します。
5. JSON ポリシードキュメントを入力するか貼り付けます。IAM ポリシー言語の詳細については、「IAM JSON ポリシーリファレンス」を参照してください。
6. ポリシーの検証中に生成されたセキュリティ警告、エラー、または一般警告をすべて解決してから、[次へ] を選択します。

> **Note**
>
> いつでも [ビジュアル] と [JSON] エディタオプションを切り替えることができます。ただし、[ビジュアル] エディタで [次へ] に変更または選択した場合、IAM はポリシーを再構成してビジュアルエディタに合わせて最適化することがあります。詳細については、「IAM ユーザーガイド」の「ポリシーの再構成」を参照してください。

7. (オプション) でポリシーを作成または編集するときに AWS マネジメントコンソール、テンプレートで使用できる JSON または YAML ポリシー CloudFormation テンプレートを生成できます。

   これを行うには、ポリシーエディタで [アクション] を選択し、次に [CloudFormation テンプレートを生成] を選択します。詳細については CloudFormation、「 AWS CloudFormation ユーザーガイド」の「 AWS Identity and Access Management リソースタイプのリファレンス」を参照してください。

8. ポリシーにアクセス権限を追加し終えたら、[次へ] を選択します。

9. [確認と作成] ページで、作成するポリシーの [ポリシー名] と [説明] (オプション) を入力します。このポリシーで定義されているアクセス許可 を確認して、ポリシーによって付与されたアクセス許可を確認します。

10. (オプション) タグをキーと 値のペアとしてアタッチして、メタデータをポリシーに追加します。IAM でのタグの使用の詳細については、IAM ユーザーガイドのAWS Identity and Access Management 「リソースのタグ」を参照してください。

11. [ポリシーを作成] をクリックして、新しいポリシーを保存します。

CodePipeline でパイプラインにマニュアルの承認アクションを追加する

CodePipeline でパイプラインのステージに承認アクションを追加した個所でパイプラインを停止することで、このアクションを手動で承認または拒否できます。

> **Note**
>
> 承認アクションをソースステージに追加することはできません。ソースステージには、ソースアクションのみ含めることができます。

承認アクションをレビューする準備が整ったときに、Amazon SNS を使用して通知を送信するには、まず次の前提条件を満たす必要があります。

- Amazon SNS リソースにアクセスできるように、アクセス許可を CodePipeline サービスロールに付与します。詳細については、「Amazon SNS アクセス権限を CodePipeline サービスロールに付与する」を参照してください。
- 承認アクションのステータスを更新できるようにするアクセス許可を組織の 1 人以上の IAM アイデンティティに付与します。詳細については、「CodePipeline で IAM ユーザーに承認アクセス許可を付与する」を参照してください。

この例では、新しい承認ステージを作成し、ステージにマニュアル承認アクションを追加します。マニュアル承認アクションを、他のアクションを含む既存のステージに追加することもできます。

CodePipeline でパイプラインにマニュアルの承認アクションを追加する (コンソール)

CodePipeline コンソールを使用して、既存の CodePipeline のパイプラインに承認アクションを追加します。新しいパイプラインを作成するときに承認アクションを追加する場合は、 CLI AWS を使用する必要があります。

1. CodePipeline コンソール (https://console.aws.amazon.com/codepipeline/) を開きます。
2. [Name] で、パイプラインを選択します。
3. パイプライン詳細ページで、[編集] を選択します。
4. 承認アクションを新しいステージに追加する場合は、承認リクエストを追加するパイプラインのポイントで [+Add stage (+ ステージの追加)] を選択し、ステージの名前を入力します。[ステージの追加 (Add stage)] ページの [Stage name (ステージ名)] に、新しいステージ名を入力します。たとえば、新しいステージを追加し、`Manual_Approval` という名前を付けます。

   承認アクションを既存のステージに追加する場合は、[ステージの編集] を選択します。
5. 承認アクションを追加するステージで、[+ Add action group (アクショングループの追加)] を選択します。
6. [アクションの編集] ページで、次の作業を行います。

   1. [アクション名] に、アクションを識別する名前を入力します。
   2. [アクションプロバイダー] の、[承認] で、[手動承認] を選択します。
   3. (オプション) [SNS トピックの ARN] で、承認アクションの通知を送るために使用するトピックの名前を選択します。

4. (オプション) [URL for review] に、承認者が検討するページまたはアプリケーションの URL を入力します。承認者は、パイプラインのコンソールビューに含まれるリンクからこの URL にアクセスできます。
5. (オプション) [コメント] に、レビュー者と共有するその他の情報を入力します。
6. [保存] を選択します。

CodePipeline でパイプラインにマニュアルの承認アクションを追加する (CLI)

CLI を使用して、承認アクションを既存のパイプラインに追加するか、パイプライン作成時に追加します。そのためには、作成中または編集中のステージで承認アクション (手動の承認タイプ) を追加します。

パイプラインの作成および編集に関する詳細は、「パイプライン、ステージ、アクションを作成する」および「CodePipeline でパイプラインを編集する」を参照してください。

承認アクションを含むパイプラインにステージを追加するには、パイプライン作成時または更新時に、次の例のように追加します。

> **Note**
>
> `configuration` セクションはオプションです。このセクションはファイルの一部であり、構造全体を示したものではありません。詳細については、「CodePipeline パイプライン構造リファレンス」を参照してください。

```
{
    "name": "MyApprovalStage",
    "actions": [
        {
            "name": "MyApprovalAction",
            "actionTypeId": {
                "category": "Approval",
                "owner": "AWS",
                "version": "1",
                "provider": "Manual"
            },
            "inputArtifacts": [],
            "outputArtifacts": [],
            "configuration": {
```

```
                "NotificationArn": "arn:aws:sns:us-
east-2:80398EXAMPLE:MyApprovalTopic",
                "ExternalEntityLink": "http://example.com",
                "CustomData": "The latest changes include feedback from Bob."},
            "runOrder": 1
        }
    ]
}
```

承認アクションが他のアクションを含むステージに存在する場合、このステージを含む JSON ファイルのセクションは、次の例のようになります。

> **Note**
>
> configuration セクションはオプションです。このセクションはファイルの一部であり、構造全体を示したものではありません。詳細については、「CodePipeline パイプライン構造リファレンス」を参照してください。

```
,
{
    "name": "Production",
    "actions": [
        {
            "inputArtifacts": [],
            "name": "MyApprovalAction",
            "actionTypeId": {
                "category": "Approval",
                "owner": "AWS",
                "version": "1",
                "provider": "Manual"
            },
            "outputArtifacts": [],
            "configuration": {
                "NotificationArn": "arn:aws:sns:us-
east-2:80398EXAMPLE:MyApprovalTopic",
                "ExternalEntityLink": "http://example.com",
                "CustomData": "The latest changes include feedback from Bob."
            },
            "runOrder": 1
        },
        {
```

```
            "inputArtifacts": [
                {
                    "name": "MyApp"
                }
            ],
            "name": "MyDeploymentAction",
            "actionTypeId": {
                "category": "Deploy",
                "owner": "AWS",
                "version": "1",
                "provider": "CodeDeploy"
            },
            "outputArtifacts": [],
            "configuration": {
                "ApplicationName": "MyDemoApplication",
                "DeploymentGroupName": "MyProductionFleet"
            },
            "runOrder": 2
        }
    ]
}
```

CodePipeline で承認アクションを承認または拒否する

パイプラインに承認アクションが含まれている場合、パイプラインの実行は、アクションが実行されたポイントで停止します。手動でこのアクションを承認しない限り、パイプラインが再開されることはありません。承認者がアクションを拒否する場合、または承認アクションでパイプラインが停止してから 7 日以内に承認レスポンスを受け取らない場合、パイプラインのステータスは「失敗」になります。

パイプラインに承認アクションを追加した人が通知を設定していると、パイプラインの情報と承認のステータスを含む E メールを受け取る場合があります。

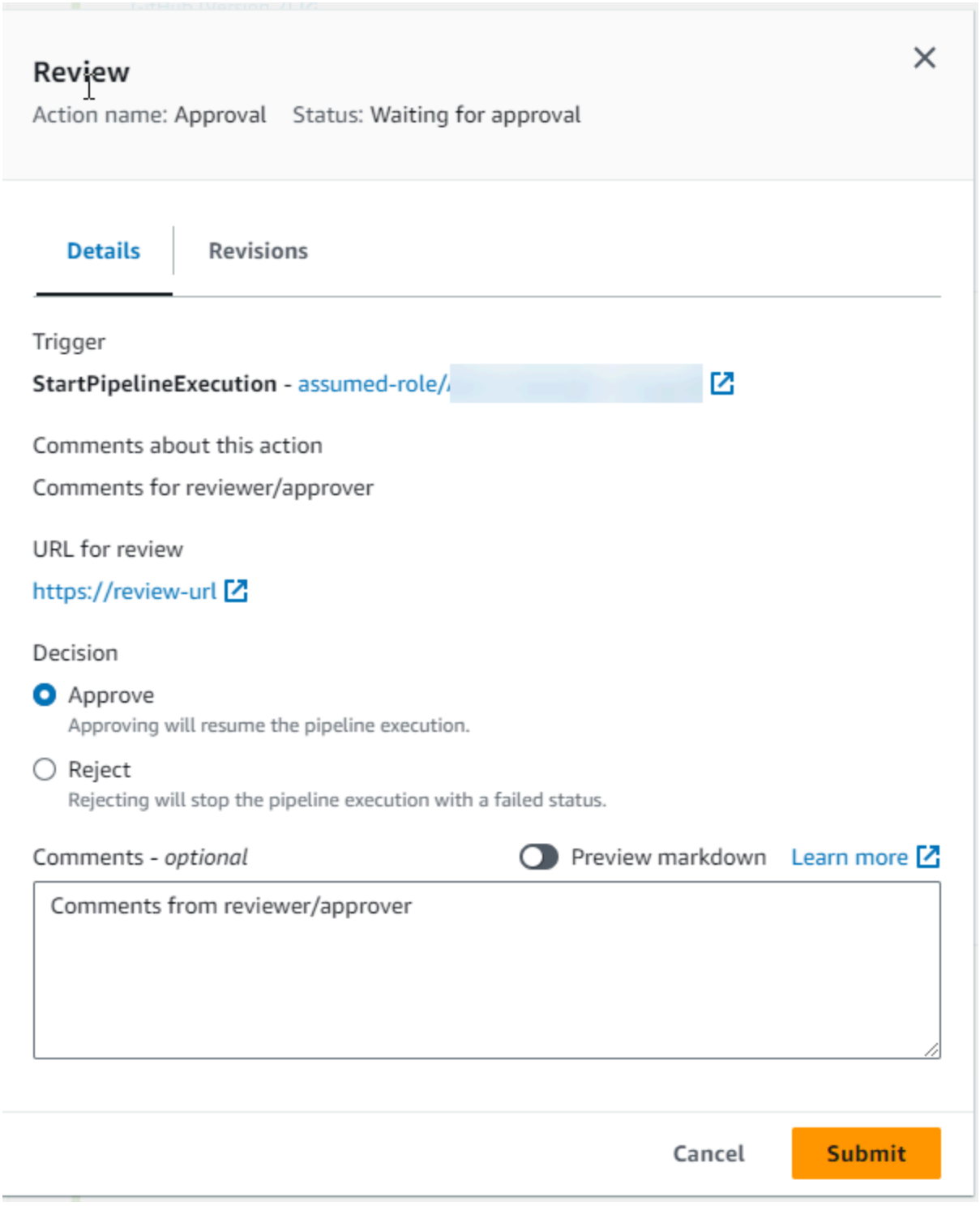
承認アクションを承認または拒否する (コンソール)

承認アクションへの直接リンクを含む通知を受け取った場合は、[Approve or reject (承認または却下)] リンクを選択し、コンソールにサインインし、このステップ 7 に進みます。そうでない場合は、以下のすべての手順を実行します。

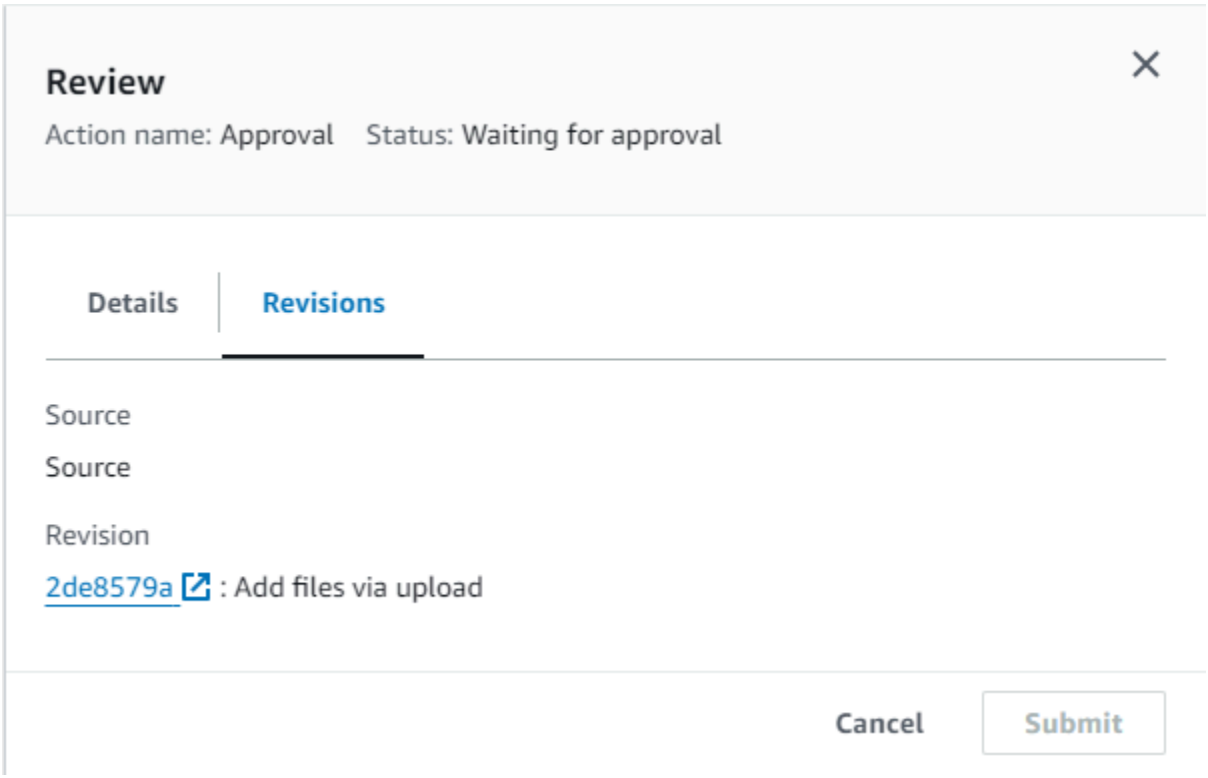
1. CodePipeline コンソール (https://console.aws.amazon.com/codepipeline/) を開きます。
2. [All Pipelines (すべてのパイプライン)] ページで、パイプラインの名前を選択します。

3. 承認アクションが含まれるステージを見つけます。[Review] (レビュー) を選択します。

   [レビュー] ダイアログボックスが表示されます。[詳細] タブには、レビューの内容とコメントが表示されます。



   [リビジョン] タブには、実行のソースリビジョンが表示されます。



4. [詳細] タブには、コメントと URL (ある場合) が表示されます。メッセージに、レビューするコンテンツの URL があれば、それも表示されます。
5. URL が表示された場合は、アクションの [レビューの URL] リンクを選択して、ターゲットウェブページを開き、コンテンツをレビューします。
6. [レビュー] ウィンドウで、アクションを承認または拒否した理由など、レビューコメントを入力し、[承認] または [拒否] を選択します。
7. [Submit] を選択してください。

承認リクエストを承認または拒否する (CLI)

CLI を使用して承認アクションに応答するには、まず get-pipeline-state コマンドを使用して、最新の承認アクションの実行に関連付けられているトークンを取得します。

1. ターミナル (Linux、macOS、または Unix) またはコマンドプロンプト (Windows) で、承認アクションが含まれるパイプラインに対して [get-pipeline-state] コマンドを実行します。たとえば、*MyFirstPipeline* という名前のパイプラインに対して、以下のように入力します:

```
aws codepipeline get-pipeline-state --name MyFirstPipeline
```

2. コマンドに対する応答で、次に示すような承認アクションの actionStates セクションの latestExecution にある token 値を見つけます。

```
{
    "created": 1467929497.204,
    "pipelineName": "MyFirstPipeline",
    "pipelineVersion": 1,
    "stageStates": [
        {
            "actionStates": [
                {
                    "actionName": "MyApprovalAction",
                    "currentRevision": {
                        "created": 1467929497.204,
                        "revisionChangeId": "CEM7d6Tp7zfelUSLCPPwo234xEXAMPLE",
                        "revisionId": "HYGp7zmwbCPPwo23xCMdTeqIlEXAMPLE"
                    },
                    "latestExecution": {
                        "lastUpdatedBy": "identity",
                        "summary": "The new design needs to be reviewed before
 release.",
                        "token": "1a2b3c4d-573f-4ea7-a67E-XAMPLETOKEN"
                    }
                }
//More content might appear here
}
```

3. プレーンテキストエディタで、JSON 形式で以下の情報を記録するファイルを追加します。

- 承認アクションを含むパイプラインの名前。
- 承認アクションを含むステージの名前。
- 承認アクションの名前。
- 前の手順で収集したトークン値。
- アクションに対する応答 (承認済みまたは却下)。応答の先頭は大文字であることが必要です。
- 概要コメント。

先ほどの *MyFirstPipeline* の例では、ファイルは以下のようになります:

```
{
  "pipelineName": "MyFirstPipeline",
```

```
  "stageName": "MyApprovalStage",
  "actionName": "MyApprovalAction",
  "token": "1a2b3c4d-573f-4ea7-a67E-XAMPLETOKEN",
  "result": {
    "status": "Approved",
    "summary": "The new design looks good. Ready to release to customers."
  }
}
```

4. **approvalstage-approved.json** のような名前でファイルを保存します。
5. 以下のように、承認 JSON ファイルの名前を指定して、put-approval-result コマンドを実行します。

> **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 codepipeline put-approval-result --cli-input-json file://approvalstage-approved.json
```

CodePipeline でのマニュアルの承認通知の JSON データ形式

Amazon SNS 通知を使用する承認アクションの場合、アクションに関する JSON データは、パイプライン停止時に Amazon SNS に対して作成し、発行されます。この JSON 出力を使用して、メッセージを Amazon SQS のキューに送信するか、 AWS Lambdaの関数を呼び出します。

> **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。詳細については、[Amazon SNS デベロッパーガイド] の [Amazon SQS キューへの Amazon SNS メッセージの送信] と [Amazon SNS 通知を使った Lambda 関数の呼び出し] を参照してください。

次の例は、CodePipeline の承認で使用可能な JSON 出力の構造を示しています。

```
{
```

```
    "region": "us-east-2",
    "consoleLink": "https://console.aws.amazon.com/codepipeline/home?region=us-east-2#/
view/MyFirstPipeline",
    "approval": {
        "pipelineName": "MyFirstPipeline",
        "stageName": "MyApprovalStage",
        "actionName": "MyApprovalAction",
        "token": "1a2b3c4d-573f-4ea7-a67E-XAMPLETOKEN",
        "expires": "2016-07-07T20:22Z",
        "externalEntityLink": "http://example.com",
        "approvalReviewLink": "https://console.aws.amazon.com/codepipeline/
home?region=us-east-2#/view/MyFirstPipeline/MyApprovalStage/MyApprovalAction/
approve/1a2b3c4d-573f-4ea7-a67E-XAMPLETOKEN",
        "customData": "Review the latest changes and approve or reject within seven
 days."
    }
}
```

CodePipeline でクロスリージョンアクションを追加する

AWS CodePipeline には、自動リリースプロセスのリソースのビルド、テスト、デプロイの設定に役立つアクションが多数含まれています。パイプラインとは異なる AWS リージョンにあるアクションをパイプラインに追加できます。 AWS のサービス がアクションのプロバイダーであり、このアクションタイプ/プロバイダータイプがパイプラインとは異なる AWS リージョンにある場合、これはクロスリージョンアクションです。

> **Note**
>
> クロスリージョンアクションはサポートされており、CodePipeline がサポートされている AWS リージョンでのみ作成できます。CodePipeline でサポートされている AWS リージョンのリストについては、「」を参照してくださいAWS CodePipeline のクォータ。

コンソール、または を使用して AWS CLI、パイプラインにクロスリージョンアクション CloudFormation を追加できます。

> **Note**
>
> CodePipeline の特定のアクションタイプは、特定の AWS リージョンでのみ使用できます。また、アクションタイプは使用できるが、そのアクションタイプの特定の AWS プロバイダーは使用できない AWS リージョンがある場合もあります。

パイプラインを作成または編集する場合は、パイプラインリージョンにアーティファクトバケットが必要であり、アクションを実行する予定のリージョンごとに 1 つのアーティファクトバケットが必要です。`ArtifactStores` パラメータの詳細については、「CodePipeline パイプライン構造リファレンス」をご参照ください。

> **Note**
>
> CodePipeline は、クロスリージョンアクションを実行するときに、ある AWS リージョンから他のリージョンへのアーティファクトのコピーを処理します。

コンソールを使用してパイプラインまたはクロスリージョンアクションを作成する場合、デフォルトのアーティファクトバケットは、アクションがあるリージョンの CodePipeline によって設定されます。 AWS CLI、 CloudFormation、または SDK を使用してパイプラインまたはクロスリージョンアクションを作成する場合は、アクションがあるリージョンごとにアーティファクトバケットを指定します。

> **Note**
>
> アーティファクトバケットと暗号化キーは、クロスリージョンアクションと同じ AWS リージョンとパイプラインと同じアカウントで作成する必要があります。

以下のアクションタイプのクロスリージョンアクションは作成できません。

- ソースアクション
- サードパーティーアクション
- カスタムアクション

> **Note**
>
> CodePipeline でクロスリージョン Lambda 呼び出しアクションを使用する場合、PutJobSuccessResult と PutJobFailureResult を使用した Lambda 実行のステータスは、CodePipeline が存在する AWS リージョンではなく、Lambda 関数が存在するリージョンに送信する必要があります。

パイプラインにステージの一部としてクロスリージョンアクションが含まれている場合、CodePipeline はクロスリージョンアクションの入力アーティファクトのみをパイプラインリージョンからアクションのリージョンにレプリケートします。

> **Note**
>
> パイプラインリージョンと CloudWatch Events の変更検出リソースが保持されているリージョンは同じままです。パイプラインがホストされているリージョンは変わりません。

パイプライン内のクロスリージョンアクションを管理する (コンソール)

CodePipeline コンソールを使用して、既存のパイプラインにクロスリージョンアクションを追加できます。[パイプラインの作成] ウィザードを使用してクロスリージョンアクションを持つ新しいパイプラインを作成するには、「カスタムパイプラインを作成する (コンソール)」を参照してください。

コンソールで、アクションプロバイダーとリージョンフィールドを選択してパイプラインステージでクロスリージョンアクションを作成します。このフィールドには、そのプロバイダーのリージョンで作成したリソースが一覧表示されます。クロスリージョンアクションを追加すると、CodePipeline はアクションのリージョンで別のアーティファクトバケットを使用します。クロスリージョンのアーティファクトバケットの詳細については、「CodePipeline パイプライン構造リファレンス」を参照してください。

パイプラインステージにクロスリージョンアクションを追加する (コンソール)

コンソールを使用してパイプラインにクロスリージョンアクションを追加します。

> **Note**
>
> 変更を保存する際にパイプラインが実行中の場合、その実行は完了しません。

クロスリージョンアクションを追加するには

1. コンソール (http://console.aws.amazon.com/codesuite/codepipeline/home) にサインインします。
2. パイプラインを選択し、[編集] を選択します。
3. 図の下部で、新しいステージを追加する場合は [+ Add stage (+ ステージの追加)] を選択します。既存のステージにアクションを追加する場合は、[Edit stage (ステージの編集)] を選択します。
4. [Edit: <Stage> (編集: <ステージ>)] で、シリアルアクションを追加する場合は [+ Add action group (+ アクショングループの追加)] を選択します。または、パラレルアクションを追加する場合は、[+Add action (+アクションの追加)] を追加します。
5. [アクションの編集] ページで、以下の操作を行います。
   a. [アクション名] に、クロスリージョンアクションの名前を入力します。
   b. [Action provider (アクションプロバイダー)] で、アクションプロバイダーを選択します。
   c. リージョンで、アクションのリソースを作成または作成する予定の AWS リージョンを選択します。リージョンを選択すると、このリージョンで使用できるリソースが一覧表示されて選択できるようになります。リージョンフィールドは、このアクションタイプとプロバイダータイプに対して AWS リソースが作成される場所を指定します。このフィールドには、アクションプロバイダーが AWS のサービスであるアクションのみが表示されます。[リージョン] フィールドは、デフォルトで、パイプラインと同じ AWS リージョン になります。
   d. [入力アーティファクト] で、前のステージからの適切な入力を選択します。たとえば、前のステージがソースステージである場合は、[SourceArtifact] を選択します。
   e. 設定するアクションプロバイダーのすべての必須フィールドに入力します。
   f. [出力アーティファクト] で、次のステージへの適切な出力を選択します。たとえば、次のステージがデプロイステージである場合は、[BuildArtifact] を選択します。
   g. [保存] を選択します。
6. [Edit: <Stage> (編集: <ステージ>)] で、完了] を選択します。
7. [保存] を選択します。

パイプラインステージでクロスリージョンアクションを編集する (コンソール)

コンソールを使用してパイプラインの既存のクロスリージョンアクションを編集します。

> **Note**
> 変更を保存する際にパイプラインが実行中の場合、その実行は完了しません。

クロスリージョンアクションを編集するには

1. コンソール (http://console.aws.amazon.com/codesuite/codepipeline/home) にサインインします。
2. パイプラインを選択し、[編集] を選択します。
3. [Edit stage (ステージの編集)] を選択します。
4. [Edit: <Stage> (編集: <ステージ>)] で、既存のアクションを編集するためのアイコンを選択します。
5. [アクションの編集] ページで、必要に応じてフィールドを変更します。
6. [Edit: <Stage> (編集: <ステージ>)] で、完了] を選択します。
7. [保存] を選択します。

パイプラインステージからクロスリージョンアクションを削除する (コンソール)

コンソールを使用してパイプラインから既存のクロスリージョンアクションを削除します。

> **Note**
> 変更を保存する際にパイプラインが実行中の場合、その実行は完了しません。

クロスリージョンアクションを削除するには

1. コンソール (http://console.aws.amazon.com/codesuite/codepipeline/home) にサインインします。
2. パイプラインを選択し、[編集] を選択します。
3. [Edit stage (ステージの編集)] を選択します。
4. [Edit: <Stage> (編集: <ステージ>)] で、既存のアクションを削除するためのアイコンを選択します。
5. [Edit: <Stage> (編集: <ステージ>)] で、完了] を選択します。
6. [保存] を選択します。

クロスリージョンアクションをパイプラインに追加する (CLI)

を使用して AWS CLI 、既存のパイプラインにクロスリージョンアクションを追加できます。

を使用してパイプラインステージでクロスリージョンアクションを作成するには AWS CLI、オプション`region`フィールドとともに設定アクションを追加します。また、アクションの リージョンにアーティファクトバケットを作成済みである必要があります。単一リージョンパイプラインの `artifactStore` パラメータを指定する代わりに、`artifactStores` パラメータを使用して各リージョンのアーティファクトバケットのリストを含めます。

> **Note**
>
> このチュートリアルおよびその例では、*##### A* がパイプラインの作成先のリージョンです。このアカウントでは、CodePipeline で使用するパイプラインアーティファクトやサービスロールの保存先である *##### A* Amazon S3 バケットにアクセスできます。*RegionB* は、CodeDeploy で使用される CodeDeploy アプリケーション、デプロイグループ、およびサービスロールが作成されるリージョンです。

前提条件

以下を作成しておく必要があります。

- *##### A* のパイプライン。
- *##### B* の Amazon S3 アーティファクトバケット
- リージョン *RegionB* のクロスリージョンデプロイアクションの CodeDeploy アプリケーションやデプロイグループなど、アクションのリソース。

クロスリージョンアクションをパイプラインに追加する (CLI)

を使用して AWS CLI 、クロスリージョンアクションをパイプラインに追加します。

クロスリージョンアクションを追加するには

1. *##### A* のパイプラインで、get-pipeline コマンドを実行し、パイプライン構造を JSON ファイルにコピーします。例えば、`MyFirstPipeline` という名前のパイプラインに対して、以下のコマンドを実行します。

```
aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
```

このコマンドは何も返しませんが、作成したファイルは、コマンドを実行したディレクトリにあります。

2. region フィールドを追加して、アクションのリージョンとリソースを含むクロスリージョンアクションを持つ新しいステージを追加します。次の JSON サンプルは、プロバイダーが CodeDeploy であるクロスリージョンデプロイアクションを含むデプロイステージを新しいリージョン に追加しますus-east-1。

```
{
                "name": "Deploy",
                "actions": [
                    {
                        "inputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "name": "Deploy",
                        "region": "RegionB",
                        "actionTypeId": {
                            "category": "Deploy",
                            "owner": "AWS",
                            "version": "1",
                            "provider": "CodeDeploy"
                        },
                        "outputArtifacts": [],
                        "configuration": {
                            "ApplicationName": "name",
                            "DeploymentGroupName": "name"
                        },
                        "runOrder": 1
                    }
```

3. パイプライン構造で artifactStore フィールドを削除し、新しいクロスリージョンアクションの artifactStores を追加します。マッピングには、アクションがある各 AWS リージョンのエントリを含める必要があります。マッピングの各エントリについて、リソースはそれぞれの AWS リージョンにある必要があります。以下の例で、ID-A は、##### A は暗号化キー ID、ID-B は、##### B の暗号化キー ID を表します。

```
"artifactStores":{
   "RegionA":{
      "encryptionKey":{
         "id":"ID-A",
         "type":"KMS"
      },
      "location":"Location1",
      "type":"S3"
   },
   "RegionB":{
      "encryptionKey":{
         "id":"ID-B",
         "type":"KMS"
      },
      "location":"Location2",
      "type":"S3"
   }
}
```

以下の JSON 例では、us-west-2 バケットは `my-storage-bucket` と表示されており、`my-storage-bucket-us-east-1` という名前の新しい us-east-1 バケットを追加します。

```
        "artifactStores": {
            "us-west-2": {
                "type": "S3",
                "location": "my-storage-bucket"
            },
            "us-east-1": {
                "type": "S3",
                "location": "my-storage-bucket-us-east-1"
            }
        },
```

4. get-pipeline コマンドを使用して取得したパイプライン構造を使用している場合、JSON ファイルから `metadata` 行を削除します。それ以外の場合は、update-pipeline コマンドで使用することはできません。`"metadata": { }` 行と、`"created"`、`"pipelineARN"`、`"updated"` フィールドを削除します。

   例えば、構造から以下の行を削除します。

```
"metadata": {
```

```
  "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
  "created": "date",
  "updated": "date"
  }
```

ファイルを保存します。

5. 変更を適用するには、パイプライン JSON ファイルを指定して、update-pipeline コマンドを実行します。

> **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 codepipeline update-pipeline --cli-input-json file://pipeline.json
```

このコマンドは、編集したパイプラインの構造全体を返します。出力は以下のようになります。

```
{
    "pipeline": {
        "version": 4,
        "roleArn": "ARN",
        "stages": [
            {
                "name": "Source",
                "actions": [
                    {
                        "inputArtifacts": [],
                        "name": "Source",
                        "actionTypeId": {
                            "category": "Source",
                            "owner": "AWS",
                            "version": "1",
                            "provider": "CodeCommit"
                        },
                        "outputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "configuration": {
```

```
                        "PollForSourceChanges": "false",
                        "BranchName": "main",
                        "RepositoryName": "MyTestRepo"
                    },
                    "runOrder": 1
                }
            ]
        },
        {
            "name": "Deploy",
            "actions": [
                {
                    "inputArtifacts": [
                        {
                            "name": "SourceArtifact"
                        }
                    ],
                    "name": "Deploy",
                    "region": "us-east-1",
                    "actionTypeId": {
                        "category": "Deploy",
                        "owner": "AWS",
                        "version": "1",
                        "provider": "CodeDeploy"
                    },
                    "outputArtifacts": [],
                    "configuration": {
                        "ApplicationName": "name",
                        "DeploymentGroupName": "name"
                    },
                    "runOrder": 1
                }
            ]
        }
    ],
    "name": "AnyCompanyPipeline",
    "artifactStores": {
        "us-west-2": {
            "type": "S3",
            "location": "my-storage-bucket"
        },
        "us-east-1": {
            "type": "S3",
            "location": "my-storage-bucket-us-east-1"
```

```
            }
        }
    }
}
```

> **Note**
>
> update-pipeline コマンドは、パイプラインを停止します。update-pipeline コマンドを実行したときにパイプラインによりリビジョンが実行されている場合、その実行は停止します。更新されたパイプラインによりそのリビジョンを実行するには、パイプラインを手動で開始する必要があります。パイプラインを手動で開始するには **start-pipeline-execution** コマンドを使用します。

6. パイプラインを更新すると、クロスリージョンアクションがコンソールに表示されます。



クロスリージョンアクションをパイプラインに追加する (CloudFormation)

CloudFormation を使用して、既存のパイプラインにクロスリージョンアクションを追加できます。

を使用してクロスリージョンアクションを追加するには CloudFormation

1. この例に示すように、Region パラメータをテンプレートの ActionDeclaration リソースに追加します。

```
ActionDeclaration:
            Type: Object
            Properties:
```

```
                ActionTypeId:
                  Type: ActionTypeId
                  Required: true
                Configuration:
                  Type: Map
                InputArtifacts:
                  Type: Array
                  ItemType:
                    Type: InputArtifact
                Name:
                  Type: String
                  Required: true
                OutputArtifacts:
                  Type: Array
                  ItemType:
                    Type: OutputArtifact
                RoleArn:
                  Type: String
                RunOrder:
                  Type: Integer
                Region:
                  Type: String
```

2. Mappings で、この例で示しているように、キー SecondRegionMap および RegionA の値をマップする RegionB という名前のマッピング用のリージョンマップを追加します。Pipeline リソースの artifactStore フィールドで、新しいクロスリージョンアクションの artifactStores マップを次のように追加します。

```
Mappings:
  SecondRegionMap:
    RegionA:
      SecondRegion: "RegionB"
    RegionB:
      SecondRegion: "RegionA"

...

          Properties:
            ArtifactStores:
              -
                Region: RegionB
                ArtifactStore:
                  Type: "S3"
```

```
                Location: test-cross-region-artifact-store-bucket-RegionB
            -
              Region: RegionA
              ArtifactStore:
                Type: "S3"
                Location: test-cross-region-artifact-store-bucket-RegionA
```

以下の YAML 例では、*##### A* バケットを us-west-2、新しい *##### B* バケットを eu-central-1 とします。

```
Mappings:
  SecondRegionMap:
    us-west-2:
      SecondRegion: "eu-central-1"
    eu-central-1:
      SecondRegion: "us-west-2"

...

          Properties:
            ArtifactStores:
              -
                Region: eu-central-1
                ArtifactStore:
                  Type: "S3"
                  Location: test-cross-region-artifact-store-bucket-eu-central-1
              -
                Region: us-west-2
                ArtifactStore:
                  Type: "S3"
                  Location: test-cross-region-artifact-store-bucket-us-west-2
```

3. 更新したテンプレートをローカルコンピュータに保存し、 CloudFormation コンソールを開きます。
4. スタックを選択し、[既存スタックの変更セットの作成] を選択します。
5. テンプレートをアップロードし、 CloudFormationに示された変更を表示します。これらがスタックに加えられる変更です。新しいリソースがリストに表示されています。
6. [実行] を選択してください。

変数の操作

CodePipeline のアクションの中には、変数を生成するものがあります。変数を使用するには、次の手順に従います。

- 名前空間をアクションに割り当てて、生成する変数をダウンストリームアクション設定で使用できるようにします。
- ダウンストリームアクションは、アクションによって生成された変数を消費するよう設定します。

  各アクションの実行の詳細を表示して、実行時にアクションによって生成された各出力変数の値を確認できます。

変数を使用するためのステップバイステップの例を参照するには:

- アップストリームアクション (CodeCommit) の変数を使用し、出力変数を生成する Lambda アクションのチュートリアルについては、チュートリアル: Lambda 呼び出しアクションで変数を使用する を参照してください。
- アップストリーム CloudFormation CloudFormation アクションのスタック出力変数を参照するアクションを含むチュートリアルについては、「」を参照してくださいチュートリアル: AWS CloudFormation デプロイアクションの変数を使用するパイプラインを作成する。
- CodeCommit コミット ID とコミットメッセージに解決される出力変数を参照するメッセージテキストを使用した手動承認アクションの例については、例: 手動承認で変数を使用する を参照してください。
- GitHub ブランチ名に解決される環境変数を持つ CodeBuild アクションの例については、例:CodeBuild 環境変数で BranchName 変数を使用する を参照してください。
- CodeBuild アクションは、ビルドの一部としてエクスポートされたすべての環境変数を変数として生成します。詳細については、「CodeBuild アクションの出力変数」を参照してください。CodeBuild で使用できる環境変数のリストについては、AWS CodeBuild ユーザーガイドの「ビルド環境の環境変数」を参照してください。

トピック

- 変数のアクションを設定する
- 出力変数を表示する
- 例: 手動承認で変数を使用する
- 例:CodeBuild 環境変数で BranchName 変数を使用する

変数のアクションを設定する

パイプラインにアクションを追加すると、そのアクションに名前空間を割り当て、以前のアクションの変数を消費するように設定できます。

変数のアクションを設定する (コンソール)

この例では、CodeCommit ソースアクションと CodeBuild ビルドアクションを含むパイプラインを作成します。CodeBuild アクションは、 CodeCommit アクションによって生成された変数を消費するよう設定されています。

名前空間が指定されていない場合、変数はアクション設定で参照できません。コンソールを使用してパイプラインを作成すると、各アクションの名前空間が自動的に生成されます。

変数を使用してパイプラインを作成するには

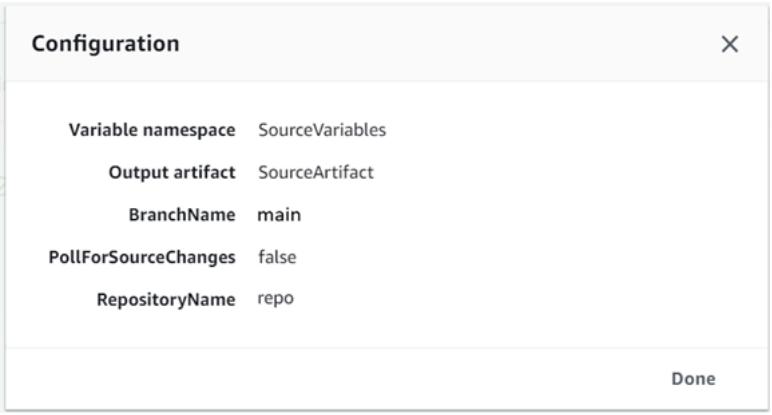
1. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. パイプラインの作成 を選択します。パイプラインに名前を入力し、[Next (次へ)] を選択します。
3. [ソース] の [プロバイダ] で、[CodeCommit] を選択します。ソースアクションの CodeCommit リポジトリとブランチを選択し、[Next (次へ)] を選択します。
4. [ビルド] の [プロバイダ] で [CodeBuild] を選択します。既存の CodeBuild ビルドプロジェクト名を選択するか、[Create project (プロジェクトを作成)] を選択します。[Create build project (ビルドプロジェクトを作成)] で、ビルドプロジェクトを作成し、[Return to CodePipeline (CodePipeline に戻る)] を選択します。

   [環境変数] の下で、[Add environment variables (環境変数を追加)] を選択します。例えば、実行 ID を変数構文 `#{codepipeline.PipelineExecutionId}` で入力し、コミット ID を変数構文 `#{SourceVariables.CommitId}` で入力します。

   > **Note**
   >
   > 変数構文は、ウィザードの任意のアクション設定フィールドに入力できます。

5. [作成] を選択します。

6. パイプラインが作成されたら、ウィザードによって作成された名前空間を表示できます。パイプラインで、名前空間を表示するステージのヘルプペインアイコンを選択します。この例では、ソースアクションの自動生成された名前空間、SourceVariables が表示されます。



既存のアクションの名前空間を編集するには

1. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. 編集するパイプラインを選択して [編集] を選択します。ソースステージで、[Edit stage (ステージを編集)] を選択します。CodeCommit アクションを追加します。
3. [アクションの編集] で、[Variable namespace (変数の名前空間)] フィールドを表示します。既存のアクションが以前に作成された場合、またはウィザードを使用せずに作成された場合は、名前空間を追加する必要があります。[Variable namespace (変数の名前空間)] に名前空間名を入力し、[Save (保存)] を選択します。

出力変数を表示するには

1. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. パイプラインが作成され、正常に実行できたら、アクションの実行詳細ページで変数の値を表示することもできます。詳細については、「変数を表示する (コンソール)」を参照してください。

変数のアクションを設定する (CLI)

create-pipeline コマンドを使用してパイプラインを作成するか、update-pipeline コマンドを使用してパイプラインを編集する場合は、アクションの設定で変数を参照したり、使用したりできます。

名前空間が指定されていない場合、アクションによって生成された変数は、ダウンストリームアクション設定で参照することはできません。

名前空間でアクションを設定するには

1. 「パイプライン、ステージ、アクションを作成する」の手順 に従って、CLI を使用してパイプラインを作成します。入力ファイルを起動して、create-pipeline コマンドに `--cli-input-json` パラメータを指定します。パイプライン構造で、`namespace` パラメータを追加し、`SourceVariables` などの名前を指定します。

```
. . .
{
          "inputArtifacts": [],
          "name": "Source",
          "region": "us-west-2",
          "namespace": "SourceVariables",
          "actionTypeId": {
            "category": "Source",
            "owner": "AWS",
            "version": "1",
            "provider": "CodeCommit"
          },
          "outputArtifacts": [

. . .
```

2. **MyPipeline.json** のような名前でファイルを保存します。
3. ターミナル (Linux、macOS、UNIX) またはコマンドプロンプト (Windows) で、create-pipelineコマンドを実行し、パイプラインを作成します。

   create-pipeline コマンドを実行したときに作成したファイルを呼び出します。例:

```
aws codepipeline create-pipeline --cli-input-json file://MyPipeline.json
```

変数を消費するダウンストリームアクションを設定するには

1. 入力ファイルを編集して、update-pipeline コマンドに `--cli-input-json` パラメータを指定します。ダウンストリームアクションで、そのアクションの設定に変数を追加します。変数は、ピリオドで区切られた名前空間とキーで構成されます。たとえば、パイプライン実行 ID とソースコミット ID の変数を追加するには、変数 `#{codepipeline.PipelineExecutionId}` に

変数の名前空間 codepipeline を指定します。変数の #{SourceVariables.CommitId} 名前空間 SourceVariables を指定します。

```
{
    "name": "Build",
    "actions": [
        {
            "outputArtifacts": [
                {
                    "name": "BuildArtifacts"
                }
            ],
            "name": "Build",
            "configuration": {
                "EnvironmentVariables": "[{\"name\":\"Execution_ID\",\"value
\":\"#{codepipeline.PipelineExecutionId}\",\"type\":\"PLAINTEXT\"},{\"name\":
\"Commit_ID\",\"value\":\"#{SourceVariables.CommitId}\",\"type\":\"PLAINTEXT\"}]",
                "ProjectName": "env-var-test"
            },
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-west-2",
            "actionTypeId": {
                "provider": "CodeBuild",
                "category": "Build",
                "version": "1",
                "owner": "AWS"
            },
            "runOrder": 1
        }
    ]
},
```

2. **MyPipeline.json** のような名前でファイルを保存します。
3. ターミナル (Linux、macOS、UNIX) またはコマンドプロンプト (Windows) で、create-pipelineコマンドを実行し、パイプラインを作成します。

   create-pipeline コマンドを実行したときに作成したファイルを呼び出します。例:

```
aws codepipeline create-pipeline --cli-input-json file://MyPipeline.json
```

出力変数を表示する

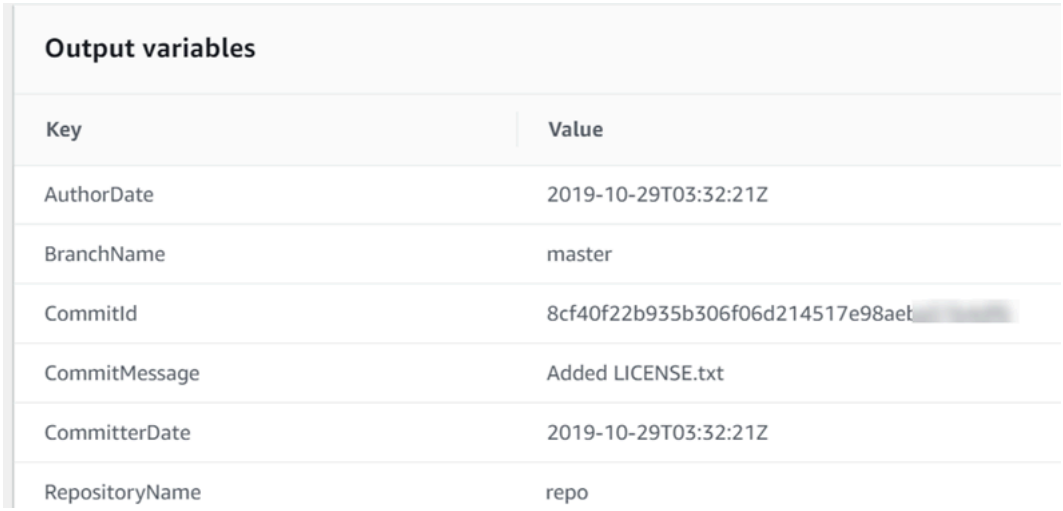
アクション実行の詳細を表示して、各実行に固有のアクションの変数を表示できます。

変数を表示する (コンソール)

コンソールを使用して、アクションの変数を表示できます。

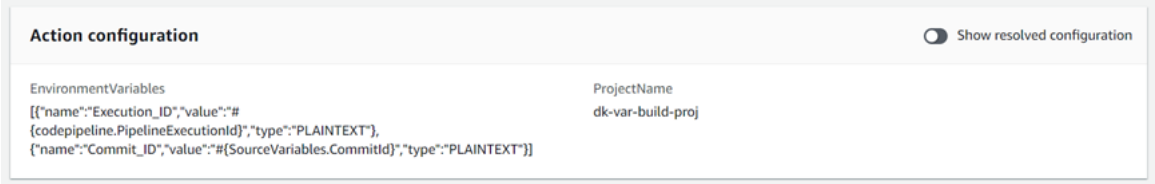
1. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。
2. [Name] で、パイプラインの名前を選択します。
3. [View history (履歴の表示)] を選択します。
4. パイプラインが正常に実行されると、ソースアクションによって生成された変数を表示できます。[View history (履歴の表示)] を選択します。パイプライン実行のアクションリストで [ソース] を選択して、CodeCommit アクションのアクション実行の詳細を表示します。アクションの詳細画面で、[Output variables (出力変数)] の下の変数を表示します。

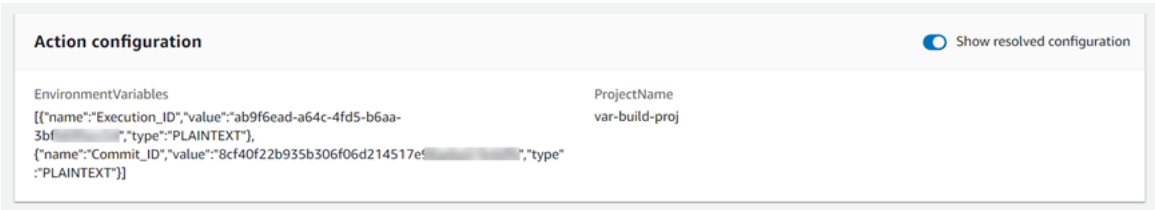


5. パイプラインが正常に実行されると、ビルドアクションによって消費される変数を表示できます。[View history (履歴の表示)] を選択します。パイプライン実行のアクションリストで、[ビルド] を選択して、CodeBuild アクションのアクション実行の詳細を表示します。アクションの詳

細ページで、[アクション設定] の下にある変数を表示します。自動生成された名前空間が表示されます。



デフォルトでは、[アクション設定] には変数の構文が表示されます。[Show resolved configuration (解決された設定を表示)] を選択して、アクションの実行中に生成された値を表示するようにリストを切り替えることができます。



変数を表示する (CLI)

list-action-executions コマンドを使用して、アクションの変数を表示できます。

1. 以下のコマンドを使用します。

```
aws codepipeline list-action-executions
```

出力には、次に示すように outputVariables パラメータが表示されます。

```
"outputVariables": {
                    "BranchName": "main",
                    "CommitMessage": "Updated files for test",
                    "AuthorDate": "2019-11-08T22:24:34Z",
                    "CommitId": "d99b0083cc10EXAMPLE",
                    "CommitterDate": "2019-11-08T22:24:34Z",
                    "RepositoryName": "variables-repo"
                },
```

2. 以下のコマンドを使用します。

```
aws codepipeline get-pipeline --name <pipeline-name>
```

CodeBuild アクションのアクション設定で、変数を表示できます。

```
{
    "name": "Build",
    "actions": [
        {
            "outputArtifacts": [
                {
                    "name": "BuildArtifact"
                }
            ],
            "name": "Build",
            "configuration": {
                "EnvironmentVariables": "[{\"name\":\"Execution_ID\",\"value
\":\"#{codepipeline.PipelineExecutionId}\",\"type\":\"PLAINTEXT\"},{\"name\":
\"Commit_ID\",\"value\":\"#{SourceVariables.CommitId}\",\"type\":\"PLAINTEXT\"}]",
                "ProjectName": "env-var-test"
            },
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-west-2",
            "actionTypeId": {
                "provider": "CodeBuild",
                "category": "Build",
                "version": "1",
                "owner": "AWS"
            },
            "runOrder": 1
        }
    ]
},
```

例: 手動承認で変数を使用する

アクションの名前空間を指定し、そのアクションが出力変数を生成する場合、承認メッセージに変数を表示する手動承認を追加できます。この例では、手動承認メッセージに変数構文を追加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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。承認を追加するパイプラインを選択します。
2. パイプラインを編集するには、[編集] を選択します。ソースアクションの後に、手動承認を追加します。[アクション名] に、承認処理の名前を入力します。
3. [アクションプロバイダ] で、[手動承認] を選択します。
4. [レビュー用の URL] で、`CommitId` の変数構文を CodeCommit URL に追加します。ソースアクションに割り当てられた名前空間をかならず使用してください。例えば、デフォルトの名前空間で `SourceVariables` CodeCommit アクションに対する変数構文は、`#{SourceVariables.CommitId}` です。

   [コメント] で、`CommitMessage` にコミットメッセージを入力します。

   ```
   Please approve this change. Commit message: #{SourceVariables.CommitMessage}
   ```
5. パイプラインが正常に実行されると、承認メッセージに変数の値を表示できます。

例:CodeBuild 環境変数で BranchName 変数を使用する

CodeBuild アクションをパイプラインに追加する場合、CodeBuild 環境変数を使用してアップストリームソースアクションから `BranchName` 変数を参照することができます。CodePipeline のアクションからの出力変数を使用して、ビルドコマンドで使用する独自の CodeBuild 環境変数を作成できます。

この例では、GitHub ソースアクションから出力変数構文を CodeBuild 環境変数に追加する方法を示します。この例の出力変数の構文は、以下 `BranchName` の GitHub ソースアクション出力変数を表します。アクションが正常に実行されると、変数が解決され、GitHub ブランチ名が表示されます。

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

   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。承認を追加するパイプラインを選択します。

2. パイプラインを編集するには、[編集] を選択します。CodeBuild アクションを含むステージで、[ステージの編集] を選択します。
3. アイコンを選択して CodeBuild アクションを編集します。
4. [編集アクション] ページの 環境変数 で、次のように入力します。

   - [名前]に、環境変数の名前を入力します。
   - [値]に、パイプライン出力変数の変数構文を入力します。これには、ソースアクションに割り当てられた名前空間が含まれます。たとえば、デフォルトの名前空間で `SourceVariables` GitHub アクションに対する出力変数構文は、`#{SourceVariables.BranchName}` です。
   - [タイプ]に、プレーンテキストを選択します。

5. パイプラインが正常に実行されると、解決された出力変数が環境変数の値になることがわかります。次のいずれかを選択します。

   - CodePipeline コンソール: パイプラインを選択してから、履歴 を選択します。最新のパイプライン実行を選択します。
     - [タイムライン]で、[送信元] のセレクタを選択します。これは GitHub 出力変数を生成するソースアクションです。View execution details (実行の詳細を表示)を選択 [出力変数]で、このアクションによって生成された出力変数のリストを表示します。
     - [タイムライン]で、[ビルド]のセレクタを選択します。これは、ビルドプロジェクトの CodeBuild 環境変数を指定するビルドアクションです。View execution details (実行の詳細を表示) を選択 [アクション設定]で、CodeBuild 環境変数を表示します。解決済み設定を表示 を選択。環境変数の値は解決済みです。GitHub ソースアクションから `BranchName` 変数を出力します。この例では、この値は `main` です。

       詳細については、「変数を表示する (コンソール)」を参照してください。
   - CodeBuild コンソール: ビルドプロジェクトを選択し、ビルド実行のリンクを選択します。[環境変数] の場合、解決された出力変数は CodeBuild 環境変数の値です。この例では、環境変数の値の [Name] は `BranchName` で、GitHub ソースアクションから [Value] 解決済み `BranchName` 出力変数を表示します。この例では、この値は `main` です。


Build logs
Phase details
Reports
Environment variables
Build details
Resource utilization
Name
Value
Type
BranchName
main
PLAINTEXT

CodePipeline でのステージ移行の操作

移行は、無効化または有効化できるパイプラインステージ間のリンクです。デフォルトでは有効化されています。無効化された移行を再度有効化すると、30 日が経過していない限り、パイプラインの残りのステージで最新リビジョンが実行されます。パイプラインを実行しても、新しい変更が検出されるか、手動でパイプラインを再実行する場合を除き、無効期間が 31 日以上の移行が再開されることはありません。

AWS CodePipeline コンソールまたは を使用して AWS CLI 、パイプラインのステージ間の移行を無効または有効にできます。

> **Note**
>
> 手動で承認するまでパイプラインの実行を停止するには、承認アクションを使用します。詳細については、「手動の承認アクションをステージに追加する」を参照してください。

トピック

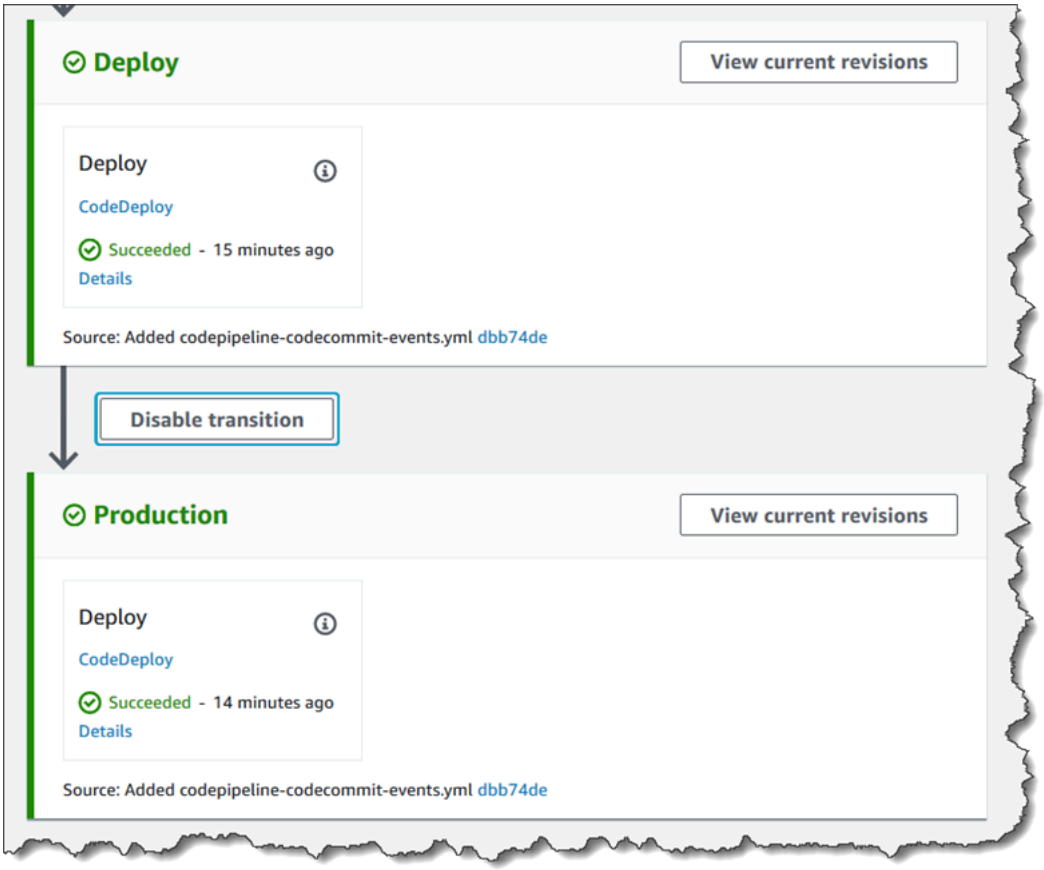
- 移行を無効化または有効化する (コンソール)
- 移行を無効化または有効化する (CLI)

移行を無効化または有効化する (コンソール)

パイプラインで遷移を無効/有効にするには

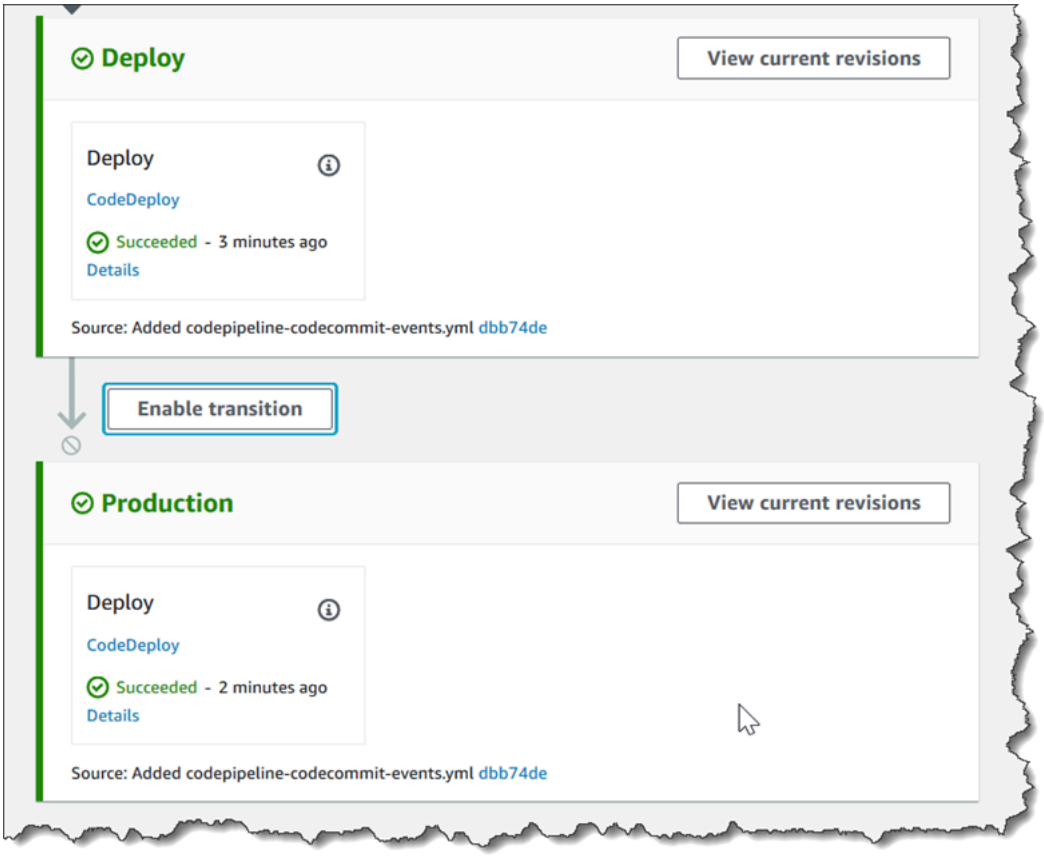
1. にサインイン AWS マネジメントコンソール し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。
2. [Name] で、遷移を有効または無効にするパイプラインの名前を選択します。これにより、パイプラインの詳細ビューが開いて、パイプラインのステージ間の遷移がわかります。
3. 実行する最後のステージの後の矢印を見つけ、その横にあるボタンを選択します。たとえば、以下のパイプラインの例で、[Staging (ステージング)] ステージのアクションを実行するが、[Production (本番稼働用)] という名前のステージのアクションを実行しないようにするには、その 2 つのステージの間の [Disable transition (移行を無効にする)] ボタンを選択します。



4. [移行を無効にする] ダイアログボックスで、移行を無効にする理由を入力し、[無効化] を選択します。

   ボタンは、矢印の前のステージと矢印の後のステージとの間で遷移が無効にされたことを示すように変わります。無効にされた移行の後のステージですでに実行されていたリビジョンは、パイプラインにより続行されますが、それ以降のリビジョンは、無効にされた移行の後には続行されません。



5. 矢印の横にある [移行を有効にする] ボタンを選択します。[Enable transition] ダイアログボックスで、[Enable] を選択します。パイプラインはすぐに 2 つのステージ間の移行を有効にします。移行が無効にされた後、初期ステージで実行されていたリビジョンがある場合、パイプラインはすぐに、以前の無効にされた移行の後のステージで最新のリビジョンの実行を開始します。パイプラインは、そのリビジョンをパイプラインの残りのすべてのステージで実行します。

> **Note**
>
> 遷移を有効にした後、CodePipeline コンソールに変更が表示されるまで数秒かかることがあります。

移行を無効化または有効化する (CLI)

を使用してステージ間の移行を無効にするには AWS CLI、 disable-stage-transition コマンドを実行します。無効にされた遷移を有効にするには、enable-stage-transition コマンドを実行します。

遷移を無効にするには

1. ターミナル (Linux、macOS、または Unix) またはコマンドプロンプト (Windows) を開き、AWS CLI を使用して disable-stage-transition コマンドを実行し、パイプラインの名前、遷移を無効にするステージの名前、遷移タイプ、そのステージへの移行を無効にする理由を指定します。コンソールを使用する場合とは異なり、ステージに入る (インバウンド) 遷移を無効にするか、すべてのアクションが完了した後にステージから出る (アウトバウンド) 遷移を無効にするかを指定する必要もあります。

   たとえば、*Staging* という名前のパイプラインで *MyFirstPipeline* という名前のステージへの遷移を無効にするには、以下のようなコマンドを入力します。

   ```
   aws codepipeline disable-stage-transition --pipeline-name MyFirstPipeline --stage-
   name Staging --transition-type Inbound --reason "My Reason"
   ```

   このコマンドは何も返しません。
2. 遷移が無効にされたことを確認するには、CodePipeline コンソールでパイプラインを表示するか、get-pipeline-state コマンドを実行します。詳細については、「パイプラインを表示する (コンソール)」および「パイプラインの詳細と履歴を表示する (CLI)」を参照してください。

遷移を有効にするには

1. ターミナル (Linux、macOS、または Unix) またはコマンドプロンプト (Windows) を開き、AWS CLI を使用して enable-stage-transition コマンドを実行し、パイプラインの名前、遷移を有効にするステージの名前、遷移タイプを指定します。

   たとえば、*Staging* という名前のパイプラインで *MyFirstPipeline* という名前のステージへの遷移を無効にするには、以下のようなコマンドを入力します。

   ```
   aws codepipeline enable-stage-transition --pipeline-name MyFirstPipeline --stage-
   name Staging  --transition-type Inbound
   ```

   このコマンドは何も返しません。
2. 移行が有効になっていることを確認するには、CodePipeline コンソールでパイプラインを表示するか、 get-pipeline-state コマンドを実行します。詳細については、「パイプラインを表示する (コンソール)」および「パイプラインの詳細と履歴を表示する (CLI)」を参照してください。

パイプラインのモニタリング

モニタリングは、AWS CodePipeline 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、モニタリングを開始する前に、以下の質問に回答するモニタリング計画を作成する必要があります。

- どのような目的でモニタリングしますか?
- 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が使用可能ですか?
- 誰がモニタリングタスクを実行しますか?
- 問題が発生した場合、だれが通知を受け取りますか?

次のツールを使用して、CodePipeline パイプラインおよびリソースをモニタリングできます。

- EventBridge イベントバスイベント — EventBridge で CodePipeline イベントを監視できます。これにより、パイプライン、ステージ、またはアクションの実行ステータスの変化が検出されます。EventBridge は、そのデータを AWS Lambda や Amazon Simple Notification Service などのターゲットにルーティングします。EventBridge のイベントは、Amazon CloudWatch Events に表示されるイベントと同じです。
- 「デベロッパー向けツールコンソール」でのパイプラインイベントの通知 — コンソールで設定した通知を使用して CodePipeline イベントを監視し、Amazon 簡易通知サービスのトピックとサブスクリプションを作成できます。詳細については、デベロッパー用ツールコンソールユーザーガイドの「通知とは何ですか」を参照してください。
- AWS CloudTrail - CloudTrail を使用して、AWS アカウントで CodePipeline によって、またはそれに代わって行われた API コールを取得し、Amazon S3 バケットにログファイルを配信します。新しいログファイルが配信されたときに CloudWatch が Amazon SNS 通知を発行するように選択することで、すばやいアクションが実行できるようにします。
- コンソールと CLI - CodePipeline コンソールと CLI を使用して、パイプラインのステータスまたは特定のパイプライン実行の詳細を表示できます。

トピック

- CodePipeline イベントのモニタリング

- イベントのプレースホルダーバケットに関するリファレンス
- を使用した CodePipeline API コールのログ記録 AWS CloudTrail
- CodePipeline CloudWatch メトリクス

CodePipeline イベントのモニタリング

EventBridge で CodePipeline イベントをモニタリングして、独自のアプリケーション、software-as-a-service (SaaS) アプリケーション、および からリアルタイムデータのストリームを配信できます AWS のサービス。 EventBridge EventBridge は、そのデータを AWS Lambda や Amazon Simple Notification Service などのターゲットにルーティングします。これらのイベントは、Amazon CloudWatch Events に表示されるイベントと同じで、 AWS リソースの変更を記述するシステムイベントのほぼリアルタイムのストリームを提供します。詳細については、Amazon EventBridge ユーザーガイドの「What Is Amazon EventBridge?」を参照してください。

> **Note**
>
> イベントを管理するには、Amazon EventBridge が好ましい方法です。Amazon CloudWatch Events と EventBridge は同じ基盤となるサービスと API ですが、EventBridge はより多くの機能を提供します。CloudWatch Events または EventBridge のいずれかで行った変更は、各コンソールに表示されます。

イベントはルールで構成されています。ルールは、次のものを選択して設定します:

- イベントパターン。各ルールは、モニタリングするイベントのソースとタイプ、およびイベントターゲットを含むイベントパターンとして表現されます。イベントをモニタリングするには、モニタリングするサービスを CodePipeline などのイベントソースとしてルールを作成します。例えば、CodePipeline をイベントソースとして使用するイベントパターンを使用してルールを作成し、パイプライン、ステージ、またはアクションの状態に変更があった場合にルールをトリガーできます。
- ターゲット。新しいルールは選択したサービスをイベントターゲットとして受け取ります。ターゲットサービスを設定することで、通知を送り、状態情報を取得し、是正措置を講じ、イベントを開始し、その他のアクションを実行します。ターゲットを追加するときには、選択したターゲットサービスを呼び出すためのアクセス許可も EventBridge に付与する必要があります。

各タイプの実行の状態変更イベントは、以下の特定のメッセージ内容で通知を送信します。

- 最初の `version` エントリは、イベントのバージョン番号を示します。
- パイプライン `detail` の `version` エントリは、パイプライン構造のバージョン番号を示します。
- パイプライン `detail` の `execution-id` エントリは、状態変更の原因となったパイプラインの実行 ID を示します。[GetPipelineExecution API リファレンス] の [AWS CodePipeline API コール] を参照してください。
- `pipeline-execution-attempt` エントリは、特定の実行 ID に対する試行回数、または再試行回数を示しています。

CodePipeline は、 AWS アカウント 内のリソースの状態が変わるたびに、EventBridge にイベントをレポートします。イベントは、以下のリソースに対して、少なくとも 1 回保証ベースで発行されます:

- パイプライン実行
- ステージ実行
- アクション実行

イベントは、上記のイベントパターンとスキーマの詳細を使用して EventBridge によって発行されます。デベロッパーツールのコンソールで設定した通知を通じて受け取るイベントなど、処理されたイベントの場合、イベントメッセージにはいくつかのバリエーションを持つイベントパターンフィールドが含まれます。例えば、`detail-type` フィールドは `detailType` に変換されます。詳細については、Amazon EventBridge API リファレンスの「PutEvents API コール」を参照してください。

次の例は、CodePipeline のイベントを示しています。可能な場合、各例は生成されたイベントのスキーマと、処理されたイベントのスキーマを示しています 。

トピック

- 詳細タイプ
- パイプラインレベルのイベント
- ステージレベルのイベント
- アクションレベルのイベント
- パイプラインイベントで通知を送信するルールを作成する

詳細タイプ

モニタリングするイベントを設定する場合、イベントの詳細タイプを選択できます。

以下の状態が変わった時に通知が送信されるように設定できます。

- 指定したパイプラインまたはすべてのパイプライン。これを制御するには、"detail-type": "CodePipeline Pipeline Execution State Change" を使用します。
- 指定したパイプラインまたはすべてのパイプライン内の、指定したステージまたはすべてのステージ。これを制御するには、"detail-type": "CodePipeline Stage Execution State Change" を使用します。
- 指定したパイプラインまたはすべてのパイプライン内の、指定したステージまたはすべてのステージ内の、指定したアクションまたはすべてのアクション。これを制御するには、"detail-type": "CodePipeline Action Execution State Change" を使用します。

> **Note**
>
> EventBridge によって発行されるイベントには、detail-type パラメータが含まれ、イベントが処理される際に、detailType に変換されます。

Detail-type	State	説明
CodePipeline パイプライン実行の状態変更	CANCELED	パイプライン構造が更新されたため、パイプライン実行がキャンセルされました。
	FAILED	パイプライン実行が正常に完了しませんでした。
	再開	失敗したパイプライン実行が RetryStageExecution API コールに応じて再試行されました。
	開始	パイプライン実行が現在実行中です。
	停止	停止プロセスが完了し、パイプラインの実行が停止します。
	停止中	パイプライン実行は、パイプライン実行を [Stop and wait (停止して待機)] するか、[Stop and abandon (停止して中止)] する要求により、停止しています。
	成功	パイプライン実行が正常に完了しました。

Detail-type	State	説明
	置き換え済み	このパイプライン実行が次のステージの完了を待機している間に、新しいパイプライン実行が進行し、代わりにパイプラインを継続しました。
CodePipeline ステージ実行の状態変更	CANCELED	パイプライン構造が更新されたため、ステージはキャンセルされました。
	FAILED	ステージは正常に完了しませんでした。
	再開	失敗したステージが `RetryStageExecution` API コールに応じて再試行されました。
	開始	このステージは現在実行中です。
	停止	停止プロセスが完了し、ステージの実行が停止します。
	停止中	ステージ実行は、パイプライン実行を [Stop and wait (停止して待機)] するか、[Stop and abandon (停止して中止)] する要求により、停止しています。
	成功	ステージは正常に完了しました。
CodePipeline アクション実行の状態変更	中止	パイプラインの実行を [Stop and abandon (停止して中止)] する要求により、アクションは中止されます。
	CANCELED	パイプライン構造が更新されたため、アクションはキャンセルされました。
	FAILED	承認アクションでは、失敗の状態は、アクションがレビュー者により拒否されたか、または誤ったアクション設定のために失敗したことを意味します。
	開始	アクションは現在実行中です。
	成功	アクションは正常に完了しました。

パイプラインレベルのイベント

パイプラインレベルのイベントは、パイプライン実行の状態が変更されたときに発生します。

トピック

- パイプライン開始イベント
- パイプライン停止イベント
- パイプライン成功イベント
- パイプラインが成功しました (Git タグを使用した例)
- パイプライン失敗イベント
- パイプラインが失敗しました (Git タグを使用した例)

パイプライン開始イベント

パイプライン実行が開始すると、以下の内容の通知が送信されます。この例は、"myPipeline" リージョンの us-east-1 という名前のパイプラインです。id フィールドはイベント ID を表し、account フィールドは、パイプラインが作成されるアカウント ID を表します。

Emitted event

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Pipeline Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-24T22:03:07Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "execution-trigger": {
            "trigger-type": "StartPipelineExecution",
            "trigger-detail": "arn:aws:sts::123456789012:assumed-role/Admin/my-user"
        },
```

```
        "state": "STARTED",
        "version": 1.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Pipeline Execution State Change",
    "region": "us-east-1",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:44:50Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
east-1:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "execution-trigger": {
            "trigger-type": "StartPipelineExecution",
            "trigger-detail": "arn:aws:sts::123456789012:assumed-role/Admin/my-user"
        },
        "state": "STARTED",
        "version": 1.0,
        "pipeline-execution-attempt": 1.0
    },
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "additionalAttributes": {}
}
```

パイプライン停止イベント

パイプライン実行が停止すると、以下の内容の通知が送信されます。この例は、myPipeline リージョンの us-west-2 という名前のパイプラインです。

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
```

```
    "detail-type": "CodePipeline Pipeline Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-24T22:02:20Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "state": "STOPPING",
        "version": 3.0,
        "pipeline-execution-attempt": 1.0
        "stop-execution-comments": "Stopping the pipeline for an update"
    }
}
```

パイプライン成功イベント

パイプライン実行が成功すると、以下の内容の通知が送信されます。この例は、myPipeline us-east-1 リージョンの という名前のパイプラインです。

Emitted event

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Pipeline Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-24T22:03:44Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "state": "SUCCEEDED",
        "version": 3.0,
```

```
        "pipeline-execution-attempt": 1.0
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Pipeline Execution State Change",
    "region": "us-east-1",
    "source": "aws.codepipeline",
    "time": "2021-06-30T22:13:51Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "state": "SUCCEEDED",
        "version": 1.0,
        "pipeline-execution-attempt": 1.0
    },
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "additionalAttributes": {}
}
```

パイプラインが成功しました (Git タグを使用した例)

パイプラインの実行が再試行され、成功したステージがあると、以下の内容の通知を送信するイベントが発生します。この例は、eu-central-1 リージョンの myPipeline という名前のパイプラインのもので、execution-trigger が Git タグに設定されています。

> **Note**
>
> execution-trigger} フィールドには、パイプラインをトリガーしたイベントの種類に応じて、tag-name または branch-name のいずれかが表示されます。

```
{
    "version": "0",
    "id": "b128b002-09fd-4574-4eba-27152726c777",
    "detail-type": "CodePipeline Pipeline Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2023-10-26T13:50:53Z",
    "region": "eu-central-1",
    "resources": [
        "arn:aws:codepipeline:eu-central-1:123456789012:BuildFromTag"
    ],
    "detail": {
        "pipeline": "BuildFromTag",
        "execution-id": "e17b5773-cc0d-4db2-9ad7-594c73888de8",
        "start-time": "2023-10-26T13:49:39.208Z",
        "execution-trigger": {
            "author-display-name": "Mary Major",
            "full-repository-name": "mmajor/sample-project",
            "provider-type": "GitLab",
            "author-email": "email_address",
            "commit-message": "Update file README.md",
            "author-date": "2023-08-16T21:08:08Z",
            "tag-name": "gitlab-v4.2.1",
            "commit-id": "commit_ID",
            "connection-arn": "arn:aws:codestar-connections:eu-
central-1:123456789012:connection/0f5b706a-1a1d-46c5-86b6-f177321bcfb2",
            "author-id": "Mary Major"
        },
        "state": "SUCCEEDED",
        "version": 32.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

パイプライン失敗イベント

パイプライン実行が成功すると、以下の内容の通知が送信されます。この例は、"myPipeline" リージョンの us-west-2 という名前のパイプラインです。

Emitted event

```
{
    "version": "0",
```

```
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Pipeline Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-31T18:55:43Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "state": "FAILED",
        "version": 4.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Pipeline Execution State Change",
    "region": "us-west-2",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:46:16Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "state": "FAILED",
        "version": 1.0,
        "pipeline-execution-attempt": 1.0
    },
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "additionalAttributes": {
        "failedActionCount": 1,
        "failedActions": [
```

```
            {
                "action": "Deploy",
                "additionalInformation": "Deployment <ID> failed"
            }
        ],
        "failedStage": "Deploy"
    }
```

パイプラインが失敗しました (Git タグを使用した例)

トリガーで設定されたパイプラインがソースステージで失敗しない限り、以下の内容の通知を送信するイベントが発生します。この例は、eu-central-1 リージョンの myPipeline という名前のパイプラインのもので、execution-trigger が Git タグに設定されています。

> **Note**
>
> execution-trigger} フィールドには、パイプラインをトリガーしたイベントの種類に応じて、tag-name または branch-name のいずれかが表示されます。

Emitted event

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Pipeline Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-31T18:55:43Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "execution-trigger": {
            "author-display-name": "Mary Major",
            "full-repository-name": "mmajor/sample-project",
            "provider-type": "GitLab",
```

```
            "author-email": "email_address",
            "commit-message": "Update file README.md",
            "author-date": "2023-08-16T21:08:08Z",
            "tag-name": "gitlab-v4.2.1",
            "commit-id": "commit_ID",
            "connection-arn": "arn:aws:codestar-connections:eu-
central-1:123456789012:connection/0f5b706a-1a1d-46c5-86b6-f177321bcfb2",
            "author-id": "Mary Major"
        },
        "state": "FAILED",
        "version": 4.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Pipeline Execution State Change",
    "region": "us-west-2",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:46:16Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "execution-trigger": {
            "author-display-name": "Mary Major",
            "full-repository-name": "mmajor/sample-project",
            "provider-type": "GitLab",
            "author-email": "email_address",
            "commit-message": "Update file README.md",
            "author-date": "2023-08-16T21:08:08Z",
            "tag-name": "gitlab-v4.2.1",
            "commit-id": "commit_ID",
            "connection-arn": "arn:aws:codestar-connections:eu-
central-1:123456789012:connection/0f5b706a-1a1d-46c5-86b6-f177321bcfb2",
            "author-id": "Mary Major"
        },
        "state": "FAILED",
```

```
        "version": 1.0,
        "pipeline-execution-attempt": 1.0
    },
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "additionalAttributes": {
        "failedActionCount": 1,
        "failedActions": [
            {
                "action": "Deploy",
                "additionalInformation": "Deployment <ID> failed"
            }
        ],
        "failedStage": "Deploy"
    }
```

ステージレベルのイベント

ステージレベルのイベントは、ステージ実行の状態が変更されたときに発生します。

トピック

- ステージ開始イベント
- ステージ停止イベント
- ステージ停止イベント
- ステージ再試行後のステージ再開イベント

ステージ開始イベント

ステージ実行が開始すると、以下の内容の通知が送信されます。この例は、"myPipeline" ステージの us-east-1 リージョンの Prod という名前のパイプラインです。

Emitted event

```
{
    "version": "0",
    "id": 01234567-EXAMPLE,
    "detail-type": "CodePipeline Stage Execution State Change",
    "source": "aws.codepipeline",
```

```
    "account": 123456789012,
    "time": "2020-01-24T22:03:07Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "version": 1.0,
        "execution-id": 12345678-1234-5678-abcd-12345678abcd,
        "start-time": "2023-10-26T13:49:39.208Z",
        "stage": "Prod",
        "state": "STARTED",
        "pipeline-execution-attempt": 1.0
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Stage Execution State Change",
    "region": "us-east-1",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:45:40Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "stage": "Source",
        "state": "STARTED",
        "version": 1.0,
        "pipeline-execution-attempt": 0.0
    },
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "additionalAttributes": {
        "sourceActions": [
            {
                "sourceActionName": "Source",
```

```
                "sourceActionProvider": "CodeCommit",
                "sourceActionVariables": {
                    "BranchName": "main",
                    "CommitId": "<ID>",
                    "RepositoryName": "my-repo"
                }
            }
        ]
    }
}
```

ステージ停止イベント

ステージ実行が停止すると、以下の内容の通知が送信されます。この例は、myPipeline ステージの us-west-2 リージョンの Deploy という名前のパイプラインです。

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Stage Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-24T22:02:20Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "stage": "Deploy",
        "state": "STOPPING",
        "version": 3.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

ステージ停止イベント

ステージ実行が停止すると、以下の内容の通知が送信されます。この例は、myPipeline ステージの us-west-2 リージョンの Deploy という名前のパイプラインです。

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Stage Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-31T18:21:39Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "stage": "Deploy",
        "state": "STOPPED",
        "version": 3.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

ステージ再試行後のステージ再開イベント

ステージ実行が再開され、ステージが再試行されると、以下の内容の通知を送信するイベントが発生します。

ステージが再試行されると、例に示すように `stage-last-retry-attempt-time` フィールドが表示されます。再試行が行われた場合、そのフィールドはすべてのステージイベントで表示されます。

> **Note**
>
> `stage-last-retry-attempt-time` フィールドは、ステージが再試行された後、以降のすべてのステージイベントに存在します。

```
{
    "version": "0",
    "id": "38656bcd-a798-5f92-c738-02a71be484e1",
    "detail-type": "CodePipeline Stage Execution State Change",
    "source": "aws.codepipeline",
```

```
    "account": "123456789012",
    "time": "2023-10-26T14:14:56Z",
    "region": "eu-central-1",
    "resources": [
        "arn:aws:codepipeline:eu-central-1:123456789012:BuildFromTag"
    ],
    "detail": {
        "pipeline": "BuildFromTag",
        "execution-id": "05dafb6a-5a56-4951-a858-968795364846",
        "stage-last-retry-attempt-time": "2023-10-26T14:14:56.305Z",
        "stage": "Build",
        "state": "RESUMED",
        "version": 32.0,
        "pipeline-execution-attempt": 2.0
    }
}
```

アクションレベルのイベント

アクションレベルのイベントは、アクション実行の状態が変更されたときに発生します。

トピック

- アクション開始イベント
- アクション成功イベント
- アクション失敗イベント
- アクション放棄イベント

アクション開始イベント

アクション実行が開始すると、以下の内容の通知が送信されます。この例は、デプロイアクション myPipeline の us-east-1 リージョンの myAction という名前のパイプラインです。

Emitted event

```
{
    "version": "0",
    "id": 01234567-EXAMPLE,
    "detail-type": "CodePipeline Action Execution State Change",
    "source": "aws.codepipeline",
    "account": 123456789012,
```

```
    "time": "2020-01-24T22:03:07Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": 12345678-1234-5678-abcd-12345678abcd,
        "start-time": "2023-10-26T13:51:09.981Z",
        "stage": "Prod",
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "myAction",
        "state": "STARTED",
        "type": {
            "owner": "AWS",
            "category": "Deploy",
            "provider": "CodeDeploy",
            "version": "1"
        },
        "version": 2.0
        "pipeline-execution-attempt": 1.0
        "input-artifacts": [
            {
                "name": "SourceArtifact",
                "s3location": {
                    "bucket": "codepipeline-us-east-1-BUCKETEXAMPLE",
                    "key": "myPipeline/SourceArti/KEYEXAMPLE"
                }
            }
        ]
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Action Execution State Change",
    "region": "us-west-2",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:45:44Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:123456789012:notificationrule/a69c62c21EXAMPLE",
```

```
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:51:09.981Z",
        "stage": "Deploy",
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "Deploy",
        "input-artifacts": [
            {
                "name": "SourceArtifact",
                "s3location": {
                    "bucket": "codepipeline-us-east-1-EXAMPLE",
                    "key": "myPipeline/SourceArti/EXAMPLE"
                }
            }
        ],
        "state": "STARTED",
        "region": "us-east-1",
        "type": {
            "owner": "AWS",
            "provider": "CodeDeploy",
            "category": "Deploy",
            "version": "1"
        },
        "version": 1.0,
        "pipeline-execution-attempt": 1.0
    },
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "additionalAttributes": {}
}
```

アクション成功イベント

アクション実行が成功すると、以下の内容の通知が送信されます。この例は、ソースアクション "myPipeline" の us-west-2 リージョンの "Source" という名前のパイプラインです。このイベントタイプには、2 つの異なる region フィールドがあります。イベント region フィールドは、パイプラインイベントのリージョンを指定します。region セクション下の detail フィールドは、アクションのリージョンを指定します。

Emitted event


```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Action Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-24T22:03:11Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:51:09.981Z",
        "stage": "Source",
        "execution-result": {
            "external-execution-url": "https://us-west-2.console.aws.amazon.com/
codecommit/home#/repository/my-repo/commit/8cf40f2EXAMPLE",
            "external-execution-summary": "Added LICENSE.txt",
            "external-execution-id": "8cf40fEXAMPLE"
        },
        "output-artifacts": [
            {
                "name": "SourceArtifact",
                "s3location": {
                    "bucket": "codepipeline-us-west-2-BUCKETEXAMPLE",
                    "key": "myPipeline/SourceArti/KEYEXAMPLE"
                }
            }
        ],
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "Source",
        "state": "SUCCEEDED",
        "region": "us-west-2",
        "type": {
            "owner": "AWS",
            "provider": "CodeCommit",
            "category": "Source",
            "version": "1"
        },
        "version": 3.0,
```

```
        "pipeline-execution-attempt": 1.0
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Action Execution State Change",
    "region": "us-west-2",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:45:44Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:ACCOUNT:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "arn:aws:codepipeline:us-west-2:123456789012:myPipeline",
        "start-time": "2023-10-26T13:51:09.981Z",
        "stage": "Source",
        "execution-result": {
            "external-execution-url": "https://us-west-2.console.aws.amazon.com/
codecommit/home#/repository/my-repo/commit/8cf40f2EXAMPLE",
            "external-execution-summary": "Edited index.html",
            "external-execution-id": "36ab3ab7EXAMPLE"
        },
        "output-artifacts": [
            {
                "name": "SourceArtifact",
                "s3location": {
                    "bucket": "codepipeline-us-west-2-EXAMPLE",
                    "key": "myPipeline/SourceArti/EXAMPLE"
                }
            }
        ],
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "Source",
        "state": "SUCCEEDED",
        "region": "us-west-2",
        "type": {
            "owner": "AWS",
            "provider": "CodeCommit",
            "category": "Source",
            "version": "1"
```

```
        },
        "version": 1.0,
        "pipeline-execution-attempt": 1.0
    },
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "additionalAttributes": {}
}
```

アクション失敗イベント

アクション実行が成功すると、以下の内容の通知が送信されます。この例は、"myPipeline" アクションの us-west-2 リージョンの "Deploy" という名前のパイプラインです。

Emitted event

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Action Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-31T18:55:43Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:51:09.981Z",
        "stage": "Deploy",
        "execution-result": {
            "external-execution-url": "https://us-west-2.console.aws.amazon.com/
codedeploy/home?#/deployments/<ID>",
            "external-execution-summary": "Deployment <ID> failed",
            "external-execution-id": "<ID>",
            "error-code": "JobFailed"
        },
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "Deploy",
```

```
        "state": "FAILED",
        "region": "us-west-2",
        "type": {
            "owner": "AWS",
            "provider": "CodeDeploy",
            "category": "Deploy",
            "version": "1"
        },
        "version": 4.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Action Execution State Change",
    "region": "us-west-2",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:46:16Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "stage": "Deploy",
        "execution-result": {
            "external-execution-url": "https://console.aws.amazon.com/codedeploy/
home?region=us-west-2#/deployments/<ID>",
            "external-execution-summary": "Deployment <ID> failed",
            "external-execution-id": "<ID>",
            "error-code": "JobFailed"
        },
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "Deploy",
        "state": "FAILED",
        "region": "us-west-2",
        "type": {
            "owner": "AWS",
            "provider": "CodeDeploy",
            "category": "Deploy",
            "version": "1"
```

```
        },
        "version": 13.0,
        "pipeline-execution-attempt": 1.0
    },
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "additionalAttributes": {
        "additionalInformation": "Deployment <ID> failed"
    }
}
```

アクション放棄イベント

アクション実行が放棄されると、以下の内容の通知が送信されます。この例は、"myPipeline" アクションの us-west-2 リージョンの "Deploy" という名前のパイプラインです。

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Action Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-31T18:21:39Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "stage": "Deploy",
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "Deploy",
        "state": "ABANDONED",
        "region": "us-west-2",
        "type": {
            "owner": "AWS",
            "provider": "CodeDeploy",
            "category": "Deploy",
            "version": "1"
        },
```

```
        "version": 3.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

パイプラインイベントで通知を送信するルールを作成する

ルールは特定のイベントを監視し、選択した AWS ターゲットにルーティングします。別の AWS アクションが発生したときに AWS アクションを自動的に実行するルール、または設定されたスケジュールで AWS アクションを定期的に実行するルールを作成できます。

トピック

- パイプラインの状態が変わる場合、通知を送信する (コンソール)
- パイプラインの状態が変わる場合、通知を送信する (CLI)

パイプラインの状態が変わる場合、通知を送信する (コンソール)

以下の手順では、EventBridge コンソールを使用して、CodePipeline で変更の通知を送信するためのルールを作成する方法を示します。

Amazon S3 ソースを使用するパイプラインをターゲットとする EventBridge ルールを作成するには

1. Amazon EventBridge コンソール (https://console.aws.amazon.com/events/) を開きます。
2. ナビゲーションペインで [ルール] を選択します。デフォルトのバスを選択したままにするか、イベントバスを選択します。[ルールを作成] を選択します。
3. [名前] で、ルールの名前を入力します。
4. [ルールタイプ] で、[イベントパターンを持つルール] を選択します。[次へ] を選択します。
5. [イベントパターン] で、[AWS のサービス] を選択します。
6. [イベントタイプ] ドロップダウンリストで、通知する状態変更のレベルを選択します。
   - パイプラインイベントに適用されるルールでは、[CodePipeline Pipeline Execution State Change] を選択します。
   - ステージレベルのイベントに適用されるルールでは、[CodePipeline Stage Execution State Change] を選択します。
   - アクションレベルのイベントに適用されるルールでは、[CodePipeline Action Execution State Change] を選択します。

7. 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には、[Specific state(s)] を選択してから、リストから 1 つ以上の値を選択します。

8. セレクタが許可するよりも詳細なイベントパターンには、[イベントパターン] ウィンドウの [パターンの編集] オプションを使用して、JSON 形式のイベントパターンを指定することもできます。

   > **Note**
   >
   > 指定されていない場合、イベントパターンは、すべてのパイプライン/ステージ/アクションの状態に対して作成されます。

   より詳細なイベントパターンについては、以下のイベントパターンの例をコピーして [イベントパターン] ウィンドウに貼り付けることができます。

   - Example

     このイベントパターンのサンプルを使用して、すべてのパイプラインで失敗したデプロイとビルドアクションをキャプチャします。

```
{
"source": [
    "aws.codepipeline"
  ],
  "detail-type": [
    "CodePipeline Action Execution State Change"
  ],
  "detail": {
    "state": [
      "FAILED"
    ],
    "type": {
      "category": ["Deploy", "Build"]
    }
  }
}
```

- Example

  このイベントパターンのサンプルを使用して、すべてのパイプラインで、拒否された、または失敗したすべての承認アクションをキャプチャします。

```
{
 "source": [
    "aws.codepipeline"
  ],
  "detail-type": [
    "CodePipeline Action Execution State Change"
  ],
  "detail": {
    "state": [
      "FAILED"
    ],
    "type": {
      "category": ["Approval"]
    }
  }
}
```

- Example

  このイベントパターンのサンプルを使用して、指定したパイプラインからすべてのイベントをキャプチャします。

```
{
"source": [
    "aws.codepipeline"
  ],
  "detail-type": [
    "CodePipeline Pipeline Execution State Change",
    "CodePipeline Action Execution State Change",
    "CodePipeline Stage Execution State Change"
  ],
  "detail": {
    "pipeline": ["myPipeline", "my2ndPipeline"]
  }
}
```

9. [次へ] を選択します。

10. [ターゲットタイプ] で、[AWS サービス] を選択します。
11. [ターゲットの選択] で、[CodePipeline] を選択します。[パイプライン ARN] に、このルールによって開始されるパイプラインの ARN を入力します。

> **Note**
>
> このパイプライン ARN を取得するには、get-pipeline コマンドを実行します。パイプライン ARN が出力に表示されます。以下の形式で作成されます。
> arn:aws:codepipeline:*region*:*account*:*pipeline-name*
> パイプライン ARN の例:
> arn:aws:codepipeline:us-east-2:80398EXAMPLE:MyFirstPipeline

12. EventBridge ルールに関連付けられたターゲットを呼び出すためのアクセス許可を EventBridge に与える IAM サービスロールを作成または指定するには (この場合、ターゲットは CodePipeline):
    - この特定のリソースの新しいロールを作成する を選択して、パイプラインの実行を開始するアクセス許可を EventBridge に付与するサービスロールを作成します。
    - 既存のロールを使用して、パイプラインの実行を開始するアクセス許可を EventBridge に付与するサービスロールを入力します。
13. [次へ] を選択します。
14. [タグ] ページで、[次へ] を選択します
15. [確認と作成] ページで、ルールの設定を確認します。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。

パイプラインの状態が変わる場合、通知を送信する (CLI)

以下の手順では、CLI を使用して、CodePipeline で変更の通知を送信するための CloudWatch Events ルールを作成する方法を示します。

を使用してルール AWS CLI を作成するには、 put-rule コマンドを呼び出し、以下を指定します。

- 作成中のルールを一意に識別する名前。この名前は、 AWS アカウントに関連付けられた CodePipeline で作成するすべてのパイプラインでユニークである必要があります。
- ルールで使用するソースと詳細フィールドのイベントパターン。詳細については、「Amazon EventBridge とイベントパターン」を参照してください。

CodePipeline をイベントソースとする EventBridge ルールを作成するには

1. put-rule コマンドを呼び出して、イベントパターンを指定するルールを作成します。(有効な状態については前のテーブルを参照してください。)

   以下のサンプルコマンドでは、--event-pattern を使用して、"MyPipelineStateChanges" というルールを作成し、"myPipeline" という名前のパイプラインでパイプライン実行に失敗したときに CloudWatch イベントを送信します。

   ```
   aws events put-rule --name "MyPipelineStateChanges" --event-pattern "{\"source\":
   [\"aws.codepipeline\"],\"detail-type\":[\"CodePipeline Pipeline Execution State
    Change\"],\"detail\":{\"pipeline\":[\"myPipeline\"],\"state\":[\"FAILED\"]}}"
   ```

2. put-targets コマンドを呼び出し、次のパラメータを含めます:

   - `--rule` パラメータは、put-rule を使用して作成した `rule_name` で使用されます。
   - `--targets` パラメータは、ターゲットリストのリスト `Id` と Amazon SNS トピックの ARN で使用されます。

   次のサンプルコマンドでは、`MyPipelineStateChanges` と呼ばれるルールに対して指定し、ターゲット `Id` は 1 番で構成されています。これは、ルールのターゲットのリストが何であるかを示し、この場合は ターゲット 1 です。このサンプルコマンドでは、Amazon SNS トピックの例 ARN も指定されます。

   ```
   aws events put-targets --rule MyPipelineStateChanges --targets
    Id=1,Arn=arn:aws:sns:us-west-2:11111EXAMPLE:MyNotificationTopic
   ```

3. EventBridge のアクセス許可を追加し、通知を呼び出す指定されたターゲットサービスを使用します。詳細については、 デベロッパーガイドの [Amazon EventBridge のリソースベースのポリシーを使用する] を参照してください。

イベントのプレースホルダーバケットに関するリファレンス

このセクションは参照のみを目的としています。イベント検出リソースを使用するパイプラインを作成する方法については、「ソースアクションと変更検出方法」を参照してください。

Amazon S3 と CodeCommit が提供するソースアクションでは、イベントベースの変更検出リソースを使用して、ソースバケットやリポジトリでの変更に応じてパイプラインをトリガーします。こ

れらのリソースは、CodeCommit リポジトリのコード変更など、パイプラインソースのイベントに応答するように設定された CloudWatch Events ルールです。Amazon S3 ソースで CloudWatch Events を使用する場合、イベントをログに記録するには、CloudTrail をオンにする必要があります。CloudTrail には、ダイジェストの送信先としての S3 バケットが必要です。CloudWatch Events リソースのログファイルには、カスタムバケットからアクセスできます。ただし、プレースホルダーバケットのデータにはアクセスできません。

- CLI または CloudFormation を使用して CloudWatch Events リソースをセットアップした場合は、パイプラインの設定時に指定したバケットで CloudTrail ファイルを見つけることができます。
- コンソールで S3 ソースを使用してパイプラインをセットアップした場合、コンソールは CloudWatch Events リソースを作成するときにの CloudTrail プレースホルダーバケットを使用します。CloudTrail ダイジェストは、パイプラインが作成される AWS リージョン のプレースホルダーバケットに格納されます。

プレースホルダーバケット以外のバケットを使用する場合は、設定を変更できます。

> **Note**
>
> CloudTrail プレースホルダーバケットに書き込まれたデータは、1 日後に自動的に期限切れになり、保持されません。

CloudTrail ログファイルの検索と管理の詳細については、CloudTrail ログファイルの取得と表示 を参照してください。

トピック

- イベントのプレースホルダーバケット名 (リージョン別)

イベントのプレースホルダーバケット名 (リージョン別)

次の表は、Amazon S3 ソースアクションでパイプラインの変更検出イベントを追跡するログファイルが含まれている S3 プレースホルダーバケットの名前の一覧です。

リージョン名	プレースホルダーバケット名	リージョン識別子
米国東部 (オハイオ)	codepipeline-cloudtrail-placeholder-bucket-us-east-2	us-east-2

リージョン名	プレースホルダーバケット名	リージョン識別子
米国東部 (バージニア北部)	codepipeline-cloudtrail-placeholder-bucket-us-east-1	us-east-1
米国西部 (北カリフォルニア)	codepipeline-cloudtrail-placeholder-bucket-us-west-1	us-west-1
米国西部 (オレゴン)	codepipeline-cloudtrail-placeholder-bucket-us-west-2	us-west-2
カナダ (中部)	codepipeline-cloudtrail-placeholder-bucket-ca-central-1	ca-central-1
欧州 (フランクフルト)	codepipeline-cloudtrail-placeholder-bucket-eu-central-1	eu-central-1
欧州 (アイルランド)	codepipeline-cloudtrail-placeholder-bucket-eu-west-1	eu-west-1
欧州 (ロンドン)	codepipeline-cloudtrail-placeholder-bucket-eu-west-2	eu-west-2
欧州 (パリ)	codepipeline-cloudtrail-placeholder-bucket-eu-west-3	eu-west-3
欧州 (ストックホルム)	codepipeline-cloudtrail-placeholder-bucket-eu-north-1	eu-north-1
アジアパシフィック (香港)	codepipeline-cloudtrail-placeholder-bucket-ap-east-1	ap-east-1
アジアパシフィック (ハイデラバード)	codepipeline-cloudtrail-placeholder-bucket-ap-south-2	ap-south-2
アジアパシフィック (ジャカルタ)	codepipeline-cloudtrail-placeholder-bucket-ap-southeast-3	ap-southeast-3

リージョン名	プレースホルダーバケット名	リージョン識別子
アジアパシフィック (メルボルン)	codepipeline-cloudtrail-placeholder-bucket-ap-southeast-4	ap-southeast-4
アジアパシフィック (ムンバイ)	codepipeline-cloudtrail-placeholder-bucket-ap-south-1	ap-south-1
アジアパシフィック (大阪)	codepipeline-cloudtrail-placeholder-bucket-ap-northeast-3-prod	ap-northeast-3
アジアパシフィック (東京)	codepipeline-cloudtrail-placeholder-bucket-ap-northeast-1	ap-northeast-1
アジアパシフィック (ソウル)	codepipeline-cloudtrail-placeholder-bucket-ap-northeast-2	ap-northeast-2
アジアパシフィック (シンガポール)	codepipeline-cloudtrail-placeholder-bucket-ap-southeast-1	ap-southeast-1
アジアパシフィック (シドニー)	codepipeline-cloudtrail-placeholder-bucket-ap-southeast-2	ap-southeast-2
アジアパシフィック (東京)	codepipeline-cloudtrail-placeholder-bucket-ap-northeast-1	ap-northeast-1
カナダ (中部)	codepipeline-cloudtrail-placeholder-bucket-ca-central-1	ca-central-1
欧州 (フランクフルト)	codepipeline-cloudtrail-placeholder-bucket-eu-central-1	eu-central-1

リージョン名	プレースホルダーバケット名	リージョン識別子
欧州 (アイルランド)	codepipeline-cloudtrail-placeholder-bucket-eu-west-1	eu-west-1
欧州 (ロンドン)	codepipeline-cloudtrail-placeholder-bucket-eu-west-2	eu-west-2
欧州 (ミラノ)	codepipeline-cloudtrail-placeholder-bucket-eu-south-1	eu-south-1
欧州 (パリ)	codepipeline-cloudtrail-placeholder-bucket-eu-west-3	eu-west-3
欧州 (スペイン)	codepipeline-cloudtrail-placeholder-bucket-eu-south-2	eu-south-2
欧州 (ストックホルム)	codepipeline-cloudtrail-placeholder-bucket-eu-north-1	eu-north-1
欧州 (チューリッヒ)*	codepipeline-cloudtrail-placeholder-bucket-eu-central-2	eu-central-2
イスラエル (テルアビブ)	codepipeline-cloudtrail-placeholder-bucket-il-central-1	il-central-1
中東 (バーレーン)*	codepipeline-cloudtrail-placeholder-bucket-me-south-1	me-south-1
中東 (UAE)	codepipeline-cloudtrail-placeholder-bucket-me-central-1	me-central-1
南米 (サンパウロ)	codepipeline-cloudtrail-placeholder-bucket-sa-east-1	sa-east-1

を使用した CodePipeline API コールのログ記録 AWS CloudTrail

AWS CodePipeline は AWS CloudTrail、CodePipeline AWS のサービス のユーザー、ロール、または によって実行されたアクションを記録するサービスである と統合されています。CloudTrail

は、CodePipeline のすべての API コールをイベントとしてキャプチャします。キャプチャされたコールには、CodePipeline コンソールのコールと、CodePipeline API オペレーションへのコードコールが含まれます。証跡を作成する場合は、CodePipeline のイベントなど、Amazon S3 バケットへの CloudTrail イベントの継続的な配信を有効にすることができます。証跡を設定しない場合でも、CloudTrail コンソールの [イベント履歴] で最新のイベントを表示できます。CloudTrail で収集された情報を使用して、CodePipeline に対するリクエスト、リクエスト元の IP アドレス、リクエスト者、リクエスト日時などの詳細を確認できます。

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 CodePipeline 情報

CloudTrail は、アカウントの作成 AWS アカウント 時に で有効になります。CodePipeline でアクティビティが発生すると、そのアクティビティは CloudTrail のイベントとして、他の AWS のサービス ベントと共に イベント履歴 に記録されます。 AWS アカウントで最近のイベントを表示、検索、ダウンロードできます。詳細については、CloudTrail イベント履歴でのイベントの表示を参照してください。

CodePipeline のイベントなど AWS アカウント、 のイベントの継続的な記録については、証跡を作成します。証跡により、CloudTrail はログファイルを Amazon S3 バケットに配信できます。デフォルトでは、コンソールで証跡を作成すると、証跡はすべての AWS リージョンに適用されます。証跡は、 AWS パーティション内のすべてのリージョンからのイベントをログに記録し、指定した Amazon S3 バケットにログファイルを配信します。さらに、CloudTrail ログで収集されたイベントデータをさらに分析して処理 AWS のサービス するように他の を設定できます。詳細については、次を参照してください:

- 証跡の作成のための概要
- CloudTrail がサポートするサービスと統合
- CloudTrail 用 Amazon SNS 通知の構成
- 複数のリージョンから CloudTrail ログファイルを受け取るおよび複数のアカウントから CloudTrail ログファイルを受け取る

すべての CodePipeline アクションは CloudTrail が記録します。これらのアクションは CodePipeline API リファレンス で説明されています。例えば、`CreatePipeline`、`GetPipelineExecution`、`UpdatePipeline` の各アクションを呼び出すと、CloudTrail ログファイルにエントリが生成されます。

各イベントまたはログエントリには、誰がリクエストを生成したかという情報が含まれます。アイデンティティ情報は、以下を判別するのに役立ちます。

- リクエストがルート認証情報または AWS Identity and Access Management (IAM) 認証情報を使用して行われたかどうか。
- リクエストがロールまたはフェデレーションユーザーのテンポラリなセキュリティ認証情報を使用して行われたかどうか。
- リクエストが、別の AWS のサービスによって送信されたかどうか。

詳細については、「CloudTrail userIdentity 要素」を参照してください。

CodePipeline ログファイルエントリについて

「証跡」は、指定した Simple Storage Service (Amazon S3) バケットにイベントをログファイルとして配信するように設定できます。CloudTrail のログファイルは、単一か複数のログエントリを含みます。イベントは、任意の出典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、アクションの日時、リクエストパラメータなどに関する情報が含まれます。CloudTrail ログファイルは、パブリック API 呼び出しの順序付けられたスタックトレースではないため、特定の順序では表示されません。

以下の例では、パイプラインのアップデートイベントの CloudTrail ログエントリを示します。ここで、パイプライン (MyFirstPipeline) が、アカウントID (80398EXAMPLE) を持つユーザー (JaneDoe-CodePipeline) によって編集されています。ユーザーは、パイプラインのソースステージ名を `Source` から `MySourceStage` に変更しました。CloudTrail ログの `requestParameters` と `responseElements` のいずれにも、編集後のパイプラインの構造全体が含まれているため、これらの要素は、以下の例で省略されています。強調 が、変更されたパイプラインの `requestParameters` 部分、以前のバージョン番号のパイプライン、`responseElements` 部分に追加されています。これは、バージョン番号が 1 ずつインクリメントすることを意味します。実際のログエントリでより多くのデータが表示された場合、編集した部分は省略記号 (...) でマークされます。

```
{
 "eventVersion":"1.03",
  "userIdentity": {
   "type":"IAMUser",
   "principalId":"AKIAI44QH8DHBEXAMPLE",
   "arn":"arn:aws:iam::80398EXAMPLE:user/JaneDoe-CodePipeline",
   "accountId":"80398EXAMPLE",
   "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
```

```
    "userName":"JaneDoe-CodePipeline",
    "sessionContext": {
    "attributes":{
     "mfaAuthenticated":"false",
     "creationDate":"2015-06-17T14:44:03Z"
     }
    },
  "invokedBy":"signin.amazonaws.com"},
  "eventTime":"2015-06-17T19:12:20Z",
  "eventSource":"codepipeline.amazonaws.com",
  "eventName":"UpdatePipeline",
  "awsRegion":"us-east-2",
  "sourceIPAddress":"192.0.2.64",
  "userAgent":"signin.amazonaws.com",
  "requestParameters":{
    "pipeline":{
    "version":1,
    "roleArn":"arn:aws:iam::80398EXAMPLE:role/CodePipeline_Service_Role",
    "name":"MyFirstPipeline",
    "stages":[
      {
      "actions":[
       {
           "name":"MySourceStage",
         "actionType":{
         "owner":"AWS",
         "version":"1",
         "category":"Source",
         "provider":"S3"
      },
     "inputArtifacts":[],
     "outputArtifacts":[
      {"name":"MyApp"}
      ],
     "runOrder":1,
     "configuration":{
      "S3Bucket":"amzn-s3-demo-source-bucket",
      "S3ObjectKey":"sampleapp_linux.zip"
       }
       }
      ],
        "name":"Source"
      },
      (...)
```

```
                },
  "responseElements":{
    "pipeline":{
      "version":2,
      (...)
            },
      "requestID":"2c4af5c9-7ce8-EXAMPLE",
      "eventID":""c53dbd42-This-Is-An-Example"",
      "eventType":"AwsApiCall",
      "recipientAccountId":"80398EXAMPLE"
      }
       ]
}
```

CodePipeline CloudWatch メトリクス

CloudWatch のメトリクスを使用して CodePipeline のパイプラインを測定できます。以下のメトリクスを使用して、パイプラインの期間と失敗したパイプライン実行を測定できます。

パイプラインは、次の 2 つのレベルでモニタリングできます。

パイプラインレベル

これらのメトリクスはパイプラインレベルです。CloudWatch で [パイプライン別] を選択します。使用可能なパイプラインは CloudWatch に一覧表示されます。

AWS アカウントレベル

これらのメトリクスは、1 つのアカウントのすべてのパイプラインが対象となります。 AWS アカウントレベルでメトリクスを表示するには、CloudWatch でアカウントメトリクスを選択します。

メトリクスの表示の詳細については、「Amazon CloudWatch ユーザーガイド」の「使用可能なメトリクスの表示」を参照してください。

トピック

- PipelineDuration
- FailedPipelineExecutions

PipelineDuration

PipelineDuration メトリクスは、パイプラインの実行期間を測定します。このメトリクスはパイプラインレベルでのみ使用できます。

単位: 秒

FailedPipelineExecutions

FailedPipelineExecutions メトリクスは失敗したパイプライン実行の数を測定します。このメトリクスは、パイプラインレベルとアカウントレベルで使用できます。

このメトリクスには、失敗したアクションを再試行したパイプライン実行の数を別途測定したものが含まれます。つまり、失敗したアクションがパイプラインで再試行されると、これは別のパイプライン実行メトリクスとしてカウントされます。例えば、S3 ソースアクションが最初に失敗しが 1 つとしてカウントされ、ソースアクションが再試行されて再度失敗する S3 ソースアクションを持つパイプラインの場合です。この例では、メトリクスは次のようになります。

```
FailedPipelineExecutionAttempts: 2
```

単位: カウント

CodePipeline のトラブルシューティング

以下の情報は、 AWS CodePipelineでの一般的な問題のトラブルシューティングに役立ちます。

トピック

- パイプラインエラー: で設定されたパイプラインは、「デプロイに失敗しました」というエラーメッセージ AWS Elastic Beanstalk を返します。提供されたロールに十分なアクセス権限がありません: サービス: AmazonElasticLoadBalancing」
- デプロイエラー: AWS Elastic Beanstalk デプロイアクションで設定されたパイプラインは、DescribeEvents」アクセス許可がない場合に失敗するのではなくハングします
- パイプラインのエラー: ソースアクションは次のようなアクセス許可の不足メッセージを返します。「CodeCommit リポジトリ repository-name にアクセスできませんでした。」 リポジトリにアクセスするための十分な権限がパイプラインの IAM ロールにあることを確認してください。」
- パイプラインのエラー: Jenkins のビルドまたはテストアクションが長期間実行された後、認証情報やアクセス許可の不足のため失敗します。
- パイプラインエラー: 別の AWS リージョンで作成されたバケットを使用してある AWS リージョンで作成されたパイプラインは、JobFailed」というコードのInternalError」を返します。
- デプロイエラー: WAR ファイルを含む ZIP ファイルは正常にデプロイされますが AWS Elastic Beanstalk、アプリケーション URL は 404 が見つかりませんというエラーを報告します
- パイプラインのアーティファクトフォルダ名が切り詰められているように見えます
- Bitbucket、GitHub、GitHub Enterprise Server、または GitLab.com に接続するための CodeBuild GitClone アクセス許可を追加します。
- CodeBuild GitClone のアクセス権限を CodeCommit ソースアクションに追加します。
- パイプラインのエラー: CodeDeployToECS アクションがあるデプロイから、「<source artifact name> からタスク定義アーティファクトファイルを読み取ろうとしたときに例外が発生しました」というエラーメッセージが返されます。
- GitHub (OAuth アプリ経由) ソースアクション: リポジトリリストに異なるリポジトリが表示されます。
- GitHub (GitHub アプリ経由) ソースアクション: リポジトリの接続を完了できません。
- Amazon S3 エラー: CodePipeline サービスロール <ARN> により、S3 バケット <BucketName> に対する S3 アクセスが拒否されました。
- Amazon S3 、Amazon ECR、または CodeCommit ソースを使用した パイプラインは自動的に起動されなくなりました。

- GitHub への接続時の Connections エラー:「問題が発生しました。ブラウザで Cookie が有効になっていることを確認してください」または「組織の所有者は GitHub アプリケーションをインストールする必要があります」
- 実行モードが QUEUED モードまたは PARALLEL モードに変更されたパイプラインは、実行制限に達すると失敗します
- PARALLEL モードのパイプラインは、QUEUED モードまたは SUPERSEDED モードに変更して編集したときに、パイプライン定義が古いままになります。
- PARALLEL モードから変更したパイプラインに、以前の実行モードが表示されます。
- ファイルパスによるトリガーフィルタリングを使用する接続を持つパイプラインは、ブランチの作成時に開始しない可能性があります
- ファイルパスによるトリガーフィルタリングを使用する接続を持つパイプラインは、ファイル制限に達したときに開始しない場合があります
- PARALLEL モードの CodeCommit または S3 ソースリビジョンは、EventBridge イベントと一致しない可能性があります
- EC2 デプロイアクションがエラーメッセージ No such file で失敗する
- EKS デプロイアクションがエラーメッセージ cluster unreachable で失敗する
- 複数のソースアクションが同じリポジトリを参照している場合、パイプラインはすべてのブランチに対してトリガーされません
- 別の問題があるため問い合わせ先を教えてください。

パイプラインエラー: で設定されたパイプラインは、「デプロイに失敗しました」というエラーメッセージ AWS Elastic Beanstalk を返します。提供されたロールに十分なアクセス権限がありません: サービス: AmazonElasticLoadBalancing」

問題: CodePipeline のサービスロールに AWS Elastic Beanstalk、Elastic Load Balancing の一部のオペレーションなど、十分なアクセス許可がありません。この問題に対処するために、CodePipeline のサービスロールが 2015 年 8 月 6 日に更新されました。この日付より前にサービスロールを作成したお客様は、サービスロールのポリシーステートメントを変更して必要なアクセス権限を追加する必要があります。

解決方法: 最も簡単な解決方法は、「CodePipeline サービスロールにアクセス許可を追加する」で記載されているようにサービスロールに対するポリシーステートメントを編集することです。

編集済みのポリシーを適用したら、パイプラインを手動で開始する のステップに従って Elastic Beanstalk を使用するパイプラインを手動で再実行します。

セキュリティのニーズに応じて、他の方法でアクセス権限を変更することもできます。

デプロイエラー: AWS Elastic Beanstalk デプロイアクションで設定されたパイプラインは、DescribeEvents」アクセス許可がない場合に失敗するのではなくハングします

問題: CodePipeline のサービスロールに、`"elasticbeanstalk:DescribeEvents"` を使用するパイプラインの AWS Elastic Beanstalkアクションが含まれている必要があります。このアクセス許可がないと、 AWS Elastic Beanstalk デプロイアクションは失敗したり、エラーを示すことなくハングします。このアクションがサービスロールにない場合、CodePipeline には AWS Elastic Beanstalk ユーザーに代わって でパイプラインデプロイステージを実行するアクセス許可がありません。

修正案: CodePipeline のサービスロールを確認します。`"elasticbeanstalk:DescribeEvents"` アクションが欠落している場合は、「CodePipeline サービスロールにアクセス許可を追加する」の手順に従い、IAM コンソールで [ポリシーの編集] 機能を使用してこのアクションを追加します。

編集済みのポリシーを適用したら、パイプラインを手動で開始する のステップに従って Elastic Beanstalk を使用するパイプラインを手動で再実行します。

パイプラインのエラー: ソースアクションは次のようなアクセス許可の不足メッセージを返します。「CodeCommit リポジトリ **`repository-name`** にアクセスできませんでした。」 リポジトリにアクセスするための十分な権限がパイプラインの IAM ロールにあることを確認してください。」

問題: CodePipeline のサービスロールには CodeCommit に対する十分な権限がなく、CodeCommit リポジトリを使用するためのサポートが、2016 年 4 月 18 日に追加される前に作成された可能性があります。この日付より前にサービスロールを作成したお客様は、サービスロールのポリシーステートメントを変更して必要なアクセス権限を追加する必要があります。

修正案: CodeCommit に必要な権限を CodePipeline サービスロールのポリシーに追加します。詳細については、「CodePipeline サービスロールにアクセス許可を追加する」を参照してください。

パイプラインのエラー: Jenkins のビルドまたはテストアクションが長期間実行された後、認証情報やアクセス許可の不足のため失敗します。

問題: Jenkins サーバーが Amazon EC2 インスタンスにインストールされている場合、インスタンスは CodePipeline に必要な権限を持つインスタンスロールで作成されていない可能性があります。Jakins サーバー、オンプレミスインスタンス、または必要な IAM ロールなしで作成された Amazon EC2 インスタンスで IAM ユーザーを使用している場合、IAM ユーザーには必要な権限がないか、Jenkins サーバーがサーバー上に設定されたプロファイルを介してこれらの認証情報にアクセスできません。

修正案: Amazon EC2 インスタンスロールまたは IAM ユーザーが、`AWSCodePipelineCustomActionAccess` 管理されたポリシーまたは同等の権限で設定されていることを確認します。詳細については、「AWS の 管理ポリシー AWS CodePipeline」を参照してください。

IAM ユーザーを使用している場合は、インスタンスで設定された AWS プロファイルが正しいアクセス許可で設定された IAM ユーザーを使用していることを確認してください。Jenkins と CodePipeline の統合のために設定した IAM ユーザー認証情報を Jenkins UI に直接提供する必要があります。これは推奨されるベストプラクティスではありません。その必要がある場合は、Jenkins サーバーが保護されており、HTTP ではなく HTTPS を使用していることを確認してください。

パイプラインエラー: 別の AWS リージョンで作成されたバケットを使用してある AWS リージョンで作成されたパイプラインは、JobFailed」というコードのInternalError」を返します。

問題: パイプラインとバケットが異なる AWS リージョンに作成されている場合、Amazon S3 バケットに保存されているアーティファクトのダウンロードは失敗します。

解決方法: アーティファクトが保存されている Amazon S3 バケットが、作成したパイプラインと同じ AWS リージョンにあることを確認します。

デプロイエラー: WAR ファイルを含む ZIP ファイルは正常にデプロイされますが AWS Elastic Beanstalk、アプリケーション URL は 404 が見つかりませんというエラーを報告します

問題: WAR ファイルは AWS Elastic Beanstalk 環境に正常にデプロイされますが、アプリケーションの URL は 404 Not Found エラーを返します。

解決方法: AWS Elastic Beanstalk ZIP ファイルは解凍できますが、ZIP ファイルに含まれる WAR ファイルは解凍できません。buildspec.yml ファイルに WAR ファイルを指定する代わりに、デプロイするコンテンツを含むフォルダを指定します。例:

```
version: 0.2

phases:
  post_build:
    commands:
      - mvn package
      - mv target/my-web-app ./
artifacts:
  files:
    - my-web-app/**/*
 discard-paths: yes
```

例については、「AWS Elastic Beanstalk CodeBuild のサンプル」を参照してください。

パイプラインのアーティファクトフォルダ名が切り詰められているように見えます

問題: パイプラインのアーティファクト名を CodePipeline で表示すると、名前が切り詰められているように見えます。これにより、複数の名前が同じように表示されたり、パイプライン名全体の表示が失われたりしたように見えます。

説明: CodePipeline はアーティファクト名を切り詰めることにより、CodePipeline でジョブワーカーの一時的な認証情報を生成するときに、Amazon S3 のフルパスがポリシーサイズの上限を超えないようにします。

アーティファクト名が切り詰められたように見えても、CodePipeline は、名前が切り詰められたアーティファクトに影響されない方法でアーティファクトバケットにマッピングします。パイプライ

ンは正常に動作します。これは、フォルダやアーティファクトでは問題となりません。パイプライン名には 100 文字の制限があります。アーティファクトフォルダ名は、短縮されたように見えても、パイプラインに対して依然として一意です。

Bitbucket、GitHub、GitHub Enterprise Server、または GitLab.com に接続するための CodeBuild GitClone アクセス許可を追加します。

ソースアクションと CodeBuild アクション AWS CodeConnections で を使用する場合、入力アーティファクトをビルドに渡す方法は 2 つあります。

- デフォルト: ソースアクションは、CodeBuild がダウンロードするコードを含む zip ファイルを生成します。
- フルクローン: ソースコードはビルド環境に直接ダウンロードできます。

  フルクローンモードでは、作業中の Git リポジトリとしてソースコードを操作できます。このモードを使用するには、接続を使用するためのアクセス許可を CodeBuild 環境 に付与する必要があります。

CodeBuild サービスロールポリシーにアクセス許可を追加するには、CodeBuild サービスロールにアタッチするカスタマーマネージドポリシーを作成します。次の手順では、 `action`フィールドで`UseConnection`アクセス許可を指定し、 `Resource`フィールドで接続 ARN を指定し、ソースリポジトリ ID を 経由で制限するポリシーを作成します`Condition`。

コンソールを使用して UseConnection のアクセス許可を追加するには

1. パイプラインの接続 ARN とソースリポジトリ ID を検索するには、パイプラインを開き、ソースアクションの (i) アイコンをクリックし、入力タブに切り替えます。

   接続 ARN の例は以下のとおりです。

   ```
   arn:aws:codeconnections:eu-central-1:123456789123:connection/
   sample-1908-4932-9ecc-2ddacee15095
   ```

   ソースリポジトリ ID の例:

   ```
   owner/test-app
   ```

接続 ARN とリポジトリ ID を CodeBuild サービスロールポリシーに追加します。

2. CodeBuild サービスロールを検索するには、パイプラインで使用されるビルドプロジェクトを選択し、ビルドの詳細タブに移動します。
3. [サービスロール] リンクを選択します。これにより IAM コンソールが開き、接続へのアクセスを許可する新しいポリシーを追加できます。
4. IAM コンソールで、アクセス許可の追加を選択し、インラインポリシーの作成を選択します。

   次のサンプルポリシーテンプレートを使用します。次の例に示すように、接続 ARN を `Resource`フィールドに追加し、リポジトリ ID を `Condition`フィールド`codeconnections:FullRepositoryId`の に追加します。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "codeconnections:UseConnection",
            "Resource": "arn:aws:codeconnections:eu-
central-1:123456789123:connection/my-connection-id",
            "Condition": {
                "StringEquals": {
                    "codeconnections:FullRepositoryId": "my-repository-id"
                }
            }
        }
    ]
}
```

   `Condition` フィールドを使用して、ビルド仕様の要件に基づいてポリシーのアクセス許可の範囲をさらに絞り込みます (`CodeConnection`条件ドキュメントを参照) 。

   [JSON] タブで、ポリシーを貼り付けます。

5. [次へ] を選択します。ポリシーの名前 (例: **connection-permissions**) を入力し、[ポリシーの作成] を選択します。

ロールのアクセス許可**connection-permissions**ポリシーにアタッチされたポリシーが表示されます。

CodeBuild GitClone のアクセス権限を CodeCommit ソースアクションに追加します。

パイプラインに CodeCommit ソースアクションがある場合、入力アーティファクトをビルドに渡す方法は 2 つあります。

- デフォルト: ソースアクションは、CodeBuild がダウンロードするコードを含む zip ファイルを生成します。
- フルクローン: ソースコードは、直接ビルド環境にダウンロードできます。

  フルクローン オプションでは、作業中の Git リポジトリとしてソースコードを操作することができます。このモードを使用するには、CodeBuild 環境がリポジトリからプルするアクセス権限を追加する必要があります。

CodeBuild サービスロールポリシーにアクセス許可を追加するには、CodeBuild サービスロールにアタッチするカスタマーマネージドポリシーを作成します。次の手順では、`codecommit:GitPull` アクセス権限を `action` フィールドで指定するポリシーを作成します。

コンソールを使用して GitPull のアクセス権限を追加するには

1. CodeBuild サービスロールを確認するには、パイプラインで使用されているビルドプロジェクトを開き、[Build details] (ビルドの詳細) タブに移動します。
2. [サービスロール] リンクを選択します。これにより IAM コンソールが開き、リポジトリへのアクセスを許可する新しいポリシーを追加できます。
3. IAM コンソールで [ポリシーのアタッチ] を選択し、[ポリシーの作成] を選択します。
4. [JSON] タブに、以下のサンプルポリシーを貼り付けます。

```
{
    "Action": [
        "codecommit:GitPull"
    ],
    "Resource": "*",
    "Effect": "Allow"
```

```
},
```

5. [ポリシーの確認] を選択します。ポリシーの名前 (例: **codecommit-gitpull**) を入力し、[ポリシーの作成] を選択します。
6. アクセス許可をアタッチしたページに戻り、ポリシーリストを更新して、作成したポリシーを選択します。[ポリシーのアタッチ] を選択します。

パイプラインのエラー: CodeDeployToECS アクションがあるデプロイから、「<source artifact name> からタスク定義アーティファクトファイルを読み取ろうとしたときに例外が発生しました」というエラーメッセージが返されます。

問題:

タスク定義ファイルは、CodePipeline から Amazon ECS へのデプロイアクション (CodeDeployToECS アクション) に必要なアーティファクトです。CodeDeployToECS デプロイアクションのアーティファクト ZIP の最大サイズは 3 MB です。ファイルが見つからないかアーティファクトのサイズが 3 MB を超える場合は、次のエラーメッセージが返されます。

<source artifact name> からタスク定義アーティファクトファイルを読み取ろうとしたときに例外が発生しました

解決方法: タスク定義ファイルがアーティファクトとして含まれていることを確認してください。ファイルがすでに存在する場合は、圧縮サイズが 3 MB 未満であることを確認します。

GitHub (OAuth アプリ経由) ソースアクション: リポジトリリストに異なるリポジトリが表示されます。

問題:

CodePipeline コンソールで GitHub (OAuth アプリ経由) アクションの認可が成功したら、GitHub リポジトリのリストから選択できます。リストに表示されるはずのリポジトリが含まれていない場合は、認可に使用したアカウントをトラブルシューティングできます。

解決方法: CodePipeline コンソールで提供されるリポジトリのリストは、承認されたアカウントが属する GitHub Organization に基づいています。GitHub で認可するために使用しているアカウントが、

リポジトリが作成されている GitHub Organization に関連付けられているアカウントであることを確認します。

GitHub (GitHub アプリ経由) ソースアクション: リポジトリの接続を完了できません。

問題:

GitHub リポジトリへの接続は AWS Connector for GitHub を使用するため、接続を作成するには、リポジトリへの組織所有者のアクセス許可または管理者アクセス許可が必要です。

解決方法: GitHub リポジトリのアクセス許可レベルの詳細については、https://docs.github.com/en/free-pro-team@latest/github/setting-up-and-managing-organizations-and-teams/permission-levels-for-an-organization を参照してください。

Amazon S3 エラー: CodePipeline サービスロール <ARN> により、S3 バケット <BucketName> に対する S3 アクセスが拒否されました。

問題:

進行中、CodePipeline の CodeCommit アクションは、パイプラインアーティファクトバケットが存在することをチェックします。アクションにチェックするアクセス許可がない場合は、Amazon S3 で `AccessDenied` のエラーが発生し、CodePipeline に次のエラーメッセージが表示されます。

CodePipeline サービスロール「arn:aws:iam:: *`AccountID`* :role/service-role/*`RoleID`* により、S3 バケット「*`BucketName`*」の S3 アクセスが拒否されています。

アクションの CloudTrail ログにも `AccessDenied` のエラーが記録されます。

解決方法: 次の作業を行います。

- CodePipeline サービスロールにアタッチされたポリシーについて、`s3:ListBucket` をポリシー内のアクションのリストに追加します。サービスロールポリシーを表示する方法については、「」を参照してくださいパイプラインの ARN とサービスロール ARN (コンソール) を表示します。。サービスロールのポリシーステートメントを編集するには「CodePipeline サービスロールにアクセス許可を追加する」を参照してください。

- パイプラインの Amazon S3 アーティファクトバケットにアタッチされたリソースベースのポリシー、通称 アーティファクトバケットポリシー の場合、CodePipeline サービスロールが `s3:ListBucket` アクセス許可を使用できるようステートメントを追加します。

  アーティファクトバケットにポリシーを追加するには

  1. パイプラインの ARN とサービスロール ARN (コンソール) を表示します。 の手順を行い、パイプラインの [設定] ページでアーティファクトバケットを選択し、Amazon S3 コンソールに表示させます。
  2. [Permissions (アクセス許可)] を選択します。
  3. [バケットポリシー] で [編集] を選択します。
  4. [ポリシー] テキストフィールドで、新しいバケットポリシーを入力するか、次の例に示すように既存のポリシーを編集します。バケットポリシーは JSON ファイルであるため、有効な JSON を入力する必要があります。

     次の例は、サービスロールのロール ID の例が *AROAEXAMPLEID* であるアーティファクトバケットのバケットポリシーステートメントを示しています。

     ```
     {
         "Effect": "Allow",
         "Principal": "*",
         "Action": "s3:ListBucket",
         "Resource": "arn:aws:s3:::BucketName",
         "Condition": {
             "StringLike": {
                 "aws:userid": "AROAEXAMPLEID:*"
             }
         }
     }
     ```

     次の例は、アクセス許可が追加された後の同じバケットポリシーステートメントを示しています。

     JSON

     ```
     {
         "Version":"2012-10-17",
         "Id": "SSEAndSSLPolicy",
         "Statement": [
             {
     ```

```
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:ListBucket",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket",
            "Condition": {
                "StringLike": {
                    "aws:userid": "AROAEXAMPLEID:*"
                }
            }
        },
        {
            "Sid": "DenyUnEncryptedObjectUploads",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
            "Condition": {
                "StringNotEquals": {
                    "s3:x-amz-server-side-encryption": "aws:kms"
                }
            }
        },
        {
            "Sid": "DenyInsecureConnections",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
            "Condition": {
                "Bool": {
                    "aws:SecureTransport": false
                }
            }
        }
    ]
}
```

詳細については、https://aws.amazon.com/blogs/security/writing-iam-policies-how-to-grant-access-to-an-amazon-s3-bucket/のステップ を参照してください。

5. [保存] を選択します。

編集済みのポリシーを適用したら、パイプラインを手動で開始する のステップに従ってパイプラインを手動で再実行します。

Amazon S3 、Amazon ECR、または CodeCommit ソースを使用した パイプラインは自動的に起動されなくなりました。

問題:

変更検出にイベントルール (EventBridge または CloudWatch Events) を使用するアクションの設定を変更した後、ソーストリガーの識別子が類似していて、同じ初期文字を持っている場合、コンソールで変更を検出できないことがあります。新しいイベントルールはコンソールによって作成されないため、パイプラインは自動的に起動されなくなります。

CodeCommit のパラメータ名の末尾にマイナーな変更を加える例として、CodeCommit ブランチ名を `MyTestBranch-1` から`MyTestBranch-2` に変更することが挙げられます。ブランチ名の末尾に変更があるため、ソースアクションのイベントルールが新しいソース設定のルールを更新、または作成しない場合があります。

これは、次のように変更検出に CWE イベントを使用するソースアクションに適用されます。

ソースアクション	パラメータおよびトリガー識別子 (コンソール)
Amazon ECR	リポジトリ名 イメージタグ
Amazon S3	バケット S3 オブジェクトキー
CodeCommit	リポジトリ名 ブランチ名

解決方法:

以下の いずれかを 実行します。

- CodeCommit、S3 および ECR 構成設定を変更して、パラメータ値の開始部分が変更されるようにします。

  例: ブランチ名を `release-branch` から `2nd-release-branch` に変更します。`release-branch-2` など、名前の末尾の変更は避けてください。
- 各パイプラインの CodeCommit、S3 および ECR 構成設定を変更します。

  例: ブランチ名を `myRepo/myBranch` から `myDeployRepo/myDeployBranch` に変更します。`myRepo/myBranch2` など、名前の末尾の変更は避けてください。
- コンソールの代わりに、 CLI または を使用して CloudFormation 、変更検出イベントルールを作成および更新します。S3 ソースアクションのイベントルールを作成する手順については、「EventBridge と を使用する Amazon S3 ソースアクションへの接続 AWS CloudTrail」を参照してください。Amazon ECR アクションのイベントルールを作成する手順については、「Amazon ECR ソースアクションと EventBridge リソース」を参照してください。CodeCommit アクションのイベントルールを作成する手順については、「CodeCommit ソースアクションと EventBridge」を参照してください。

コンソールでアクション設定を 編集した後、コンソールによって作成された更新された変更検出リソースを 容認します。

GitHub への接続時の Connections エラー:「問題が発生しました。ブラウザで Cookie が有効になっていることを確認してください」または「組織の所有者は GitHub アプリケーションをインストールする必要があります」

問題:

CodePipeline で GitHub ソースアクション用に接続を作成するには、GitHub 組織の所有者であることが必要です。組織のリポジトリではない場合、ユーザーがリポジトリの所有者である必要があります。接続の作成者が組織の所有者以外である場合、組織の所有者へのリクエストが作成され、次のエラーのいずれかが表示されます。

問題が発生しました。ブラウザで Cookie が有効になっていることを確認してください

OR

組織の所有者は GitHub アプリケーションをインストールする必要があります

解決策: GitHub 組織のリポジトリである場合、組織の所有者が GitHub リポジトリへの接続を作成する必要があります。組織のリポジトリでない場合、ユーザーがリポジトリの所有者である必要があります。

実行モードが QUEUED モードまたは PARALLEL モードに変更されたパイプラインは、実行制限に達すると失敗します

問題: QUEUED モードのパイプラインの場合、同時実行の最大数は 50 です。この制限に達すると、パイプラインはステータスメッセージなしで失敗します。

可能な修正: 実行モードのパイプライン定義を編集するときは、他の編集アクションとは別に編集を行います。

QUEUED 実行モードまたは PARALLEL 実行モードの詳細については、「CodePipeline の概念 」を参照してください。

PARALLEL モードのパイプラインは、QUEUED モードまたは SUPERSEDED モードに変更して編集したときに、パイプライン定義が古いままになります。

問題: 並列モードのパイプラインで、パイプライン実行モードを QUEUED または SUPERSEDED に編集したときに、PARALLEL モードのパイプライン定義は更新されません。PARALLEL モードを更新するときに更新されたパイプライン定義は、SUPERSEDEDED モードまたは QUEUED モードでは使用されません。

考えられる修正: 並列モードのパイプラインの場合、パイプライン実行モードを QUEUED または SUPERSEDED に編集するときに、パイプライン定義を同時に更新しないでください。

QUEUED 実行モードまたは PARALLEL 実行モードの詳細については、「CodePipeline の概念 」を参照してください。

PARALLEL モードから変更したパイプラインに、以前の実行モードが表示されます。

問題: PARALLEL モードのパイプラインで、パイプライン実行モードを QUEUED または SUPERSEDED に編集したときに、パイプライン状態には更新された状態が PARALLEL として表

示されません。パイプラインを PARALLEL から QUEUED または SUPERSEDED に変更した場合、SUPERSEDED モードまたは QUEUED モードのパイプラインの状態は、そのいずれかのモードで最後に実行されたときの状態になります。パイプラインがそのモードで実行されたことがない場合、状態は空になります。

考えられる修正： 並列モードのパイプラインの場合、パイプライン実行モードを QUEUED または SUPERSEDED に編集したときに、実行モードの表示に PARALLEL 状態が表示されないことに注意してください。

QUEUED 実行モードまたは PARALLEL 実行モードの詳細については、「CodePipeline の概念 」を参照してください。

ファイルパスによるトリガーフィルタリングを使用する接続を持つパイプラインは、ブランチの作成時に開始しない可能性があります

説明: パイプラインに接続を使用するソースアクション (BitBucket ソースアクションなど) がある場合、ファイルパスでのフィルタリングを許可する Git 設定でトリガーをセットアップすることで、パイプラインを開始できます。場合によっては、ファイルパスでフィルタリングされたトリガーを持つパイプラインの場合、ファイルパスフィルターを持つブランチが最初に作成されたときにパイプラインが開始されないことがあります。これは、変更されたファイルを AWS CodeConnections 接続で解決できないためです。トリガーの Git 設定がファイルパスでフィルタリングするように設定されている場合、フィルターを持つブランチがソースリポジトリに作成されたばかりのときにパイプラインは開始されません。ファイルパスでのフィルタリングの詳細については、「」を参照してくださいコードプッシュまたはプルリクエストイベントタイプを使用してトリガーを追加する。

結果: 例えば、CodePipeline のパイプラインでファイルパスフィルターがブランチ「B」にある場合、パイプラインはブランチ「B」の作成時にトリガーされません。パイプラインは、ファイルパスフィルターがない場合でも開始します。

ファイルパスによるトリガーフィルタリングを使用する接続を持つパイプラインは、ファイル制限に達したときに開始しない場合があります

説明: パイプラインに接続を使用するソースアクション (BitBucket ソースアクションなど) がある場合、ファイルパスでのフィルタリングを許可する Git 設定でトリガーをセットアップすることで、パイプラインを開始できます。CodePipeline は最初の 100 個のファイルを取得するため、トリガーの

Git 設定がファイルパスでフィルタリングするように設定されている場合、100 個を超えるファイルがある場合、パイプラインが開始されないことがあります。ファイルパスでのフィルタリングの詳細については、「」を参照してください[コードプッシュまたはプルリクエストイベントタイプを使用してトリガーを追加する](#)。

結果: 例えば、差分に 150 個のファイルが含まれている場合、CodePipeline は最初の 100 個のファイル (特定の順序なし) を取得し、指定したファイルパスフィルターと照合します。ファイルパスフィルターと一致するファイルが CodePipeline が取得した 100 個のファイルに含まれていない場合、パイプラインは呼び出されません。

PARALLEL モードの CodeCommit または S3 ソースリビジョンは、EventBridge イベントと一致しない可能性があります

説明: PARALLEL モードのパイプライン実行の場合、実行は CodeCommit リポジトリコミットなどの最新の変更で開始する場合があり、これは EventBridge イベントの変更とは異なることがあります。パイプラインを開始するコミット間やイメージタグ間に一瞬の差がある場合、CodePipeline がイベントを受信して実行を開始するときに、別のコミットやイメージタグがプッシュされると、CodePipeline (CodeCommit アクションなど) はその時点で HEAD コミットをクローンします。

結果: PARALLEL モードのパイプラインに CodeCommit または S3 ソースがある場合、パイプライン実行をトリガーした変更に関係なく、ソースアクションは常に開始時に HEAD をクローンします。例えば、PARALLEL モードのパイプラインの場合、コミットをプッシュすると、実行 1 のパイプラインが開始し、2 番目のパイプライン実行で 2 番目のコミットが使用されます。

EC2 デプロイアクションがエラーメッセージ **`No such file`** で失敗する

説明: EC2 デプロイアクションがインスタンスのターゲットディレクトリにあるアーティファクトを解凍すると、アクションはスクリプトを実行します。スクリプトがターゲットディレクトリにあるが、アクションがスクリプトを実行できない場合、アクションはそのインスタンスで失敗し、残りのインスタンスはデプロイに失敗します。

次のエラーメッセージのようなエラーは、ターゲットディレクトリが `/home/ec2-user/deploy/` で、ソースリポジトリパスが `myRepo/postScript.sh` であるデプロイのログに表示されます。

- ```
  Instance i-0145a2d3f3EXAMPLE is FAILED on event AFTER_DEPLOY, message:
  ```
  ```

```
----------ERROR-------
chmod: cannot access '/home/ec2-user/deploy/myRepo/postScript.sh': No such file or
 directory
/var/lib/<path>/_script.sh: line 2: /home/ec2-user/deploy/myRepo/postScript.sh: No
 such file or directory
failed to run commands: exit status 127
```

- ```
  Executing commands on instances i-0145a2d3f3EXAMPLE, SSM command id <ID>, commands:
   chmod u+x /home/ec2-user/deploy/script.sh
  ----------ERROR-------: No such file or directory
  ```

結果: パイプラインでデプロイアクションが失敗します。

解決方法: トラブルシューティングを行うには、次の手順を実行します。

1. ログを表示して、スクリプトが失敗する原因となったインスタンスを確認します。
2. ディレクトリ (cd) をインスタンスのターゲットディレクトリに変更します。インスタンスでスクリプトをテスト実行します。
3. ソースリポジトリで、スクリプトファイルを編集して、問題の原因となっている可能性のあるコメントやコマンドを削除します。

EKS デプロイアクションがエラーメッセージ **cluster unreachable** で失敗する

説明: EKS デプロイアクションが実行されると、アクションは `cluster unreachable` エラーメッセージで失敗します。このメッセージは、アクセス許可がないためにクラスターにアクセスに問題があることを示しています。ファイルのタイプ (Helm チャートまたは Kubernetes マニフェストファイル) に基づいて、エラーメッセージが次のように表示されます。

- Helm チャートを使用している EKS デプロイアクションの場合、次のエラーメッセージのようなエラーが表示されます。

```
error message:

helm upgrade --install my-release test-chart --wait
Error: Kubernetes cluster unreachable: the server has asked for the client to provide
 credentials
```
```

- Kubernetes マニフェストファイルを使用している EKS デプロイアクションの場合、次のエラーメッセージのようなエラーが表示されます。

```
kubectl apply -f deployment.yaml
Error: error validating "deployment.yaml": error validating data: failed to download
 openapi: the server has asked for the client to provide credentials
```

結果: パイプラインでデプロイアクションが失敗します。

解決方法: 既存のロールを使用する場合は、CodePipeline のサービスロールを EKS デプロイアクションを使用するために必要なアクセス許可で更新する必要があります。さらに、CodePipeline のサービスロールにクラスターへのアクセスを許可するには、クラスターにアクセスエントリを追加し、アクセスエントリのサービスロールを指定する必要があります。

1. CodePipeline のサービスロールに EKS デプロイアクションに必要なアクセス許可があることを確認します。アクセス許可のリファレンスについては、「サービスロールのポリシーのアクセス許可」を参照してください。
2. クラスターにアクセスエントリを追加し、アクセス用の CodePipeline のサービスロールを指定します。例については、ステップ 4: CodePipeline サービスロールのアクセスエントリを作成するを参照してください。

複数のソースアクションが同じリポジトリを参照している場合、パイプラインはすべてのブランチに対してトリガーされません

問題: パイプラインには、接続を使用する複数のソースアクション (Bitbucket、GitHub、GitLab 接続など) があり、各ソースアクションは同じリポジトリの異なるブランチを指します。変更をプッシュすると、いずれかのブランチのみがパイプラインをトリガーします。接続のウェブフックサブスクリプションは、ブランチごとではなく、パイプラインとリポジトリの組み合わせに登録されます。そのため、1 つのパイプライン内で同じリポジトリの異なるブランチをターゲットとする複数のソースアクションはサポートされていません。

解決方法: 個別にトリガーするブランチごとに個別のパイプラインを使用します。

別の問題があるため問い合わせ先を教えてください。

これらの他のリソースを試してください。

- AWS Support にお問い合わせください。
- [CodePipeline フォーラム] でお問い合わせください。
- クォータの引き上げをリクエストする。詳細については、「AWS CodePipeline のクォータ」を参照してください。

> **Note**
>
> クォータの引き上げリクエストの処理には、最大 2 週間かかる場合があります。

のセキュリティ AWS CodePipeline

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と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 は、 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責任を担います AWS クラウド。 は、お客様が安全に使用できるサービス AWS も提供します。サードパーティーの監査者は、AWS コンプライアンスプログラムコンプライアンスプログラムの一環として、当社のセキュリティの有効性を定期的にテストおよび検証。が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 CodePipeline、「コンプライアンスプログラムAWS による対象範囲内のサービスコンプライアンスプログラム」を参照してください。
- クラウドのセキュリティ – 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、CodePipeline 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 CodePipeline を設定する方法について説明します。また、CodePipeline 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

- でのデータ保護 AWS CodePipeline
- の ID とアクセスの管理 AWS CodePipeline
- CodePipeline でのロギングとモニタリング
- のコンプライアンス検証 AWS CodePipeline
- の耐障害性 AWS CodePipeline
- のインフラストラクチャセキュリティ AWS CodePipeline
- セキュリティのベストプラクティス

でのデータ保護 AWS CodePipeline

AWS の責任共有モデル は、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 AWS CLIまたは他の AWS のサービス を操作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。

以下のセキュリティのベストプラクティスも CodePipeline でのデータ保護に対処します。

- CodePipeline 用に Amazon S3 に保存したアーティファクトのサーバー側の暗号化を設定する
- AWS Secrets Manager を使用してデータベースパスワードまたはサードパーティー API キーを追跡する

ネットワーク間のトラフィックのプライバシー

Amazon VPC AWS のサービス は、定義した仮想ネットワーク (仮想プライベートクラウド) で AWS リソースを起動するために使用できる です。CodePipeline は、プライベート IP アドレスを持つ Elastic Network Interface AWS のサービス を使用する間のプライベート通信を容易にする AWS テクノロジーである AWS PrivateLink を搭載した Amazon VPC エンドポイントをサポートしています。つまり、VPC のプライベートエンドポイントを介して CodePipeline に直接接続し、VPC と AWS ネットワーク内にすべてのトラフィックを保持できます。以前は、VPC 内で実行されているアプリケーションから、CodePipeline にインターネット接続する必要がありました。VPC では、次のようなネットワーク設定を管理することができます。

- IP アドレス範囲
- Subnets
- ルートテーブル、
- ネットワークゲートウェイ。

VPC を CodePipeline に接続するには、CodePipeline のインターフェイス VPC エンドポイントを定義します。このタイプのエンドポイントにより、VPC を AWS のサービスに接続できるようになります。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 CodePipeline への接続を提供します。VPC を設定する方法の詳細については、VPC ユーザーガイド参照してください。

保管中の暗号化

CodePipeline のデータは、保管時に を使用して暗号化されます AWS KMS keys。コードアーティファクトは、カスタマー所有の S3 バケットに保存され、 AWS マネージドキー またはカスタマーマネージドキーで暗号化されます。詳細については、「CodePipeline 用に Amazon S3 に保存したアーティファクトのサーバー側の暗号化を設定する」を参照してください。

転送中の暗号化

すべてのサービス間の通信は、SSL/TLS を使用して転送中に暗号化されます。

暗号化キーの管理

コードアーティファクトを暗号化するためのデフォルトのオプションを選択する場合、CodePipeline は AWS マネージドキーを使用します。これを変更または削除することはできません AWS マネージドキー。でカスタマーマネージドキーを使用して S3 バケット内のアーティファクトを AWS KMS 暗号化または復号する場合は、必要に応じてこのカスタマーマネージドキーを変更または口ーテーションできます。

> **Important**
>
> CodePipeline は、対称 KMS キーのみをサポートしています。非対称キーを使用して S3 bucket のデータを暗号化しないでください。

CodePipeline 用に Amazon S3 に保存したアーティファクトのサーバー側の暗号化を設定する

Amazon S3 アーティファクトのサーバー側の暗号化を設定するには、次の 2 つの方法があります。

- CodePipeline は、パイプラインの作成ウィザードを使用してパイプラインを作成する AWS マネージドキー ときに、S3 アーティファクトバケットとデフォルトを作成します。 AWS マネージドキー はオブジェクトデータと共に暗号化され、 によって管理されます AWS。
- 独自の カスタマーマネージドキー を作成して管理できます。

> **Important**
>
> CodePipeline は、対称 KMS キーのみを対応しています。非対称 KMS キーを使用して S3 bucket のデータを暗号化しないでください。

デフォルトの S3 キーを使用している場合、この AWS マネージドキーを変更または削除することはできません。でカスタマーマネージドキーを使用して S3 バケット内のアーティファクトを AWS KMS 暗号化または復号する場合は、必要に応じてこのカスタマーマネージドキーを変更または口ーテーションできます。

Amazon S3 は、バケットに格納されているすべてのオブジェクトに対してサーバー側の暗号化が必要な場合に使用できるバケットポリシーをサポートしています。例えば、SSE-KMS を使用したサーバー側の暗号化を要求する `s3:PutObject` ヘッダーがリクエストに含まれていない場合、次のバケットポリシーはすべてのユーザーに対し、オブジェクト (`x-amz-server-side-encryption`) をアップロードするアクセス許可を拒否します。

JSON

```
{
    "Version":"2012-10-17",
    "Id": "SSEAndSSLPolicy",
    "Statement": [
        {
            "Sid": "DenyUnEncryptedObjectUploads",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::codepipeline-us-west-2-89050EXAMPLE/*",
            "Condition": {
                "StringNotEquals": {
                    "s3:x-amz-server-side-encryption": "aws:kms"
                }
            }
        },
        {
            "Sid": "DenyInsecureConnections",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::codepipeline-us-west-2-89050EXAMPLE/*",
            "Condition": {
                "Bool": {
                    "aws:SecureTransport": "false"
                }
            }
        }
    ]
}
```

サーバー側の暗号化の詳細については AWS KMS、「サーバー側の暗号化を使用したデータの保護」および「 (SSE-KMS) に保存 AWS Key Management Service されている KMS キーを使用したサーバー側の暗号化を使用したデータの保護」を参照してください。

詳細については AWS KMS、「 AWS Key Management Service デベロッパーガイド」を参照してください。

トピック

- を表示する AWS マネージドキー
- CloudFormation または を使用して S3 バケットのサーバー側の暗号化を設定する AWS CLI

を表示する AWS マネージドキー

[パイプラインの作成] ウィザードを使用して最初のパイプラインを作成すると、パイプラインを作成したのと同じリージョンに S3 バケットが作成されます。バケットは、パイプラインアーティファクトを格納するために使用されます。パイプラインを実行すると、アーティファクトが、S3 バケットに入れられるか、そこから取得されます。デフォルトでは、CodePipeline は AWS マネージドキー for Amazon S3 (aws/s3キー) AWS KMS を使用して でサーバー側の暗号化を使用します。これは AWS マネージドキー 作成され、 AWS アカウントに保存されます。アーティファクトが S3 バケットから取得されると、CodePipeline は同じ SSE-KMS プロセスを使用してアーティファクトを復号化します。

に関する情報を表示するには AWS マネージドキー

1. にサインイン AWS マネジメントコンソール し、 AWS KMS コンソールを開きます。
2. ウェルカムページが表示される場合は、[今すぐ始める] を選択します。
3. サービスのナビゲーションペインで、[AWS managed keys] を選択します。
4. パイプラインのリージョンを選択します。例えば、パイプラインが us-east-2 に作成されている場合は、フィルタが 米国西部 (オハイオ州) に設定されていることを確認します。

   CodePipeline で使用できるリージョンとエンドポイントの詳細については、「AWS CodePipeline エンドポイントとクォータ」を参照してください。
5. リスト内のパイプラインに使用されるエイリアスがあるキー (デフォルトは aws/s3) を選択します。キーの基本情報が表示されます。

CloudFormation または を使用して S3 バケットのサーバー側の暗号化を設定する AWS CLI

CloudFormation または を使用してパイプライン AWS CLI を作成する場合は、サーバー側の暗号化を手動で設定する必要があります。上記のサンプルバケットポリシーを使用して、独自のカスタマーマネージドキーを作成します。デフォルトの AWS マネージドキーキーの代わりに独自のキーを使用することもできます。独自のキーを選択する理由には、次のようなものがあります。

- 組織のビジネス要件またはセキュリティ要件を満たすために、スケジュールに基づいてキーのローテーションをする必要があります。
- 別の AWS アカウントに関連付けられたリソースを使用するパイプラインを作成する必要があります。これには、 カスタマーマネージドキー を使用する必要があります。詳細については、「別の AWS アカウントのリソースを使用する CodePipeline でパイプラインを作成する」を参照してください。

暗号化のベストプラクティスでは、暗号化キーの広範な再利用を推奨していません。ベストプラクティスとして、キーを定期的にローテーションします。 AWS KMS キーの新しい暗号化マテリアルを作成するには、カスタマーマネージドキーを作成し、新しいカスタマーマネージドキーを使用するようにアプリケーションまたはエイリアスを変更します。または、既存の カスタマー管理キー の自動キーローテーションを有効にすることができます。

カスタマーマネージドキーをローテーションするには、「キーローテーション」を参照してください。

> **Important**
>
> CodePipeline は、対称 KMS キーのみを対応しています。非対称キーを使用して S3 bucket のデータを暗号化しないでください。

AWS Secrets Manager を使用してデータベースパスワードまたはサードパーティー API キーを追跡する

を使用して、データベース認証情報、API キー、およびその他のシークレット AWS Secrets Manager をライフサイクル全体でローテーション、管理、取得することをお勧めします。Secrets Manager を使用すると、コード内のハードコードされた認証情報 (パスワードを含む) を、Secrets Manager への API コールで置き換えて、プログラムでシークレットを取得することができます。

詳細については、AWS 「Secrets Manager とは」を参照してください。 AWS 「Secrets Manager ユーザーガイド」の「」を参照してください。

CloudFormation テンプレートでシークレットであるパラメータ (OAuth 認証情報など) を渡すパイプラインの場合、Secrets Manager に保存したシークレットにアクセスする動的参照をテンプレートに含める必要があります。参照 ID パターンと例については、「Secrets Manager のシークレットのAWS CloudFormation ユーザーガイド」を参照してください。パイプラインで GitHub Webhook のテンプレートスニペットで動的参照を使用する例については、「Webhook リソースの設定」を参照してください。

関連情報

シークレットを管理する際に役立つ関連リソースは以下の通りです。

- Secrets Manager は、Amazon RDS シークレットのローテーションなど、データベースの認証情報を自動的にローテーションできます。詳細については、AWS 「 Secrets Manager ユーザーガイド」の「Secrets Manager シークレットのローテーション」を参照してください。 AWS
- Secrets Manager の動的参照を CloudFormation テンプレートに追加する方法については、https://aws.amazon.com/blogs/security/how-to-create-and-retrieve-secrets-managed-in-aws-secrets-manager-using-aws-cloudformation-template/ を参照してください。

の ID とアクセスの管理 AWS CodePipeline

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 CodePipeline リソースの使用を 承認する (アクセス許可を付与する) かを制御します。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
- が IAM と AWS CodePipeline 連携する方法
- AWS CodePipeline アイデンティティベースのポリシーの例
- AWS CodePipeline リソースベースのポリシーの例

- AWS CodePipeline ID とアクセスのトラブルシューティング
- アクセス許可に関するリファレンス
- CodePipeline サービスロールを管理する

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 CodePipeline 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 (「が IAM と AWS CodePipeline 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 CodePipeline 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 アカウントのルートユーザー

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 つのサインインアイデンティティから始めます。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。

IAM ユーザーとグループ

IAM ユーザーは、特定の個人やアプリケーションに対する特定のアクセス許可を持つアイデンティティです。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。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。

リソースベースのポリシー

リソースベースのポリシーは、リソースにアタッチする JSON ポリシードキュメントです。例としては、IAM ロール信頼ポリシーや Amazon S3 バケットポリシーなどがあります。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーでは、IAM の AWS マネージドポリシーを使用できません。

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。

が IAM と AWS CodePipeline 連携する方法

IAM を使用して CodePipeline へのアクセスを管理する前に、CodePipeline で使用できる IAM 機能について理解しておく必要があります。CodePipeline およびその他の AWS のサービス が IAM と連携する方法の概要を把握するには、AWS のサービス「IAM ユーザーガイド」の「IAM と連携する」を参照してください。

トピック

- CodePipeline アイデンティティベースのポリシー
- CodePipeline リソースベースのポリシー
- CodePipeline タグに基づく許可
- CodePipeline IAM ロール

CodePipeline アイデンティティベースのポリシー

IAM アイデンティティベースの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、またアクションを許可または拒否する条件を指定できます。CodePipeline は、特定のアクション、リソース、および条件キーをサポートしています。JSON ポリシーで使用するすべての要素については、「IAM ユーザーガイド」の「IAM JSON ポリシー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。

CodePipeline のポリシーアクションは、アクションの前にプレフィックス `codepipeline:` を使用します：。

例えば、アカウント内の既存のパイプラインを表示するアクセス許可を他のユーザーに付与するには、ユーザーのポリシーに codepipeline:GetPipeline アクションを含めます。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。CodePipeline 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": [
      "codepipeline:action1",
      "codepipeline: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、Get 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": "codepipeline:Get*"
```

CodePipeline 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 CodePipelineによって定義されたアクション」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": "*"
```

リソースとオペレーション

では、プライマリリソースはパイプラインです。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。 は、ステージ、アクション、カスタムアクションなど、プライマリリソースで使用できる他のリソースを選択します。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 Amazon リソースネーム (ARN) が関連付けられています。ARNs、の「Amazon リソースネーム (ARN) と AWS のサービス 名前空間」を参照してくださいAmazon Web Services 全般のリファレンス。パイプラインに関連付けられたパイプラインの ARN を取得するには、「設定コンソールに表示されるパイプラインの ARN 」を参照してください。詳細については、「パイプラインの ARN とサービスロール ARN (コンソール) を表示します。」を参照してください。

リソースタイプ	ARN 形式
パイプライン	arn:aws:コードパイプライン:[##]:[#####]:[#######]
ステージ	arn:aws:コードパイプライン:[##]:[#####]:[#######/#####]
アクション	arn:aws:codepipeline:*region*:*account*:*pipeline-name* /*stage-name* /*action-name*
カスタムアクション	arn:aws:codepipeline:*region*:*account*:actionty pe:*owner*/*category*/*provider*/*version*
すべてのリソース	arn:aws:codepipeline:*
特定リージョンの特定アカウントが所有するすべての リソース	arn:aws:コードパイプライン:[##]:[#####]:*

> **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、 では、イベントパターンおよびルールで完全一致を使用します。イベントパターンの作成時に正しい ARN 文字を使用して、一致させるリソース内の ARN 構文とそれらの文字が一致する必要があります。

では、リソースレベルのアクセス許可をサポートする API コールがあります。リソースレベルのアクセス許可は、API コールでリソース ARN を指定できるかどうか、または API コールでワイルドカードを使用したすべてのリソースの呼び出しのみが可能かどうかを示します。リソースレベルのアクセス許可の詳細な説明と、リソースレベルのアクセス許可をサポートする CodePipeline API コールの一覧については、アクセス許可に関するリファレンス を参照してください。

例えば、以下の要領で ARN を使用して、ステートメント内の特定のパイプライン (*myPipeline*) を指定することができます。

```
"Resource": "arn:aws:codepipeline:us-east-2:111222333444:myPipeline"
```

また、以下の要領で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パイプラインを指定することもできます。

```
"Resource": "arn:aws:codepipeline:us-east-2:111222333444:*"
```

すべてのリソースを指定する場合、または特定の API アクションが ARN をサポートしていない場合は、以下のように、Resource エレメント内でワイルドカード文字 (*) を使用します。

```
"Resource": "*"
```

> **Note**
>
> IAM ポリシーを作成するとき、最小限の特権を認めるという標準的なセキュリティアドバイスに従いましょう。そうすれば、タスクを実行するというリクエストのアクセス許可のみを認めることができます。API コールが ARN をサポートしている場合、リソースレベルのアクセス許可をサポートしていて、ワイルドカード文字 (*) を使用する必要はありません。

一部の API コールは、複数のリソース (など) を受け入れますGetPipeline。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"]
```

には、 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、「アクセス許可に関するリファレンス」を参照してください。

条件キー

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。

Condition 要素は、定義された基準に基づいてステートメントが実行される時期を指定します。イコールや未満などの条件演算子を使用して条件式を作成して、ポリシーの条件とリクエスト内の値を

一致させることができます。すべての AWS グローバル条件キーを確認するには、「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。

CodePipeline は独自の条件キーを定義し、一部のグローバル条件キーの使用をサポートしています。すべての AWS グローバル条件キーを確認するには、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。

すべての Amazon EC2 アクションは `aws:RequestedRegion` および `ec2:Region` 条件キーをサポートします。詳細については、「例: 特定のリージョンへのアクセスの制限」を参照してください。

CodePipeline 条件キーのリストを確認するには、IAM ユーザーガイド の AWS CodePipelineの条件キーを参照してください。条件キーを使用できるアクションとリソースについては、「 で定義されるアクション AWS CodePipeline」を参照してください。

例

CodePipeline アイデンティティベースのポリシーの例については、AWS CodePipeline 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。

CodePipeline リソースベースのポリシー

CodePipeline は、リソースベースのポリシーに対応していません。ただし、CodePipeline に関連する S3 サービスのリソースベースのポリシーの例が提供されています。

例

CodePipeline リソースベースのポリシーの例を表示する場合、AWS CodePipeline リソースベースのポリシーの例 を参照してください。

CodePipeline タグに基づく許可

CodePipeline リソースにタグをアタッチしたり、リクエスト内のタグを CodePipeline に渡したりできます。タグに基づいてアクセスを制御するには、`codepipeline:ResourceTag/`*`key-name`*、`aws:RequestTag/`*`key-name`*、または `aws:TagKeys` の条件キーを使用して、ポリシーの条件要素 でタグ情報を提供します。CodePipeline リソースのタグ付けの詳細については、リソースのタグ付け を参照してください。

リソースのタグに基づいてリソースへのアクセスを制限するためのアイデンティティベースポリシーの例を表示するには、「タグを使用した CodePipeline リソースへのアクセスのコントロール」を参照してください。

CodePipeline 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。

CodePipeline で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。

CodePipeline は、一時的な認証情報の使用をサポートしています。

サービスロール

CodePipeline は、サービスロールをユーザーに代わって引き受けることをサービスに許可します。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IAM 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、IAM 管理者はこの役割の権限を変更できます。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。

CodePipeline はサービスロールに対応しています。

AWS CodePipeline 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、IAM ユーザーおよびロールには、CodePipeline 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。IAM 管理者は、ユーザーとロールに必要な、指定されたリソースで特定の API オペレーションを実行する権限をユーザーとロールに付与する IAM ポリシーを作成する必要があります。続いて、管理者はそれらの権限が必要な IAM ユーザーまたはグループにそのポリシーをアタッチする必要があります。

JSON ポリシードキュメントのこれらの例を使用して、IAM 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。

別のアカウントのリソースを使用するパイプラインを作成する方法、および関連するポリシーの例については、「別の AWS アカウントのリソースを使用する CodePipeline でパイプラインを作成する」を参照してください。

トピック

- ポリシーに関するベストプラクティス
- コンソールでのリソースの表示
- 自分の権限の表示をユーザーに許可する
- アイデンティティベースのポリシー (IAM) の例
- タグを使用した CodePipeline リソースへのアクセスのコントロール
- コンソールの使用に必要な許可
- コンソールでコンピューティングログを表示するために必要なアクセス許可
- AWS の 管理ポリシー AWS CodePipeline
- 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

ID ベースのポリシーは、ユーザーのアカウント内の CodePipeline 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。

コンソールでのリソースの表示

コンソールには、サインインしている AWS リージョンの AWS アカウントのリポジトリのリストを表示するListRepositoriesアクセス許可が必要です。このコンソールには、大文字と小文字を区別しない検索をリソースに対して迅速に実行するための [Go to resource (リソースに移動)] 機能も含まれています。この検索は、サインインしている AWS リージョンの AWS アカウントで実行されます。次のリソースは、以下のサービス全体で表示されます。

- AWS CodeBuild: プロジェクトの構築
- AWS CodeCommit: リポジトリ
- AWS CodeDeploy: アプリケーション
- AWS CodePipeline: パイプライン

この検索をすべてのサービスのリソースにわたって実行するには、次のアクセス権限が必要です。

- AWS CodeBuild: `ListProjects`
- CodeCommit: `ListRepositories`
- CodeDeploy: `ListApplications`
- CodePipeline: `ListPipelines`

あるサービスに対するアクセス権限がない場合、そのサービスのリソースに関して結果は返されません。表示のアクセス権限がある場合でも、表示に対する明示的な Deny が設定されているリソースについては、結果が返されません。

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": "*"
        }
    ]
```

```
}
```

アイデンティティベースのポリシー (IAM) の例

ポリシーを IAM アイデンティティにアタッチできます。例えば、次のオペレーションを実行できます。

- アカウントのユーザーまたはグループにアクセス許可ポリシーをアタッチする – コンソールでパイプラインを表示するアクセス許可をユーザーに付与するには、ユーザーが属するユーザーまたはグループにアクセス許可ポリシーをアタッチできます。
- アクセス権限ポリシーをロールにアタッチする (クロスアカウントの許可を付与) - ID ベースのアクセス権限ポリシーを IAM ロールにアタッチして、クロスアカウントの権限を付与することができます。たとえば、アカウント A の管理者は、次のように別の AWS アカウント (アカウント B など) または にクロスアカウントアクセス許可を付与するロールを作成できます AWS のサービス 。
  1. アカウント A の管理者は、IAM ロールを作成して、アカウント A のリソースに許可を付与するロールに許可ポリシーをアタッチします。
  2. アカウント A の管理者は、アカウント B をそのロールを引き受けるプリンシパルとして識別するロールに、信頼ポリシーをアタッチします。
  3. アカウント B の管理者は、アカウント B のユーザーにロールを引き受けるアクセス許可を委任できます。これにより、アカウント B のユーザーがアカウント A のリソースを作成またはアクセスできるようになります。ロールを引き受けるアクセス AWS のサービス 許可を付与する場合は、信頼ポリシーのプリンシパルもプリンシ AWS のサービス パルになることができます。

  IAM を使用した許可の委任の詳細については、「IAM ユーザーガイド」の「アクセス管理」を参照してください。

以下に示しているアクセス許可ポリシーの例では、`us-west-2 region` で `MyFirstPipeline` という名前のパイプライン内のすべてのステージ間の移行を無効または有効にするアクセス許可を付与しています。

JSON

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

```
      "Action" : [
        "codepipeline:EnableStageTransition",
        "codepipeline:DisableStageTransition"
      ],
      "Resource" : [
        "arn:aws:codepipeline:us-west-2:111222333444:MyFirstPipeline/*"
      ]
    }
  ]
}
```

次の例は、111222333444 アカウントのポリシーを示しています。このポリシーでは、コンソールMyFirstPipelineで という名前のパイプラインを表示することはできますが、変更することはできません。このポリシーは、AWSCodePipeline_ReadOnlyAccess マネージドポリシーに基づいていますが、パイプライン MyFirstPipeline に固有であるため、このマネージドポリシーを直接使用することはできません。ポリシーを特定のパイプラインに制限しない場合は、 によって作成および保守されているいずれかのマネージドポリシーを使用することを検討してください。詳細については、「マネージドポリシーの使用」を参照してください。このポリシーは、アクセス用に作成した IAM ロール (CrossAccountPipelineViewers という名前のロールなど) にアタッチする必要があります。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "codepipeline:GetPipeline",
        "codepipeline:GetPipelineState",
        "codepipeline:GetPipelineExecution",
        "codepipeline:ListPipelineExecutions",
        "codepipeline:ListActionExecutions",
        "codepipeline:ListActionTypes",
        "codepipeline:ListPipelines",
        "codepipeline:ListTagsForResource",
        "iam:ListRoles",
        "s3:ListAllMyBuckets",
        "codecommit:ListRepositories",
        "codedeploy:ListApplications",
```

```
        "lambda:ListFunctions",
        "codestar-notifications:ListNotificationRules",
        "codestar-notifications:ListEventTypes",
        "codestar-notifications:ListTargets"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:codepipeline:us-west-2:111222333444:MyFirstPipeline"
    },
    {
      "Action": [
        "codepipeline:GetPipeline",
        "codepipeline:GetPipelineState",
        "codepipeline:GetPipelineExecution",
        "codepipeline:ListPipelineExecutions",
        "codepipeline:ListActionExecutions",
        "codepipeline:ListActionTypes",
        "codepipeline:ListPipelines",
        "codepipeline:ListTagsForResource",
        "iam:ListRoles",
        "s3:GetBucketPolicy",
        "s3:GetObject",
        "s3:ListBucket",
        "codecommit:ListBranches",
        "codedeploy:GetApplication",
        "codedeploy:GetDeploymentGroup",
        "codedeploy:ListDeploymentGroups",
        "elasticbeanstalk:DescribeApplications",
        "elasticbeanstalk:DescribeEnvironments",
        "lambda:GetFunctionConfiguration",
        "opsworks:DescribeApps",
        "opsworks:DescribeLayers",
        "opsworks:DescribeStacks"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Sid": "CodeStarNotificationsReadOnlyAccess",
      "Effect": "Allow",
      "Action": [
        "codestar-notifications:DescribeNotificationRule"
      ],
      "Resource": "*",
      "Condition": {
```

```
        "ArnLike": {
          "codestar-notifications:NotificationsForResource":
 "arn:aws:iam::*:role/Service*"
        }
      }
    }
  ]
}
```

このポリシーを作成したら、アカウント (111222333444) に IAM ロールを作成し、そのロールにポリシーをアタッチします。ロールの信頼関係で、このロールを引き受ける AWS アカウントを追加する必要があります。次の例は、*111111111111* AWS アカウントのユーザーが アカウントで定義されたロールを引き受けることを許可するポリシーを示しています111222333444。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111111111111:root"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

次の例は、ユーザーが *111111111111* AWS アカウントで *CrossAccountPipelineViewers* という名前のロールを引き受けることを許可する 111222333444 アカウントで作成されたポリシーを示しています。

JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource":
 "arn:aws:iam::111222333444:role/CrossAccountPipelineViewers"
        }
    ]
}
```

アカウントのユーザーがアクセスを許可される通話とリソースを制限する IAM ポリシーを作成し、管理者ユーザーにそれらのポリシーをアタッチできます。IAM ロールの作成方法と の IAM ポリシーステートメントの例については、「」を参照してくださいカスタマーマネージドポリシーの例。

タグを使用した CodePipeline リソースへのアクセスのコントロール

IAM ポリシーステートメントの条件は、CodePipeline アクションに必要なリソースへのアクセス許可を指定するために使用する構文の一部です。条件内でタグを使用することは、リソースとリクエストへのアクセスをコントロールするひとつの方法です。CodePipeline リソースのタグ付けの詳細情報は、リソースのタグ付け を参照してください。このトピックでは、タグベースのアクセスコントロールについて説明します。

IAM ポリシーの設計時に特定のリソースへのアクセス権を付与することで、詳細なアクセス許可を設定できます。管理するリソースの数が増えるに従って、このタスクはより困難になります。リソースにタグ付けしてポリシーステートメント条件でタグを使用することにより、このタスクをより容易にすることができます。特定のタグを使用して任意のリソースへのアクセス権を一括して付与します。次に、作成時や以降の段階で、このタグを関連リソースに繰り返し適用します。

タグは、リソースにアタッチしたり、タグ付けをサポートするサービスへのリクエストに渡したりすることができます。CodePipeline では、リソースにタグを付けることができ、一部のアクションにタグを含めることができます。IAM ポリシーを作成するときに、タグ条件キーを使用して以下をコントロールできます。

- どのユーザーがパイプラインリソースに対してアクションを実行できるか (リソースに既に付けられているタグに基づいて)。
- どのタグをアクションのリクエストで渡すことができるか。
- リクエストで特定のタグキーを使用できるかどうか。

文字列条件演算子では、キーと文字列値の比較に基づいてアクセスを制限する Condition 要素を構築できます。Null 条件以外の条件演算子名の末尾に IfExists を追加できます。「ポリシーキーがリクエストのコンテキストで存在する場合、ポリシーで指定されたとおりにキーを処理します。キーが存在しない場合、条件要素は true と評価されます。」 例えば、StringEqualsIfExists を使用して、他のタイプのリソースには存在しない可能性のある条件キーに基づいた制限を行うことができます。

タグ条件キーの完全な構文と意味については、「タグを使用したアクセスコントロール」を参照してください。条件キーの詳細については、以下のリソースを参照してください。このセクションに示す CodePipeline ポリシーの例は、条件キーに関する以下の情報に沿っています。また、リソースのネスティングなど CodePipeline 特有の考慮点についても、例を交えて補足説明しています。

- 文字列条件演算子
- AWS のサービス IAM で動作する
- SCP 構文
- IAM JSON ポリシーエレメント: 条件
- aws:RequestTag/tag-key
- CodePipeline の条件キー

次の例は、CodePipeline ユーザー用のポリシーでタグ条件を指定する方法を示しています。

Example 1: リクエストのタグに基づいてアクションを制限する

AWSCodePipeline_FullAccess マネージドユーザーポリシーは、すべてのリソースに対して任意の CodePipeline アクションを実行する無制限のアクセス許可をユーザーに付与します。

以下ののポリシーでは、この権限を制限し、権限のないユーザーがリクエストに特定のタグが記載されたパイプラインを作成することを許可しません。これを行うには、リクエストに指定されているタグ Project の値が ProjectA または ProjectB のいずれかである場合、CreatePipeline アクションを拒否します。(この aws:RequestTag 条件キーを使用して、IAM リクエストで渡すことができるタグをコントロールします)。

以下の例で、ポリシーの目的は、指定したタグ値を使用してパイプラインを作成するアクセス許可を、権限のないユーザーに与えないことです。ただし、パイプラインを作成するには、パイプライン自体に加えてリソース (パイプラインのアクションやステージなど) にアクセスする必要があります。ポリシーに指定された 'Resource' が '*' であるため、このポリシーは、ARN の付いたリソースのうち、パイプラインの作成時に作成されるすべてのリソースに適用されます。これらの追加のリソースにはタグ条件キーがないため、StringEquals のチェックは失敗し、ユーザーはいずれ

のタイプのインスタンスも起動できません。これに対応するには、StringEqualsIfExists 条件演算子を代わりに使用します。そうすれば、条件キーが存在する場合のみにテストが行われます。

以下のコードは次のように解釈できます。「チェックされるリソースには "RequestTag/Project" 条件キーがあり、キー値が projectA で始まる場合にのみ、アクションを許可します。チェックされるリソースにこの条件キーがなくても問題ありません。」

また、このポリシーでは aws:TagKeys 条件キーを使用して、タグ変更アクションにこれらの同じタグ値を含めることを許可しないことで、これらの権限のないユーザーがリソースを改ざんするのを防ぎます。お客様の管理者は、権限のない管理者ユーザーには、マネージドユーザーポリシーに加えて、この IAM ポリシーをアタッチする必要があります。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "codepipeline:CreatePipeline",
        "codepipeline:TagResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEqualsIfExists": {
          "aws:RequestTag/Project": ["ProjectA", "ProjectB"]
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": [
        "codepipeline:UntagResource"
      ],
      "Resource": "*",
      "Condition": {
        "ForAllValues:StringEquals": {
          "aws:TagKeys": ["Project"]
        }
      }
    }
```

```
  ]
}
```

Example 2: リソースタグに基づいてタグ付けアクションを制限する

AWSCodePipeline_FullAccess マネージドユーザーポリシーは、すべてのリソースに対して任意の CodePipeline アクションを実行する無制限のアクセス許可をユーザーに付与します。

次のポリシーは、この権限を制限し、権限のないユーザーに対して特定のプロジェクトのパイプラインでアクションを実行するアクセス許可を拒否します。そのために、ProjectA または ProjectB の値のいずれかを持つ Project という名前のタグをこのリソースが持つ場合、一部のアクションを拒否します。(このaws:ResourceTag 条件キーを使用して、それらのリソースのタグに基づいて、このリソースへのアクセスをコントロールします)。お客様の管理者は、権限のない IAM ユーザーには、マネージドユーザーポリシーに加えて、この IAM ポリシーをアタッチする必要があります。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "codepipeline:TagResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Project": ["ProjectA", "ProjectB"]
        }
      }
    }
  ]
}
```

Example 3: リクエストのタグに基づいてアクションを許可する

次のポリシーでは、CodePipeline の開発パイプラインを作成するアクセス許可をユーザーに付与します。

これを行うには、リクエストに指定されているタグ Project の値が ProjectA である場合に、CreatePipeline アクションと TagResource アクションを許可します。つまり、指定できるタグキーは Project のみで、その値は ProjectA であることが必要です。

この aws:RequestTag 条件キーを使用して、IAM リクエストで渡すことができるタグをコントロールします。aws:TagKeys 条件は、タグキーの大文字と小文字を区別します。このポリシーは、AWSCodePipeline_FullAccess マネージドユーザーポリシーがアタッチされていないユーザーまたはロールに便利です。このマネージドポリシーは、すべてのリソースに対して任意の CodePipeline アクションを実行する無制限のアクセス許可をユーザーに付与します。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codepipeline:CreatePipeline",
        "codepipeline:TagResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/Project": "ProjectA"
        },
        "ForAllValues:StringEquals": {
          "aws:TagKeys": ["Project"]
        }
      }
    }
  ]
}
```

Example 4: リソースタグに基づいてタグ削除アクションを制限する

AWSCodePipeline_FullAccess マネージドユーザーポリシーは、すべてのリソースに対して任意の CodePipeline アクションを実行する無制限のアクセス許可をユーザーに付与します。

次のポリシーは、この権限を制限し、権限のないユーザーに対して特定のプロジェクトのパイプラインでアクションを実行するアクセス許可を拒否します。そのために、ProjectA または ProjectB の値のいずれかを持つ Project という名前のタグをこのリソースが持つ場合、一部のアクションを拒否します。

また、このポリシーでは aws:TagKeys 条件キーを使用して、タグ変更アクションに Project タグを完全に削除することを許可しないことで、これらの権限のないユーザーがリソースを改ざんするのを防ぎます。お客様の管理者は、権限のないユーザーまたはロールには、マネージドユーザーポリシーに加えて、この IAM ポリシーをアタッチする必要があります。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "codepipeline:UntagResource"
      ],
      "Resource": "*",
      "Condition": {
        "ForAllValues:StringEquals": {
          "aws:TagKeys": ["Project"]
        }
      }
    }
  ]
}
```

コンソールの使用に必要な許可

コンソールで を使用するには、以下のサービスからの最小限のアクセス許可が必要です。

- AWS Identity and Access Management
- Amazon Simple Storage Service

これらのアクセス許可により、アカウントの他の AWS リソースを記述できます AWS 。

他のサービスをパイプラインに取り入れた場合は、次のアクセス許可が 1 つ以上必要になることがあります。

- AWS CodeCommit
- AWS CodeBuild
- CloudFormation
- AWS CodeDeploy
- AWS Elastic Beanstalk
- AWS Lambda
- AWS OpsWorks

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。「AWSCodePipeline_ReadOnlyAccess」で説明されているとおり、ユーザーがコンソールを使用できること、および AWS の 管理ポリシー AWS CodePipeline 管理ポリシーがユーザーにアタッチされていることを確認してください。

AWS CLI または API 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。

コンソールでコンピューティングログを表示するために必要なアクセス許可

CodePipeline コンソールでコマンドアクションのログを表示するには、コンソールロールにアクセス許可が必要です。コンソールでログを表示するには、コンソールロールに `logs:GetLogEvents` アクセス許可を追加します。

コンソールロールポリシーステートメントで、次の例に示すように、アクセス許可の範囲をパイプラインレベルに絞り込みます。

```
{
    "Effect": "Allow",
    "Action": [
        "Action": "logs:GetLogEvents"
    ],
    "Resource": "arn:aws:logs:*:YOUR_AWS_ACCOUNT_ID:log-group:/aws/
codepipeline/YOUR_PIPELINE_NAME:*"
}
```

AWS の 管理ポリシー AWS CodePipeline

AWS 管理ポリシーは、 によって作成および管理されるスタンドアロンポリシーです AWS。 AWS 管理ポリシーは、ユーザー、グループ、ロールにアクセス許可の割り当てを開始できるように、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ように設計されています。

AWS 管理ポリシーは、すべての AWS お客様が使用できるため、特定のユースケースに対して最小特権のアクセス許可を付与しない場合があることに注意してください。ユースケースに固有のカスタマー管理ポリシーを定義して、アクセス許可を絞り込むことをお勧めします。

AWS 管理ポリシーで定義されているアクセス許可は変更できません。が AWS 管理ポリシーで定義されたアクセス許可 AWS を更新すると、ポリシーがアタッチされているすべてのプリンシパル ID (ユーザー、グループ、ロール) に影響します。 AWS は、新しい が起動されるか、新しい API オペレーション AWS のサービス が既存のサービスで使用できるようになったときに、 AWS 管理ポリシーを更新する可能性が高くなります。

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。

> **Important**
>
> AWS マネージドポリシー `AWSCodePipelineFullAccess` および `AWSCodePipelineReadOnlyAccess` は置き換えられました。`AWSCodePipeline_FullAccess` および `AWSCodePipeline_ReadOnlyAccess` ポリシーを使用してください。

AWS 管理ポリシー: **AWSCodePipeline_FullAccess**

これは CodePipeline へのフルアクセスを許可するポリシーです。IAM コンソールで JSON ポリシードキュメントを表示するには、「AWSCodePipeline_FullAccess」を参照してください。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- codepipeline - CodePipeline に対するアクセス許可を付与します。
- chatbot - チャットアプリケーションで Amazon Q Developer のリソースを管理するためのアクセス許可をプリンシパルに付与します。
- cloudformation – プリンシパルがリソーススタックを管理できるようにするアクセス許可を付与します CloudFormation。
- cloudtrail - プリンシパルに、CloudTrail でリソースのログ記録を管理するためのアクセス許可を付与します。
- codebuild - プリンシパルに、CodeBuild でビルドリソースを利用するためのアクセス許可を付与します。
- codecommit - プリンシパルに、CodeCommit でソースリソースを利用するためのアクセス許可を付与します。
- codedeploy - プリンシパルに、CodeDeploy でデプロイリソースを利用するためのアクセス許可を付与します。
- codestar-notifications – プリンシパルが AWS CodeStar Notifications のリソースにアクセスできるようにするアクセス許可を付与します。
- ec2 - CodeCatalyst でのデプロイにおいて、Amazon EC2 で Elastic Load Balancing を管理するためのアクセス許可を付与します。
- ecr - Amazon ECR でリソースを利用するためのアクセス許可を付与します。
- elasticbeanstalk - プリンシパルに、Elastic Beanstalk でリソースを利用するためのアクセス許可を付与します。
- iam - プリンシパルに、IAM でロールとポリシーを管理するためのアクセス許可を付与します。
- lambda - プリンシパルに、Lambda でリソースを管理するためのアクセス許可を付与します。
- events - プリンシパルに、CloudWatch Events でリソースを管理するためのアクセス許可を付与します。
- opsworks – プリンシパルが のリソースを管理できるようにするアクセス許可を付与します AWS OpsWorks。
- s3 - プリンシパルに、Amazon S3 でリソースを管理するためのアクセス許可を付与します。
- sns - プリンシパルに、Amazon SNS で通知リソースを管理するためのアクセス許可を付与します。

- `states` – プリンシパルがステートマシンを表示できるようにするアクセス許可を付与します AWS Step Functions。ステートマシンは、状態の集まりで構成され、それぞれの状態がタスクを管理し、状態間の遷移を制御します。

ポリシーについては、「AWSCodePipeline_FullAccess」を参照してください。

AWS 管理ポリシー: **AWSCodePipeline_ReadOnlyAccess**

これは CodePipeline への読み取り専用アクセスを許可するポリシーです。IAM コンソールで JSON ポリシードキュメントを表示するには、「AWSCodePipeline_ReadOnlyAccess」を参照してください。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- `codepipeline` - CodePipeline でのアクションに対するアクセス許可を付与します。
- `codestar-notifications` – プリンシパルが AWS CodeStar Notifications のリソースにアクセスできるようにするアクセス許可を付与します。
- `s3` - プリンシパルに、Amazon S3 でリソースを管理するためのアクセス許可を付与します。
- `sns` - プリンシパルに、Amazon SNS で通知リソースを管理するためのアクセス許可を付与します。

ポリシーについては、「AWSCodePipeline_ReadOnlyAccess」を参照してください。

AWS 管理ポリシー: **AWSCodePipelineApproverAccess**

これは、手動承認アクションを承認または拒否するためのアクセス許可を付与するポリシーです。IAM コンソールで JSON ポリシードキュメントを表示するには、「AWSCodePipelineApproverAccess」を参照してください。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- `codepipeline` - CodePipeline でのアクションに対するアクセス許可を付与します。

ポリシーについては、「AWSCodePipelineApproverAccess」を参照してください。

AWS 管理ポリシー: **`AWSCodePipelineCustomActionAccess`**

これは、CodePipeline でカスタムアクションを作成するか、ビルドアクションまたはテストアクションのために Jenkins リソースを統合するアクセス許可を付与するポリシーです。IAM コンソールで JSON ポリシードキュメントを表示するには、「AWSCodePipelineCustomActionAccess」を参照してください。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- `codepipeline` - CodePipeline でのアクションに対するアクセス許可を付与します。

ポリシーについては、「AWSCodePipelineCustomActionAccess」を参照してください。

CodePipeline のマネージドポリシーと通知

CodePipeline は、パイプラインへの重要な変更をユーザーに通知できる通知機能をサポートしています。CodePipeline 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の通知に関連するアクセス許可

この管理ポリシーは、CodeCommit、CodeBuild、CodeDeploy、および Notifications の関連サービスとともに CodePipeline のアクセス許可を付与します。 CodeCommit CodeBuild CodeDeploy AWS CodeStar このポリシーは、Amazon S3、Elastic Beanstalk、CloudTrail、Amazon EC2、CloudFormationなど、パイプラインと統合する他のサービスで作業するためのアクセス許可も付与

します。これらのマネージドポリシーのいずれかが適用されたユーザーは、通知の Amazon SNS トピックの作成と管理、トピックに対するユーザーのサブスクライブとサブスクライブ解除、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、Slack 用に設定されたチャットアプリケーションクライアントで Amazon Q Developer の一覧表示を行うこともできます。

AWSCodePipeline_FullAccess 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。

```
    {
        "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" : {
            "StringLike" : {"codestar-notifications:NotificationsForResource" :
 "arn:aws:codepipeline:us-west-2:111222333444:MyFirstPipeline"}
        }
    },
    {
        "Sid": "CodeStarNotificationsListAccess",
        "Effect": "Allow",
        "Action": [
            "codestar-notifications:ListNotificationRules",
            "codestar-notifications:ListTargets",
            "codestar-notifications:ListTagsforResource",
            "codestar-notifications:ListEventTypes"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CodeStarNotificationsSNSTopicCreateAccess",
        "Effect": "Allow",
        "Action": [
            "sns:CreateTopic",
            "sns:SetTopicAttributes"
        ],
```

```
        "Resource": "arn:aws:sns:*:*:codestar-notifications*"
    },
    {
        "Sid": "SNSTopicListAccess",
        "Effect": "Allow",
        "Action": [
            "sns:ListTopics"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CodeStarNotificationsChatbotAccess",
        "Effect": "Allow",
        "Action": [
            "chatbot:DescribeSlackChannelConfigurations",
            "chatbot:ListMicrosoftTeamsChannelConfigurations"
          ],
       "Resource": "*"
    }
```

読み取り専用マネージドポリシーの通知に関連するアクセス許可

AWSCodePipeline_ReadOnlyAccess 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。このポリシーを適用したユーザーは、リソースの通知を表示できますが、リソースを作成、管理、サブスクライブすることはできません。

```
  {
        "Sid": "CodeStarNotificationsPowerUserAccess",
        "Effect": "Allow",
        "Action": [
            "codestar-notifications:DescribeNotificationRule"
        ],
        "Resource": "*",
        "Condition" : {
            "StringLike" : {"codestar-notifications:NotificationsForResource" :
 "arn:aws:codepipeline:us-west-2:111222333444:MyFirstPipeline"}
        }
    },
    {
        "Sid": "CodeStarNotificationsListAccess",
        "Effect": "Allow",
        "Action": [
            "codestar-notifications:ListNotificationRules",
```

```
            "codestar-notifications:ListEventTypes",
            "codestar-notifications:ListTargets"
        ],
        "Resource": "*"
    }
```

IAM と通知の詳細については、「AWS CodeStar Notifications の Identity and Access Management」を参照してください。

AWS CodePipeline AWS 管理ポリシーの更新

このサービスがこれらの変更の追跡を開始してからの CodePipeline の AWS マネージドポリシーの更新に関する詳細を表示します。このページの変更に関する自動通知を受け取るには、CodePipeline の [ドキュメント履歴] ページで RSS フィードをサブスクライブしてください。

変更	説明	日付
AWSCodePipeline_FullAccess - 既存のポリシーの更新	CodePipeline は、`ListStacks` を CloudFormationでサポートするためのアクセス許可を、このポリシーに追加しました。	2024 年 3 月 15 日
AWSCodePipeline_FullAccess - 既存のポリシーの更新	このポリシーが更新され、チャットアプリケーションで Amazon Q Developer のアクセス許可が追加されました。詳細については、「CodePipeline のマネージドポリシーと通知」を参照してください。	2023 年 6 月 21 日
AWSCodePipeline_FullAccess および AWSCodePipeline_ReadOnlyAccess マネージドポリシー - 既存のポリシーの更新	CodePipeline はこれらのポリシーにアクセス許可を追加し、チャットアプリケーションで Amazon Q Developer (`chatbot:ListMicrosoftTeamsChannelConfigurations`) を使用し	2023 年 5 月 16 日

変更	説明	日付
	て、追加の通知タイプをサポートするようになりました。	
AWSCodePipelineFullAccess - 廃止	このポリシーは AWSCodePipeline_FullAccess に置き換えられました。 2022 年 11 月 17 日以降、このポリシーは新しいユーザー、グループ、またはロールにアタッチできなくなりました。詳細については、「AWS の 管理ポリシー AWS CodePipeline」を参照してください。	2022 年 11 月 17 日
AWSCodePipelineReadOnlyAccess - 廃止	このポリシーは AWSCodePipeline_ReadOnlyAccess に置き換えられました。 2022 年 11 月 17 日以降、このポリシーは新しいユーザー、グループ、またはロールにアタッチできなくなりました。詳細については、「AWS の 管理ポリシー AWS CodePipeline」を参照してください。	2022 年 11 月 17 日
CodePipeline が変更の追跡を開始	CodePipeline は AWS 、管理ポリシーの変更の追跡を開始しました。	2021 年 3 月 12 日

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 アクションのアクセス許可を付与するユーザーポリシーの例を示しています。これらのポリシーは、API、 AWS SDKs、または を使用している場合に機能します AWS CLI。コンソールを使用している場合は、コンソールに固有の追加のアクセス許可を付与する必要があります。詳細については、「コンソールの使用に必要な許可」を参照してください。

> **Note**
>
> 各例は全て、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: パイプラインの状態を取得するアクセス許可を付与する
- 例 2: ステージ間の移行を有効または無効にするアクセス許可を付与する
- 例 3: 使用可能なすべてのアクションタイプのリストを取得するアクセス許可を付与する
- 例 4: 手動の承認アクションを承認または拒否するアクセス許可を付与する
- 例 5: カスタムアクションのジョブをポーリングするアクセス許可を付与する
- 例 6: Jenkins と AWS CodePipeline の統合に関するポリシーをアタッチまたは編集する
- 例 7: パイプラインへのクロスアカウントアクセスを設定する

例 1: パイプラインの状態を取得するアクセス許可を付与する

以下の例では、パイプライン (MyFirstPipeline) の状態を取得する権限を付与します。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:GetPipelineState"
```

```
            ],
            "Resource": "arn:aws:codepipeline:us-
west-2:111222333444:MyFirstPipeline"
        }
    ]
}
```

例 2: ステージ間の移行を有効または無効にするアクセス許可を付与する

以下の例では、パイプライン (MyFirstPipeline) のすべてのステージ間での移行を無効化または有効化するアクセス許可を付与します。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:DisableStageTransition",
                "codepipeline:EnableStageTransition"
            ],
            "Resource": "arn:aws:codepipeline:us-
west-2:111222333444:MyFirstPipeline/*"
        }
    ]
}
```

ユーザーが、パイプラインの 1 つのステージでの移行を無効化または有効化できるようにするには、ステージを指定する必要があります。例えば、ユーザーがパイプライン Staging のステージ MyFirstPipeline の移行を有効化または無効化できるようにするには、以下を行います。

```
"Resource": "arn:aws:codepipeline:us-west-2:111222333444:MyFirstPipeline/Staging"
```

例 3: 使用可能なすべてのアクションタイプのリストを取得するアクセス許可を付与する

以下の例では、us-west-2 リージョンのパイプラインで利用できるすべてのアクションの種類のリストを取得するためのアクセス許可を付与します。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:ListActionTypes"
            ],
            "Resource": "arn:aws:codepipeline:us-
west-2:111222333444:actiontype:*"
        }
    ]
}
```

例 4: 手動の承認アクションを承認または拒否するアクセス許可を付与する

以下の例では、パイプライン MyFirstPipeline のステージ Staging で手動の承認アクションを承認または拒否するためのアクセス許可を付与します。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:PutApprovalResult"
            ],
            "Resource": "arn:aws:codepipeline:us-
west-2:111222333444:MyFirstPipeline/Staging/*"
        }
    ]
}
```

例 5: カスタムアクションのジョブをポーリングするアクセス許可を付与する

以下の例では、カスタムアクション TestProvider のジョブをポーリングするためのアクセス許可を付与します。これは、すべてのパイプラインで最初のバージョンのアクションの種類である Test です。

> **Note**
>
> カスタムアクションのジョブワーカーは、異なる AWS アカウントを使用して設定するか、特定の IAM ロールで実行する必要があります。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:PollForJobs"
            ],
            "Resource": [
                "arn:aws:codepipeline:us-
west-2:111222333444:actionType:Custom/Test/TestProvider/1"
            ]
        }
    ]
}
```

例 6: Jenkins と AWS CodePipeline の統合に関するポリシーをアタッチまたは編集する

ビルドまたはテストに Jenkins を使用するようにパイプラインを設定する場合は、その統合用に別の ID を作成し、Jenkins と の統合に必要な最小限のアクセス許可を持つ IAM ポリシーをアタッチします。このポリシーは、AWSCodePipelineCustomActionAccess マネージドポリシーと同じです。以下の例では、Jenkins との統合で使用するポリシーを示します。

JSON

```
{
    "Version":"2012-10-17",

    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:AcknowledgeJob",
                "codepipeline:GetJobDetails",
                "codepipeline:PollForJobs",
                "codepipeline:PutJobFailureResult",
                "codepipeline:PutJobSuccessResult"
            ],
            "Resource": "*"
        }
    ]
}
```

例 7: パイプラインへのクロスアカウントアクセスを設定する

別の AWS アカウントのユーザーおよびグループのパイプラインへのアクセスを設定できます。推奨される方法は、パイプラインが作成されたアカウントにロールを作成することです。ロールは、他の AWS アカウントのユーザーがそのロールを引き受けてパイプラインにアクセスすることを許可する必要があります。詳細については、「ウォークスルー: ロールを使用したクロスアカウントアクセス」を参照してください。

次の例は、80398EXAMPLE アカウントのポリシーを示しています。このポリシーでは、コンソールMyFirstPipelineで という名前のパイプラインを表示することはできますが、変更することはできません。このポリシーは、AWSCodePipeline_ReadOnlyAccess マネージドポリシーに基づいていますが、パイプライン MyFirstPipeline に固有であるため、このマネージドポリシーを直接使用することはできません。ポリシーを特定のパイプラインに制限しない場合は、 によって作成および保守されているいずれかのマネージドポリシーを使用することを検討してください。詳細については、「マネージドポリシーの使用」を参照してください。このポリシーは、アクセス用に作成した IAM ロール (CrossAccountPipelineViewers という名前のロールなど) にアタッチする必要があります。

JSON

```
{
    "Version":"2012-10-17",

    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:GetPipeline",
                "codepipeline:GetPipelineState",
                "codepipeline:ListActionTypes",
                "codepipeline:ListPipelines",
                "iam:ListRoles",
                "s3:GetBucketPolicy",
                "s3:GetObject",
                "s3:ListAllMyBuckets",
                "s3:ListBucket",
                "codedeploy:GetApplication",
                "codedeploy:GetDeploymentGroup",
                "codedeploy:ListApplications",
                "codedeploy:ListDeploymentGroups",
                "elasticbeanstalk:DescribeApplications",
                "elasticbeanstalk:DescribeEnvironments",
                "lambda:GetFunctionConfiguration",
                "lambda:ListFunctions"
            ],
            "Resource": "arn:aws:codepipeline:us-
east-2:111122223333:MyFirstPipeline"
        }
    ]
}
```

このポリシーを作成したら、アカウント (80398EXAMPLE) に IAM ロールを作成し、そのロールにポリシーをアタッチします。ロールの信頼関係で、このロールを引き受ける AWS アカウントを追加する必要があります。

次の例は、ユーザーが 80398EXAMPLE アカウントで という名前のロールを引き受けることを許可する *111111111111* AWS アカウントCrossAccountPipelineViewersで作成されたポリシーを示しています。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource":
 "arn:aws:iam::111122223333:role/CrossAccountPipelineViewers"
        }
    ]
}
```

AWS CodePipeline リソースベースのポリシーの例

Simple Storage Service (Amazon S3) などの他のサービスでは、リソースベースの許可ポリシーもサポートされています。例えば、ポリシーを S3 バケットにアタッチして、そのバケットに対するアクセス許可を管理できます。CodePipeline は、リソースベースのポリシーをサポートしていませんが、バージョニングされた S3 バケットのパイプラインで使用されるアーティファクトを保存します。

Example S3 バケットのポリシーを作成して、CodePipeline のアーティファクトストアとして使用する方法

バージョニングされた S3 バケットを CodePipeline のアーティファクトストアとして使用します。[パイプラインの作成] ウィザードを使用して最初のパイプラインを作成した場合、この S3 バケットは、アーティファクトストアにアップロードされているすべてのオブジェクトが暗号化されており、バケットへの接続が安全であることを保証するために作成されます。ベストプラクティスとして、独自の S3 バケットを作成する場合は、以下のポリシーまたはその要素をバケットに追加することを検討してください。このポリシーでは、S3 バケットの ARN は `codepipeline-us-east-2-1234567890` です。この ARN を S3 バケットの ARN に置き換えます。

JSON

```
{
    "Version":"2012-10-17",
    "Id": "SSEAndSSLPolicy",
```

```
    "Statement": [
        {
            "Sid": "DenyUnEncryptedObjectUploads",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
            "Condition": {
                "StringNotEquals": {
                    "s3:x-amz-server-side-encryption": "aws:kms"
                }
            }
        },
        {
            "Sid": "DenyInsecureConnections",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
            "Condition": {
                "Bool": {
                    "aws:SecureTransport": false
                }
            }
        }
    ]
}
```

AWS CodePipeline ID とアクセスのトラブルシューティング

次の情報は、CodePipeline 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

- CodePipeline でアクションを実行する権限がない
- iam:PassRole を実行する権限がありません
- 管理者として CodePipeline へのアクセスを他のユーザーに許可したい
- AWS アカウント以外のユーザーに CodePipeline リソースへのアクセスを許可したい

CodePipeline でアクションを実行する権限がない

にアクションを実行する権限がないと AWS マネジメントコンソール 通知された場合は、管理者に連絡してサポートを依頼する必要があります。お客様のユーザー名とパスワードを発行したのが、担当の管理者です。

以下の例のエラーは、mateojackson IAM ユーザーがコンソールを使用してパイプラインの詳細を表示しようとしているが、 codepipeline:GetPipeline の許可がない場合に発生します。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 codepipeline:GetPipeline on resource: my-pipeline
```

この場合、Mateo は、codepipeline:GetPipeline アクションを使用して my-pipeline リソースにアクセスできるように、管理者にポリシーの更新を依頼します。

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ありませんというエラーが表示された場合、管理者に問い合わせ、サポートを依頼する必要があります。お客様のユーザー名とパスワードを発行したのが、担当の管理者です。CodePipeline にロールを渡すことができるようにポリシーを更新するよう、管理者に依頼します。

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。

以下の例のエラーは、marymajor という IAM ユーザーがコンソールを使用して CodePipeline でアクションを実行しようする場合に発生します。ただし、アクションには、サービスロールによってサービスに許可が付与されている必要があります。Mary には、ロールをサービスに渡す許可がありません。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

この場合、Mary は iam:PassRole アクションの実行が許可されるように、担当の管理者にポリシーの更新を依頼します。

管理者として CodePipeline へのアクセスを他のユーザーに許可したい

CodePipeline へのアクセスを他のユーザーに許可するには、アクセスを必要とするユーザーまたはアプリケーションにアクセス許可を付与する必要があります。 AWS IAM アイデンティティセンター

を使用してユーザーとアプリケーションを管理する場合は、アクセスレベルを定義するアクセス許可セットをユーザーまたはグループに割り当てます。アクセス許可セットは、ユーザーまたはアプリケーションに関連付けられている IAM ロールに自動的に IAM ポリシーを作成して割り当てます。詳細については、「AWS IAM アイデンティティセンター ユーザーガイド」の「アクセス許可セット」を参照してください。

IAM アイデンティティセンターを使用していない場合は、アクセスを必要とするユーザーまたはアプリケーションの IAM エンティティ (ユーザーまたはロール) を作成する必要があります。次に、CodePipeline の適切なアクセス許可を付与するポリシーを、そのエンティティにアタッチする必要があります。アクセス許可が付与されたら、ユーザーまたはアプリケーション開発者に認証情報を提供します。これらの認証情報を使用して AWSにアクセスします。IAM ユーザー、グループ、ポリシー、アクセス許可の作成の詳細については、「IAM ユーザーガイド」の「IAM アイデンティティ」と「IAM のポリシーとアクセス許可」を参照してください。

AWS アカウント以外のユーザーに CodePipeline 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:

- CodePipeline の今後の対応予定機能の詳細は、が IAM と AWS CodePipeline 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。

アクセス許可に関するリファレンス

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コントロールの設定やアクセス許可ポリシーのを作成する際に、以下の表をリファレンスとして使用します。この表には、各 API オペレーション、およびその実行のためのアクセス権限を付与できる対応するアクションを示しています。リソースレベルのアクセス許可をサポートするオペレーションの場合、テーブルには、アクセス許可を付与できる AWS リソースが一覧表示されます。アクションは、ポリシーの `Action` フィールドで指定します。

リソースレベルのアクセス許可は、ユーザーがアクションを実行できるリソースを指定できるアクセス許可です。 AWS CodePipeline は、リソースレベルのアクセス許可を部分的にサポートします。つまり、一部の AWS CodePipeline API コールでは、満たす必要がある条件に基づいてユーザーがこれらのアクションを使用できるタイミングや、ユーザーが使用できるリソースを制御できます。例えば、パイプラインの実行情報を一覧表示できるが、特定のパイプラインに限定するアクセス許可をユーザーに付与できます。

> **Note**
>
> [リソース] 列には、リソースレベルのアクセス許可をサポートする API コールに必要なリソースが一覧表示されます。リソースレベルのアクセス許可をサポートしない API コールの場合は、それを使用するためのアクセス許可をユーザーに付与できますが、ポリシーステートメントのリソース要素でワイルドカード (*) を指定する必要があります。

API オペレーションおよびアクションで必要なアクセス権限

AcknowledgeJob

アクション:`codepipeline:AcknowledgeJob`

特定のジョブに関する情報や、そのジョブがジョブワーカーで受け取られたかどうかについて表示するために必要です。カスタムアクションにのみ使用されます。

リソース: ポリシーの `Resource` 要素ではワイルドカード (*) のみがサポートされます。

AcknowledgeThirdPartyJob

アクション:`codepipeline:AcknowledgeThirdPartyJob`

ジョブワーカーが特定のジョブを受け取ったことを確認するために必要です。パートナーアクションにのみ使用されます。

リソース: ポリシーの Resource 要素ではワイルドカード (*) のみがサポートされます。

CreateCustomActionType

アクション:codepipeline:CreateCustomActionType

AWS アカウントに関連付けられたすべてのパイプラインで使用できる新しいカスタムアクションを作成するために必要です。カスタムアクションにのみ使用されます。

リソース:

アクションの種類

arn:aws:codepipeline:*region*:*account*:actiontype:*owner*/*category*/*provider*/*versio*

CreatePipeline

アクション:codepipeline:CreatePipeline

パイプラインを作成するために必要です。

リソース:

パイプライン

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

DeleteCustomActionType

アクション:codepipeline:DeleteCustomActionType

カスタムアクションを削除済みとしてマークするために必要です。カスタムアクションの PollForJobs は、アクションが削除対象としてマークされた後は失敗します。カスタムアクションにのみ使用されます。

リソース:

アクションの種類

arn:aws:codepipeline:*region*:*account*:actiontype:*owner*/*category*/*provider*/*versio*

DeletePipeline

アクション:codepipeline:DeletePipeline

パイプラインを削除するために必要です。

リソース:

パイプライン

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

DeleteWebhook

アクション: codepipeline:DeleteWebhook

Webhook を削除するために必要です。

リソース:

ウェブフック

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

DeregisterWebhookWithThirdParty

アクション: codepipeline:DeregisterWebhookWithThirdParty

ウェブフックが削除される前に、CodePipeline で作成されたウェブフックと検出されるイベントの外部ツールとの間の接続を削除するように要求されます。現在のところ、GitHub のアクションの種類をターゲットとするウェブフックでのみサポートされています。

リソース:

ウェブフック

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

DisableStageTransition

アクション:codepipeline:DisableStageTransition

パイプラインのアーティファクトが、パイプラインの次のステージに移行しないようにするために必要です。

リソース:

パイプライン

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

EnableStageTransition

アクション:codepipeline:EnableStageTransition

パイプラインのアーティファクトが、パイプラインのステージに移行できるようにするために必要です。

リソース:

パイプライン

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

GetJobDetails

アクション:codepipeline:GetJobDetails

ジョブに関する情報を取得するために必要です。カスタムアクションにのみ使用されます。

リソース: リソースは必要ありません。

GetPipeline

アクション:codepipeline:GetPipeline

パイプライン ARN を含む、パイプラインの構造、ステージ、アクション、およびメタデータを取得するために必要です。

リソース:

パイプライン

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

GetPipelineExecution

アクション:codepipeline:GetPipelineExecution

パイプラインの実行に関する情報 (アーティファクト、パイプラインの実行 ID、パイプラインの名前、バージョン、ステータスなど) を取得するために必要です。

リソース:

パイプライン

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

GetPipelineState

アクション:codepipeline:GetPipelineState

パイプラインの状態に関する情報 (ステージ、アクションなど) を取得するために必要です。

リソース:

パイプライン

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

GetThirdPartyJobDetails

アクション:codepipeline:GetThirdPartyJobDetails

サードパーティーアクションのジョブの詳細をリクエストするために必要です。パートナーアクションにのみ使用されます。

リソース: ポリシーの Resource 要素ではワイルドカード (*) のみがサポートされます。

ListActionTypes

アクション:codepipeline:ListActionTypes

アカウントに関連付けられたすべての CodePipeline のアクションタイプの概要を生成するために必要です。

リソース:

アクションの種類

arn:aws:codepipeline:*region*:*account*:actiontype:*owner*/*category*/*provider*/*versio*

ListPipelineExecutions

アクション:codepipeline:ListPipelineExecutions

パイプラインの最新の実行の概要を生成するために必要です。

リソース:

パイプライン

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

ListPipelines

アクション:codepipeline:ListPipelines

アカウントに関連付けられたすべてのパイプラインの概要を生成するために必要です。

リソース:

ワイルドカードを使用したパイプライン ARN (パイプライン名レベルのリソースレベルの許可はサポートされていません)

arn:aws:codepipeline:*region*:*account*:*

ListTagsForResource

アクション:codepipeline:ListTagsForResource

指定したリソースのタグを一覧表示するために必要です。

リソース:

アクションの種類

arn:aws:codepipeline:*region*:*account*:actiontype:*owner*/*category*/*provider*/*versio*

パイプライン

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

ウェブフック

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

ListWebhooks

アクション: codepipeline:ListWebhooks

そのリージョンのアカウントの全ウェブフックの一覧表示に必要です。

リソース:

ウェブフック

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

PollForJobs

アクション:codepipeline:PollForJobs

CodePipeline を実行するジョブに関する情報を取得するために必要です。

リソース:

アクションの種類

arn:aws:codepipeline:*region*:*account*:actiontype:*owner*/*category*/*provider*/*versio*

PollForThirdPartyJobs

アクション:codepipeline:PollForThirdPartyJobs

ジョブワーカーが影響するサードパーティージョブが存在するかどうかを判断するために必要です。パートナーアクションにのみ使用されます。

リソース: ポリシーの Resource 要素ではワイルドカード (*) のみがサポートされます。

PutActionRevision

アクション:codepipeline:PutActionRevision

新しいリビジョンに関する CodePipeline の情報をソースにレポートするために必要です。

リソース:

アクション

arn:aws:codepipeline:*region*:*account*:*pipeline-name*/*stage-name*/*action-name*

PutApprovalResult

アクション:codepipeline:PutApprovalResult

CodePipeline に対する手動の承認リクエストの応答をレポートするために必要です。有効な応答は Approved および Rejectedです。

リソース:

アクション

arn:aws:codepipeline:*region*:*account*:*pipeline-name*/*stage-name*/*action-name*

> **Note**
>
> この API コールはリソースレベルのアクセス権限をサポートします。ただし、IAM コンソールまたは Policy Generator を使用して、リソース ARN を指定するポリシーを "codepipeline:PutApprovalResult" で作成すると、エラーが発生する場合があります。エラーが発生した場合は IAM コンソールの [JSON] タブまたは CLI を使用してポリシーを作成することができます。

PutJobFailureResult

アクション:codepipeline:PutJobFailureResult

ジョブワーカーによってパイプラインに返されたジョブの失敗をレポートするために必要です。カスタムアクションにのみ使用されます。

リソース: ポリシーの Resource 要素ではワイルドカード (*) のみがサポートされます。

PutJobSuccessResult

アクション:codepipeline:PutJobSuccessResult

ジョブワーカーによってパイプラインに返されたジョブの成功をレポートするために必要です。カスタムアクションにのみ使用されます。

リソース: ポリシーの Resource 要素ではワイルドカード (*) のみがサポートされます。

PutThirdPartyJobFailureResult

アクション:codepipeline:PutThirdPartyJobFailureResult

ジョブワーカーによってパイプラインに返されるサードパーティージョブの失敗をレポートするために必要です。パートナーアクションにのみ使用されます。

リソース: ポリシーの Resource 要素ではワイルドカード (*) のみがサポートされます。

PutThirdPartyJobSuccessResult

アクション:codepipeline:PutThirdPartyJobSuccessResult

ジョブワーカーによってパイプラインに返されるサードパーティージョブの成功をレポートするために必要です。パートナーアクションにのみ使用されます。

リソース: ポリシーの Resource 要素ではワイルドカード (*) のみがサポートされます。

PutWebhook

アクション: codepipeline:PutWebhook

ウェブフックを作成するために必要です。

リソース:

ウェブフック

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

RegisterWebhookWithThirdParty

アクション: codepipeline:RegisterWebhookWithThirdParty

リソース:

ウェブフックの作成後、ウェブフック URL を生成するために、サポートされるサードパーティーを設定することが必要です。

ウェブフック

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

RetryStageExecution

アクション:codepipeline:RetryStageExecution

ステージで最後に失敗したアクションを再試行することでパイプラインの実行を再開するために必要です。

リソース:

パイプライン

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

StartPipelineExecution

アクション:codepipeline:StartPipelineExecution

指定されたパイプラインを開始するため (具体的には、パイプラインの一部として指定されたソース場所への最新のコミットの処理を開始するため) に必要です。

リソース:

パイプライン

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

TagResource

アクション:codepipeline:TagResource

指定されたリソースにタグを付けるために必要です。

リソース:

アクションの種類

arn:aws:codepipeline:*region*:*account*:actiontype:*owner*/*category*/*provider*/*versio*

パイプライン

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

ウェブフック

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

UntagResource

アクション:codepipeline:UntagResource

指定されたリソースのタグを削除するために必要です。

リソース:

アクションの種類

arn:aws:codepipeline:*region*:*account*:actiontype:*owner*/*category*/*provider*/*versio*

パイプライン

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

ウェブフック

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

UpdatePipeline

アクション:codepipeline:UpdatePipeline

その構造を編集または変更して、指定のパイプラインを更新するために必要です。

リソース:

パイプライン

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

CodePipeline サービスロールを管理する

サービスロールには、パイプラインで使用される AWS リソースへのアクセスを制御する 1 つ以上のポリシーが設定されています。このロールにさらにポリシーをアタッチしたり、ロールにアタッチされたポリシーを編集したり、 で他のサービスロールのポリシーを設定したりできます AWS。また、パイプラインへクロスアカウントアクセスを設定する際に、ポリシーをロールにアタッチすることもできます。

Important

ポリシーステートメントを変更するか、他のポリシーをロールにアタッチすると、パイプラインの動作が停止することがあります。 のサービスロールは、必ず影響を理解した上で変更するようにしてください。パイプラインは、必ずサービスロールに変更を加えてからテストします。

Note

コンソールでは、2018 年 9 月より前に作成されたサービスロールは、oneClick_AWS-CodePipeline-Service_*ID-Number* という名前で作成されています。
2018 年 9 月以降に作成されたサービスロールは、サービスロール名の形式 AWSCodePipelineServiceRole-*Region*-*Pipeline_Name* を使用します。例え

ば、MyFirstPipeline という名前のパイプラインが eu-west-2 にある場合、コンソールはロールとポリシー AWSCodePipelineServiceRole-eu-west-2-MyFirstPipeline の名前を付けます。

CodePipeline のサービスロールポリシー

CodePipeline のサービスロールポリシーステートメントには、パイプラインを管理するための最小限必要なアクセス許可が含まれています。サービスロールのステートメントを編集して、使用していないリソースへのアクセスを削除または追加します。CodePipeline が各アクションに使用する最小限必要なアクセス許可については、適切なアクションリファレンスを参照してください。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowS3BucketAccess",
      "Effect": "Allow",
      "Action": [
        "s3:GetBucketVersioning",
        "s3:GetBucketAcl",
        "s3:GetBucketLocation"
      ],
      "Resource": [
        "arn:aws:s3:::[[pipeArtifactBucketNames]]"
      ],
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "{{accountId}}"
        }
      }
    },
    {
      "Sid": "AllowS3ObjectAccess",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:PutObjectAcl",
        "s3:GetObject",
```

```
        "s3:GetObjectVersion",
        "s3:PutObjectTagging",
        "s3:GetObjectTagging",
        "s3:GetObjectVersionTagging"
      ],
      "Resource": [
        "arn:aws:s3:::[[pipeArtifactBucketNames]]/*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "{{accountId}}"
        }
      }
    }
  ]
}
```

> **Note**
>
> ポリシーでは、ソースバケット内の S3 オブジェクトにタグがある場合、次のアクセス許可が必要です。
>
> ```
> s3:PutObjectTagging
> s3:GetObjectTagging
> s3:GetObjectVersionTagging
> ```

CodePipeline サービスロールからアクセス許可を削除する

サービスロールのステートメントを編集して、使用していないリソースへのアクセスを削除します。例えば、いずれのパイプラインにも Elastic Beanstalk が含まれていない場合は、ポリシーステートメントを編集して Elastic Beanstalk リソースへのアクセスを許可するセクションを削除できます。

同様に、いずれのパイプラインにも CodeDeploy が含まれていない場合は、ポリシーステートメントを編集して CodeDeploy リソースへのアクセスを許可するセクションを削除できます。

```
    {
    "Action": [
        "codedeploy:CreateDeployment",
        "codedeploy:GetApplicationRevision",
```

```
            "codedeploy:GetDeployment",
            "codedeploy:GetDeploymentConfig",
            "codedeploy:RegisterApplicationRevision"
        ],
        "Resource": "*",
        "Effect": "Allow"
},
```

CodePipeline サービスロールにアクセス許可を追加する

サービスロールのポリシーステートメントは、パイプラインで使用する前に、デフォルトのサービスロールのポリシーステートメントに含まれていない AWS のサービス のアクセス許可で更新する必要があります。

これは、パイプラインに使用するサービスロールが、 のサポートが に追加される前に作成された場合に特に重要です AWS のサービス。

以下の表は、他の AWS のサービスにサポートが追加された場合の例を示しています。

AWS のサービス	CodePipeline のサポート日付
CodePipeline の呼び出しアクションのサポートが追加されました。「CodePipeline 呼び出しアクションのサービスロールポリシーのアクセス許可」を参照してください。	2025 年 3 月 14 日
EC2 アクションのサポートが追加されました。「EC2 デプロイアクションのサービスロールポリシーのアクセス許可」を参照してください。	2025 年 2 月 21 日
EKS アクションのサポートが追加されました。「サービスロールのポリシーのアクセス許可」を参照してください。	2025 年 2 月 20 日
Amazon Elastic Container Registry の ECRBuildAndPublish アクションのサポートが追加されました。「サービスロールのアクセス許可: ECRBuildAndPublish アクション」を参照してください。	2024 年 11 月 22 日

AWS のサービス	CodePipeline のサポート日付
Amazon Inspector の InspectorScan アクションのサポートが追加されました。「サービスロールのアクセス許可: InspectorScan アクション」を参照してください。	2024 年 11 月 22 日
コマンドアクションのサポートが追加されました。「サービスロールのアクセス許可: コマンドアクション」を参照してください。	2024 年 10 月 3 日
CloudFormation アクションのサポートが追加されました。「サービスロールのアクセス許可: CloudFormationStackSet アクション」および「サービスロールのアクセス許可: CloudFormationStackInstances アクション」を参照してください。	2020 年 12 月 30 日
CodeCommit フルクローン出力アーティファクト形式のサポートが追加されました。「サービスロールのアクセス許可: CodeCommit アクション」を参照してください。	2020 年 11 月 11 日
AWS CodeBuild バッチビルドアクションのサポートが追加されました。「サービスロールのアクセス許可: CodeCommit アクション」を参照してください。	2020 年 7 月 30 日
AWS AppConfig アクションのサポートが追加されました。「サービスロールのアクセス許可: AppConfig アクション」を参照してください。	2020 年 6 月 22 日
AWS Step Functions アクションのサポートが追加されました。「サービスロールのアクセス許可: StepFunctions アクション」を参照してください。	2020 年 5 月 27 日

AWS のサービス	CodePipeline のサポート日付
AWS CodeStar Connections アクションのサポートが追加されました。「サービスロールのアクセス許可: AWS CodeConnections アクション」を参照してください。	2019 年 12 月 18 日
S3 デプロイアクションのサポートが追加されました。「サービスロールのアクセス許可: S3 デプロイアクション」を参照してください。	2019 年 1 月 16 日
CodeDeployToECS アクションのサポートが追加されました。「サービスロールのアクセス許可: CodeDeployToECS アクション」を参照してください。	2018 年 11 月 27 日
Amazon ECR アクションのサポートが追加されました。「サービスロールのアクセス許可: Amazon ECR アクション」を参照してください。	2018 年 11 月 27 日
Service Catalog アクションのサポートが追加されました。「サービスロールのアクセス許可: Service Catalog アクション」を参照してください。	2018 年 10 月 16 日
AWS Device Farm アクションのサポートが追加されました。「サービスロールのアクセス許可: AWS Device Farm アクション」を参照してください。	2018 年 7 月 19 日
Amazon ECS アクションのサポートが追加されました。「サービスロールのアクセス許可: Amazon ECS 標準アクション」を参照してください。	2017 年 12 月 12 日 / 2017 年 7 月 21 日から開始されたタグ付け承認のオプトインのための更新

AWS のサービス	CodePipeline のサポート日付
CodeCommit アクションのサポートが追加されました。「サービスロールのアクセス許可: CodeCommit アクション」を参照してください。	2016 年 4 月 18 日
AWS OpsWorks アクションのサポートが追加されました。「サービスロールのアクセス許可: AWS OpsWorks アクション」を参照してください。	2016 年 6 月 2 日
CloudFormation アクションのサポートが追加されました。「サービスロールのアクセス許可: CloudFormation アクション」を参照してください。	2016 年 11 月 3 日
AWS CodeBuild アクションのサポートが追加されました。「サービスロールのアクセス許可: CodeBuild アクション」を参照してください。	2016 年 12 月 1 日
Elastic Beanstalk アクションのサポートが追加されました。「サービスロールのアクセス許可: ElasticBeanstalk のデプロイアクション」を参照してください。	初回サービス起動
CodeDeploy アクションのサポートが追加されました。「サービスロールのアクセス許可: AWS CodeDeploy アクション」を参照してください。	初回サービス起動
S3 ソースアクションのサポートが追加されました。「サービスロールのアクセス許可: S3 ソースアクション」を参照してください。	初回サービスの起動

サポートされているサービスのアクセス許可を追加するには、次の手順に従います。

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/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、ロールのリストから `AWS-CodePipeline-Service` ロール を選択します。
3. [アクセス権限] タブの [インラインポリシー] で、サービスロールポリシーの列の [ポリシーの編集] を選択します。
4. [Policy Document] ボックスに必要なアクセス許可を追加します。

> **Note**
>
> IAM ポリシーを作成するとき、最小限の特権を認めるという標準的なセキュリティアドバイスに従いましょう。そうすれば、タスクを実行するというリクエストのアクセス許可のみを認めることができます。一部の API コールはリソースベースのアクセス許可をサポートしており、アクセスを制限できます。たとえば、この場合、`DescribeTasks` および `ListTasks` を呼び出す際のアクセス許可を制限するために、ワイルドカード文字 (*) をリソース ARN またはワイルドカード文字 (*) を含むリソース ARN に置き換えることができます。最小権限アクセスを付与するポリシーの作成の詳細については、https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#grant-least-privilege を参照してください。

5. [Review policy] (ポリシーの確認) を選択して、ポリシーにエラーがないことを確認します。ポリシーにエラーがなければ、ポリシーの適用 を選択します。

CodePipeline でのロギングとモニタリング

のログ記録機能を使用して AWS 、ユーザーがアカウントで実行したアクションと使用されたリソースを判断できます。ログファイルは次の情報を表示します：

- アクションが実行された日時。
- アクションのソース IP アドレス。
- 不適切なアクセス権限が理由で失敗したアクション。

ロギング機能は次の AWS のサービスで使用できます。

- AWS CloudTrail は、 によって行われた、または に代わって行われた AWS API コールおよび関連イベントのログ記録に使用できます AWS アカウント。詳細については、「を使用した CodePipeline API コールのログ記録 AWS CloudTrail」を参照してください。
- Amazon CloudWatch Events を使用して、 AWS クラウド リソースと実行するアプリケーションをモニタリングできます AWS。定義したメトリクスに基づいて、 Amazon CloudWatch Events でアラートを作成できます。詳細については、「CodePipeline イベントのモニタリング」を参照してください。

のコンプライアンス検証 AWS CodePipeline

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、「コンプライアンスAWS のサービス 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。

アーティファクトを使用して、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。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。

の耐障害性 AWS CodePipeline

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、高度に冗長なネットワークで接続された、物理的に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ー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。

のインフラストラクチャセキュリティ AWS CodePipeline

マネージドサービスである AWS CodePipeline は、 AWS グローバルネットワークセキュリティで保護されています。 AWS セキュリティサービスと がインフラストラクチャ AWS を保護する方法については、AWS 「 クラウドセキュリティ」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のベストプラクティスを使用して環境を AWS 設計するには、「Security Pillar AWS Well-Architected Framework」の「Infrastructure Protection」を参照してください。

AWS 公開された API コールを使用して、ネットワーク経由で CodePipeline にアクセスします。クライアントは以下をサポートする必要があります。

- Transport Layer Security (TLS)。TLS 1.2 が必須で、TLS 1.3 をお勧めします。
- DHE (楕円ディフィー・ヘルマン鍵共有) や ECDHE (楕円曲線ディフィー・ヘルマン鍵共有) などの完全前方秘匿性 (PFS) による暗号スイート。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。

セキュリティのベストプラクティス

CodePipeline には、独自のセキュリティポリシーを開発および実装する際に考慮する必要のあるいくつかのセキュリティ機能が用意されています。以下のベストプラクティスは一般的なガイドラインであり、完全なセキュリティソリューションを説明するものではありません。これらのベストプラクティスはお客様の環境に適切ではないか、十分ではない場合があるため、これらは指示ではなく、有用な考慮事項と見なしてください。

パイプラインに接続するソースリポジトリには暗号化と認証を使用します。こちらがセキュリティで使用する CodePipeline のベストプラクティスです。

- トークンやパスワードのようなシークレットを含むパイプラインやアクション設定を作成する場合は、そのシークレットをアクション設定や、パイプラインレベルまたは CloudFormation 設定で定義された変数のデフォルト値に直接入力しないでください。シークレットの情報がログに表示される可能性があるためです。AWS Secrets Manager を使用してデータベースパスワードまたはサードパーティー API キーを追跡する で説明しているように、Secrets Manager を使用してシークレットを設定して保存し、パイプラインとアクション設定には、シークレットの参照を使用します。

- S3 ソースバケットを使用するパイプラインを作成する場合は、 AWS KMS keysを管理して CodePipeline 用の Amazon S3 に保存されたアーティファクトのサーバー側の暗号化を設定します。CodePipeline 用に Amazon S3 に保存したアーティファクトのサーバー側の暗号化を設定するに説明があります。
- Jenkins アクションプロバイダを使用しており、Jenkins ビルドプロバイダをパイプラインのビルドまたはテスト作業に使用する際は、ECS インスタンスに Jenkins をインストールし、別の EC2 インスタンスプロファイルを構成してください。インスタンスプロファイルが、Amazon S3 からファイルを取得するなど、プロジェクトのタスクを実行するために必要な AWS アクセス許可のみを Jenkins に付与していることを確認します。Jenkins インスタンスプロファイルのロールを作成する方法については、「Jenkins 統合に使用する IAM ロールを作成する」のステップを参照してください。

CodePipeline パイプライン構造リファレンス

CodePipeline を使用して、アプリケーションのソースコードを構築、テスト、デプロイするタスクの実行手順が自動化された CI/CD パイプラインを構築できます。このリファレンスセクションでは、パイプラインの JSON 構造とパラメータについて詳しく説明します。パイプラインの使用方法を説明する概念の大まかなリストについては、「CodePipeline の概念 」を参照してください。

- パイプラインを作成するときは、ソースコードを含むとともに、ソースコードの変更をコミットしたときにパイプラインを開始する S3 バケット、CodeCommit リポジトリ、Bitbucket リポジトリ、GitHub リポジトリなどの利用可能なプロバイダーとソースアクションを選択します。このリファレンスセクションでは、パイプラインで使用可能なソースに関するリファレンス情報を提供します。ソースアクションの操作の詳細については、「CodePipeline でパイプラインを編集する」を参照してください。
- パイプラインの実行時に自動的に含めるテスト、ビルド、デプロイのアクションとプロバイダーを選択できます。このリファレンスセクションでは、使用可能なアクションと、それらがパイプライン JSON にどのように適合するかに関するリファレンス情報を提供します。
- 完成したパイプラインは、ソースステージと、アプリケーションをデプロイしてテストするアクションを設定する追加のステージで構成されます。アプリケーションをデプロイする DevOps パイプラインの概念的な例については、「DevOps パイプラインの例」を参照してください。

デフォルトでは、 で正常に作成したパイプライン AWS CodePipeline の構造は有効です。ただし、JSON ファイルを手動で作成または編集してパイプラインを作成したり、 からパイプラインを更新したりすると AWS CLI、無効な構造が誤って作成される可能性があります。次のリファレンスは、パイプライン構造の要件や、問題のトラブルシューティング方法を理解するのに役立ちます。すべてのパイプラインに適用される AWS CodePipeline のクォータ の制約を参照してください。

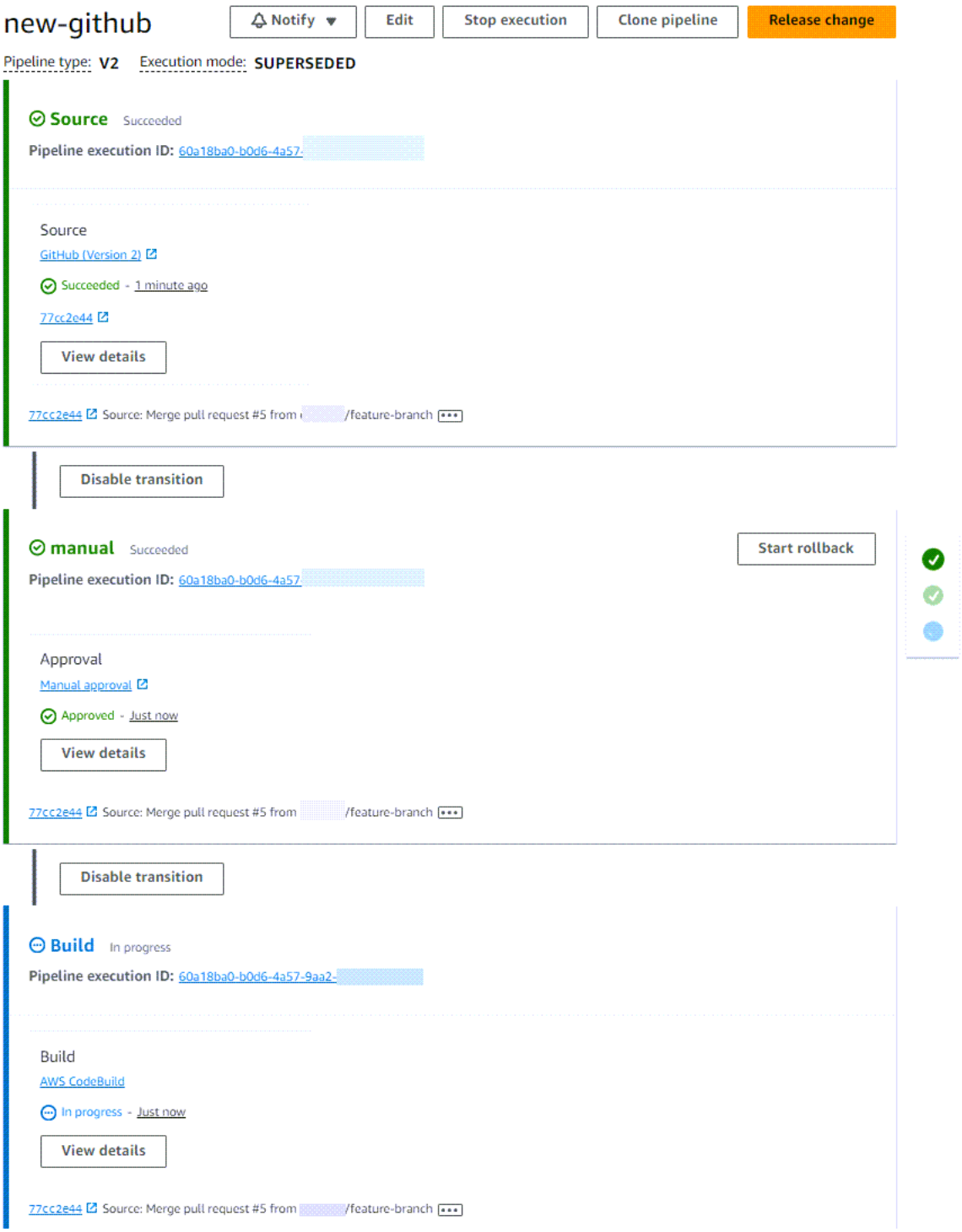
以下のセクションでは、高レベルのパラメータと、パイプライン構造におけるこれらのパラメータの位置について説明します。パイプライン構造の要件は、以下のパイプラインコンポーネントタイプごとに各セクションで詳しく説明しています。

- パイプライン宣言のフィールドリファレンス
- ステージ宣言のフィールドリファレンス
- アクションの宣言のフィールドリファレンス
- アクションタイプ別のCodePipeline の有効なアクションプロバイダーのリスト

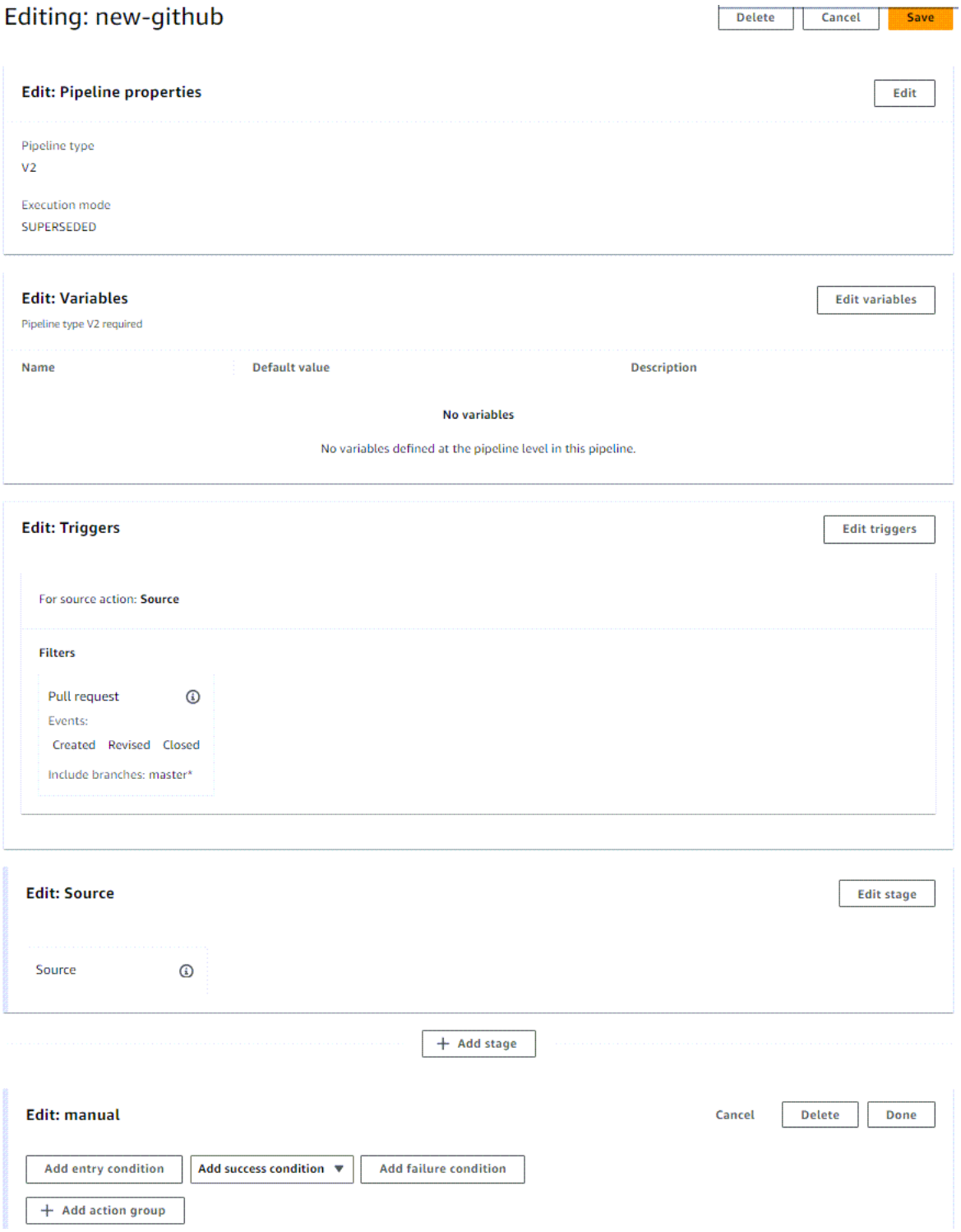
- PollForSourceChanges パラメータの有効な設定のリファレンス
- アクションタイプ別の有効な入力/出力アーティファクトの数のリファレンス
- プロバイダータイプ別の有効な設定パラメータへのリンクのリスト

詳細については、「CodePipeline API ガイド」の「PipelineDeclaration」オブジェクトを参照してください。

次のパイプラインコンソールビューの例は、new-github という名前のパイプライン、ステージ (Source、manual、Build)、GitHub (GitHub アプリ経由) からのアクション、手動承認、CodeBuild アクションプロバイダーを示しています。



パイプライン編集モードをコンソール図で表示すると、次の例に示すように、ソースの上書き、トリガー、アクションを編集できます。



トピック

- パイプライン宣言
- ステージ宣言
- アクションの宣言

- CodePipeline の有効なアクションプロバイダー
- PollForSourceChanges パラメータの有効な設定
- アクションタイプ別の有効な入力/出力アーティファクトの数
- プロバイダータイプ別の有効な設定パラメータ

パイプライン宣言

パイプラインレベルおよびメタデータレベルのパイプラインには、以下のパラメータと構文を含む基本構造があります。パイプラインパラメータは、パイプラインで実行するアクションとステージの構造を表します。

詳細については、「CodePipeline API ガイド」の「PipelineDeclaration」オブジェクトを参照してください。

次の例は、V2 タイプのパイプラインについて、パイプラインレベルおよびメタデータレベルのパイプライン構造を JSON と YAML の両方で示しています。

YAML

```
pipeline:
  name: MyPipeline
  roleArn: >-
    arn:aws:iam::ACCOUNT_ID:role/service-role/AWSCodePipelineServiceRole-us-west-2-
MyPipeline
  artifactStore:
    type: S3
    location: amzn-s3-demo-bucket
  stages:
    ...
  version: 6
  executionMode: SUPERSEDED
  pipelineType: V2
  variables:
  - name: MyVariable
    defaultValue: '1'
  triggers:
  - providerType: CodeStarSourceConnection
    gitConfiguration:
      sourceActionName: Source
      push:
```

```
      - branches:
          includes:
          - main
          excludes:
          - feature-branch
        pullRequest:
        - events:
          - CLOSED
          branches:
            includes:
            - main*
metadata:
  pipelineArn: 'arn:aws:codepipeline:us-west-2:ACCOUNT_ID:MyPipeline'
  created: '2019-12-12T06:49:02.733000+00:00'
  updated: '2020-09-10T06:34:07.447000+00:00'
  pollingDisabledAt: '2020-09-10T06:34:07.447000+00:00'
```

JSON

```
{
    "pipeline": {
        "name": "MyPipeline",
        "roleArn": "arn:aws:iam::ACCOUNT_ID:role/service-role/
AWSCodePipelineServiceRole-us-west-2-MyPipeline",
        "artifactStore": {
            "type": "S3",
            "location": "amzn-s3-demo-bucket"
        },
        "stages": {
            ...
    },
        "version": 6,
        "executionMode": "SUPERSEDED",
                "pipelineType": "V2",
        "variables": [
            {
                "name": "MyVariable",
                "defaultValue": "1"
            }
        ],
        "triggers": [
            {
                "providerType": "CodeStarSourceConnection",
```

```
                "gitConfiguration": {
                    "sourceActionName": "Source",
                    "push": [
                        {
                            "branches": {
                                "includes": [
                                    "main"
                                ],
                                "excludes": [
                                    "feature-branch"
                                ]
                            }
                        }
                    ],
                    "pullRequest": [
                        {
                            "events": [
                                "CLOSED"
                            ],
                            "branches": {
                                "includes": [
                                    "main*"
                                ]
                            }
                        }
                    ]
                }
            }
        ]
    },
    "metadata": {
        "pipelineArn": "arn:aws:codepipeline:us-west-2:ACCOUNT_ID:MyPipeline",
        "created": "2019-12-12T06:49:02.733000+00:00",
        "updated": "2020-09-10T06:34:07.447000+00:00",
        "pollingDisabledAt": "2020-09-10T06:34:07.447000+00:00"
    }
}
```

name

パイプラインの名前。パイプラインを編集または更新する場合、パイプライン名は変更できません。

> **Note**
>
> 既存のパイプラインの名前を変更するには、CLI `get-pipeline` コマンドを使用して、パイプライン構造を含む JSON ファイルを作成します。次に、CLI `create-pipeline` コマンドを使用してその構造を持つ新しいパイプラインを作成し、新しい名前を付けます。

roleArn

CodePipeline サービスロールの IAM ARN (arn:aws:iam::80398EXAMPLE:role/CodePipeline_Service_Role など)。

コンソールを使用して、JSON 構造ではなくパイプラインサービスロール ARN を表示するには、コンソールでパイプラインを選択し、[設定] を選択します。[全般] タブに、[サービスロール ARN] フィールドが表示されます。

artifactStore または artifactStores

`artifactStore` フィールドには、同じ AWS リージョン内のすべてのアクションを持つパイプラインのアーティファクトバケットタイプと場所が含まれます。パイプラインとは異なるリージョンにアクションを追加すると、`artifactStores`マッピングを使用して、アクションが実行される各 AWS リージョンのアーティファクトバケットが一覧表示されます。パイプラインを作成または編集する場合は、パイプラインリージョンにアーティファクトバケットが必要であり、アクションを実行する予定のリージョンごとに 1 つのアーティファクトバケットが必要です。

> **Note**
>
> パイプライン構造では、パイプラインに `artifactStore` または `artifactStores` のどちらかを含める必要がありますが、両方を使用することはできません。パイプラインでクロスリージョンアクションを作成する場合は、`artifactStores` を使用する必要があります。

以下の例では、`artifactStores` パラメータを使用するクロスリージョンアクションを含むパイプラインの基本構造を示しています。

```
    "pipeline": {
        "name": "YourPipelineName",
        "roleArn": "CodePipeline_Service_Role",
```

```
        "artifactStores": {
            "us-east-1": {
                "type": "S3",
                "location": "S3 artifact bucket name, such as amzn-s3-demo-bucket"
            },
            "us-west-2": {
                "type": "S3",
                "location": "S3 artifact bucket name, such as amzn-s3-demo-bucket"
            }
        },
        "stages": [
            {

...
```

type

Amazon S3 として指定した、アーティファクトバケットの場所タイプ。

location

コンソールを使用してパイプラインを初めて作成するときに自動的に生成される Amazon S3 バケットの名前 (codepipeline-us-east-2-1234567890 など)、またはこの目的のためにプロビジョニングする任意の Amazon S3 バケットの名前。

stages

このパラメータには、パイプラインの各ステージの名前が含まれます。パイプライン構造のステージレベルのパラメータと構文の詳細については、「CodePipeline API ガイド」の「StageDeclaration」オブジェクトを参照してください。

ステージのパイプライン構造には、以下の要件があります。

- パイプラインには、少なくとも 2 つのステージが含まれている必要がある
- パイプラインの最初のステージには、少なくとも 1 つのソースアクションが含まれている必要がある ソースアクションのみを含めることができる
- ソースアクションを含むことができるのは、パイプラインの最初のステージのみである
- 各パイプラインの少なくとも 1 つのステージに、ソースアクション以外のアクションが含まれている必要がある
- パイプライン内のすべてのステージ名は一意である必要がある

- ステージ名は、CodePipeline コンソール内で編集することはできません。を使用してステージ名を編集し AWS CLI、ステージに 1 つ以上のシークレットパラメータ (OAuth トークンなど) を含むアクションが含まれている場合、それらのシークレットパラメータの値は保持されません。パラメータ値 (AWS CLIよって返される JSON で、4 つのアスタリスクでマスクされている) は手動で入力し、JSON 構造に含める必要があります。

> **Important**
>
> 30 日以上非アクティブなパイプラインでは、パイプラインのポーリングが無効になります。詳細については、パイプイライン構造リファレンスの「pollingDisabledAt」を参照してください。パイプラインをポーリングからイベントベースの変更検出に移行する手順については、「変更検出方法」を参照してください。

version

パイプラインのバージョン番号は自動的に生成され、パイプラインを更新するたびに更新されます。

executionMode

パイプライン実行モードを設定すると、キュー、優先、並列モードでの実行など、連続した実行のパイプライン動作を指定できます。詳細については、「パイプライン実行モードを設定または変更する」を参照してください。

> **Important**
>
> PARALLEL モードのパイプラインでは、ステージのロールバックは使用できません。同様に、ロールバック結果タイプの障害条件を PARALLEL モードのパイプラインに追加することはできません。

pipelineType

パイプラインタイプは、V2 タイプのパイプラインなど、パイプラインで利用可能な構造と機能を指定します。詳細については、「パイプラインのタイプ」を参照してください。

variables

パイプラインレベルの変数は、パイプラインの作成時に定義され、パイプラインの実行時に解決されます。詳細については、「変数リファレンス」を参照してください。パイプラインの実行時に渡されるパイプラインレベルの変数のチュートリアルについては、「チュートリアル: パイプラインレベルの変数を使用する」を参照してください。

triggers

トリガーを使用すると、特定のブランチやプルリクエストの変更を検出したときなど、特定のイベントタイプやフィルタリングされたイベントタイプに応じて開始するようにパイプラインを設定できます。トリガーは、GitHub、Bitbucket、GitLab など、CodePipeline の CodeStarSourceConnection アクションを実行する接続を使用するソースアクションに対して設定できます。接続を使用するソースアクションの詳細については、「を使用してパイプラインにサードパーティーのソースプロバイダーを追加する AWS CodeConnections」を参照してください。

詳細とより具体的な例については、「トリガーとフィルタリングを使用してパイプラインを自動的に開始する」を参照してください。

フィルタリングでは、「構文での glob パターンの使用」で詳述しているように、正規表現パターンを glob 形式でサポートしています。

> **Note**
>
> CodeCommit および S3 ソースアクションには、設定済みの変更検出リソース (EventBridge ルール)、またはオプションを使用してソースの変更をリポジトリにポーリングする必要があります。Bitbucket、GitHub、または GitHub Enterprise Server のソースアクションを持つパイプラインの場合、ウェブフックを設定したり、デフォルトでポーリングを行う必要はありません。接続アクションは、変更検出を管理します。

> **Important**
>
> 30 日以上非アクティブなパイプラインでは、パイプラインのポーリングが無効になります。詳細については、パイプイライン構造リファレンスの「pollingDisabledAt」を参照してください。パイプラインをポーリングからイベントベースの変更検出に移行する手順については、「変更検出方法」を参照してください。

gitConfiguration フィールド

イベントタイプ、ブランチ、ファイルパス、タグ、プルリクエストイベントでフィルタリングするためのパラメータなど、トリガーの Git 設定。

JSON 構造内のフィールドの定義は以下のとおりです。

- sourceActionName: Git 設定のパイプラインソースアクションの名前。
- push: フィルタリングを含むプッシュイベント。これらのイベントは、異なるプッシュフィルター間では OR 演算を使用し、フィルター内では AND 演算を使用します。
  - branches: フィルタリングするブランチ。ブランチは、[含める] と [除外する] の間で AND 演算を使用します。
    - includes: 含めるブランチをフィルタリングするパターン。[含める] では OR 演算を使用します。
    - excludes: 除外するブランチをフィルタリングするパターン。[除外する] では OR 演算を使用します。
  - filePaths: フィルタリングするファイルパス名。
    - includes: 含めるファイルパスをフィルタリングするパターン。[含める] では OR 演算を使用します。
    - excludes: 除外するファイルパスをフィルタリングするパターン。[除外する] では OR 演算を使用します。
  - tags: フィルタリングするタグ名。
    - includes: 含めるタグをフィルタリングするパターン。[含める] では OR 演算を使用します。
    - excludes: 除外するタグをフィルタリングするパターン。[除外する] では OR 演算を使用します。
- pullRequest: プルリクエストイベントとプルリクエストフィルターのフィルタリングを含むプルリクエストイベント。
  - events: オープン、更新、またはクローズしたプルリクエストイベントを指定どおりにフィルタリングします。
  - branches: フィルタリングするブランチ。ブランチは、[含める] と [除外する] の間で AND 演算を使用します。
    - includes: 含めるブランチをフィルタリングするパターン。[含める] では OR 演算を使用します。

- excludes: 除外するブランチをフィルタリングするパターン。[除外する] では OR 演算を使用します。
- filePaths: フィルタリングするファイルパス名。
  - includes: 含めるファイルパスをフィルタリングするパターン。[含める] では OR 演算を使用します。
  - excludes: 除外するファイルパスをフィルタリングするパターン。[除外する] では OR 演算を使用します。

プッシュおよびプルリクエストイベントタイプのトリガー設定の例を次に示します。

```
"triggers": [
            {
                "provider": "Connection",
                "gitConfiguration": {
                    "sourceActionName": "ApplicationSource",
                    "push": [
                        {
                            "filePaths": {
                                "includes": [
                                    "projectA/**",
                                    "common/**/*.js"
                                ],
                                "excludes": [
                                    "**/README.md",
                                    "**/LICENSE",
                                    "**/CONTRIBUTING.md"
                                ]
                            },
                            "branches": {
                                "includes": [
                                    "feature/**",
                                    "release/**"
                                ],
                                "excludes": [
                                    "mainline"
                                ]
                            },
                            "tags": {
                                "includes": [
                                    "release-v0", "release-v1"
                                ],
```

```
                            "excludes": [
                                "release-v2"
                            ]
                        }
                    }
                ],
                "pullRequest": [
                    {
                        "events": [
                            "CLOSED"
                        ],
                        "branches": {
                            "includes": [
                                "feature/**",
                                "release/**"
                            ],
                            "excludes": [
                                "mainline"
                            ]
                        },
                        "filePaths": {
                            "includes": [
                                "projectA/**",
                                "common/**/*.js"
                            ],
                            "excludes": [
                                "**/README.md",
                                "**/LICENSE",
                                "**/CONTRIBUTING.md"
                            ]
                        }
                    }
                ]
            }
        }
    ],
```

包含および除外のイベントタイプ **push** フィールド

プッシュイベントタイプの Git 設定フィールドレベルの包含および除外動作を次のリストに示します。

```
push (OR operation is used between push and pullRequest or multiples)
```

```
    filePaths (AND operation is used between filePaths, branches, and tags)
        includes (AND operation is used between includes and excludes)
            **/FILE.md, **/FILE2 (OR operation is used between file path names)
        excludes (AND operation is used between includes and excludes)
            **/FILE.md, **/FILE2 (OR operation is used between file path names)
    branches (AND operation is used between filePaths, branches, and tags)
        includes (AND operation is used between includes and excludes)
            BRANCH/**", "BRANCH2/** (OR operation is used between branch names)
        excludes (AND operation is used between includes and excludes)
            BRANCH/**", "BRANCH2/** (OR operation is used between branch names)
    tags (AND operation is used between filePaths, branches, and tags)
         includes (AND operation is used between includes and excludes)
            TAG/**", "TAG2/** (OR operation is used between tag names)
         excludes (AND operation is used between includes and excludes)
            TAG/**", "TAG2/** (OR operation is used between tag names)
```

包含および除外のイベントタイプ **pull request** フィールド

プルリクエストイベントタイプの Git 設定フィールドレベルの包含および除外動作を次のリストに示します。

```
pullRequest (OR operation is used between push and pullRequest or multiples)
    events (AND operation is used between events, filePaths, and branches). Includes/
excludes are N/A for pull request events.
    filePaths (AND operation is used between events, filePaths, and branches)
        includes (AND operation is used between includes and excludes)
            **/FILE.md, **/FILE2 (OR operation is used between file path names)
        excludes (AND operation is used between includes and excludes)
            **/FILE.md, **/FILE2 (OR operation is used between file path names)
    branches (AND operation is used between events, filePaths, and branches)
        includes (AND operation is used between includes and excludes)
            BRANCH/**", "BRANCH2/** (OR operation is used between branch names)
        excludes (AND operation is used between includes and excludes)
            BRANCH/**", "BRANCH2/** (OR operation is used between branch names)
```

metadata

パイプラインメタデータフィールドはパイプライン構造とは異なり、編集することはできません。パイプラインを更新すると、updated メタデータフィールドの日付が自動的に変更されます。

pipelineArn

パイプラインの Amazon リソースネーム (ARN)。

コンソールを使用して、JSON 構造ではなくパイプライン ARN を表示するには、コンソールでパイプラインを選択し、[設定] を選択します。[全般] タブに、[パイプライン ARN] フィールドが表示されます。

created

パイプラインの作成日時。

updated

パイプラインの最終更新日時。

pollingDisabledAt

変更検出のポーリング用に設定されたパイプラインについて、ポーリングが無効になった日時。

30 日以上非アクティブなパイプラインでは、パイプラインのポーリングが無効になります。

- 30 日間実行しない場合、非アクティブなパイプラインのポーリングは無効になります。
- EventBridge、CodeStar Connections、またはウェブフックを使用するパイプラインは影響を受けません。
- アクティブなパイプラインは影響を受けません。

詳細については、「CodePipeline API ガイド」の PipelineMetadata オブジェクトの `pollingDisabledAt` パラメータを参照してください。パイプラインをポーリングからイベントベースの変更検出に移行する手順については、「変更検出方法」を参照してください。

ステージ宣言

ステージレベルのパイプラインには、以下のパラメータと構文を含む基本構造があります。詳細については、「CodePipeline API ガイド」の「StageDeclaration」オブジェクトを参照してください。

次の例は、ステージレベルのパイプライン構造を JSON と YAML の両方で示しています。この例には、`Source` および `Build` という 2 つのステージが含まれています。`onSuccess` の条件と `beforeEntry` の条件の 2 つも含まれています。

YAML

```
pipeline:
  name: MyPipeline
  roleArn: >-
    arn:aws:iam::ACCOUNT_ID:role/service-role/AWSCodePipelineServiceRole-us-west-2-
MyPipeline
  artifactStore:
    type: S3
    location: amzn-s3-demo-bucket
  stages:
    - name: Source
      actions:
        - name: Source
          ...
    - name: Build
      actions:
        - name: Build
          ...
      onSuccess:
        conditions:
        - result: ROLLBACK
          rules:
          - name: DeploymentWindowRule
         ...
      beforeEntry:
        conditions:
        - result: FAIL
          rules:
          - name: MyLambdaRule
         ...
  version: 6
metadata:
  pipelineArn: 'arn:aws:codepipeline:us-west-2:ACCOUNT_ID:MyPipeline'
  created: '2019-12-12T06:49:02.733000+00:00'
  updated: '2020-09-10T06:34:07.447000+00:00'
```

JSON

```
{
    "pipeline": {
        "name": "MyPipeline",
```

```
        "roleArn": "arn:aws:iam::ACCOUNT_ID:role/service-role/
AWSCodePipelineServiceRole-us-west-2-MyPipeline",
        "artifactStore": {
            "type": "S3",
            "location": "amzn-s3-demo-bucket"
        },
        "stages": [
            {
                "name": "Source",
                "actions": [
                    {
                        "name": "Source",
                        ...
                    }
                ]
            },
            {
                "name": "Build",
                "actions": [
                    {
                        "name": "Build",
                        ...
                    }
                ],
                "onSuccess": {
                    "conditions": [
                        {
                            "result": "ROLLBACK",
                            "rules": [
                                {
                                    "name": "DeploymentWindowRule",
                                    ...
                                }
                            ]
                        }
                    ]
                },
                "beforeEntry": {
                    "conditions": [
                        {
                            "result": "FAIL",
                            "rules": [
                                {
                                    "name": "MyLambdaRule",
```

```
                                ...
                            }
                        ]
                    }
                ]
            }
        }
    ],

        }
    ],
    "version": 6
},
"metadata": {
    "pipelineArn": "arn:aws:codepipeline:us-west-2:ACCOUNT_ID:MyPipeline",
    "created": "2019-12-12T06:49:02.733000+00:00",
    "updated": "2020-09-10T06:34:07.447000+00:00"
}
}
```

name

ステージの名前。

actions

アクションレベルのパイプラインには、以下のパラメータと構文を含む基本構造があります。パラメータと例を表示するには、「アクションの宣言」を参照してください。

conditions

条件には、CodePipeline のルールのリストで使用できる 1 つ以上のルールが含まれます。条件内のすべてのルールが成功すると、条件は満たされます。条件が満たされない場合に、指定した結果が適用されるように、条件を設定できます。

以下のタイプの条件を設定できます。

- beforeEntry
- onFailure
- onSuccess

詳細な説明と例については、ステージの条件を設定する を参照してください。

rules

各条件内には、一連の順序付けられたルールがあり、セットとしてまとめて評価されます。したがって、条件内の 1 つのルールが失敗すると、条件は失敗します。ルール条件は、パイプラインのランタイムに上書きできます。

利用可能なルールは、ルールリファレンスに記載しています。詳細については、「ルール構造リファレンス」でルール構造リファレンスを参照してください。

アクションの宣言

アクションレベルのパイプラインには、以下のパラメータと構文を含む基本構造があります。詳細については、「CodePipeline API ガイド」の「ActionDeclaration」オブジェクトを参照してください。

次の例は、アクションレベルのパイプライン構造を JSON と YAML の両方で示しています。

YAML

```
. . .

  stages:
    - name: Source
      actions:
        - name: Source
          actionTypeId:
            category: Source
            owner: AWS
            provider: S3
            version: '1'
          runOrder: 1
          configuration:
            PollForSourceChanges: 'false'
            S3Bucket: amzn-s3-demo-bucket
            S3ObjectKey: codedeploy_linux.zip
          outputArtifacts:
            - name: SourceArtifact
          inputArtifacts: []
          region: us-west-2
          namespace: SourceVariables
```

```
    - name: Build
      actions:
        - name: Build
          actionTypeId:
            category: Build
            owner: AWS
            provider: CodeBuild
            version: '1'
          runOrder: 1
          configuration:
            EnvironmentVariables: >-
              [{"name":"ETag","value":"#{SourceVariables.ETag}","type":"PLAINTEXT"}]
            ProjectName: my-project
          outputArtifacts:
            - name: BuildArtifact
          inputArtifacts:
            - name: SourceArtifact
          region: us-west-2
          namespace: BuildVariables
          runOrder: 1
          configuration:
            CustomData: >-
              Here are the exported variables from the build action: S3 ETAG:
              #{BuildVariables.ETag}
          outputArtifacts: []
          inputArtifacts: []
          region: us-west-2
```

JSON

```
. . .

        "stages": [
            {
                "name": "Source",
                "actions": [
                    {
                        "name": "Source",
                        "actionTypeId": {
                            "category": "Source",
                            "owner": "AWS",
                            "provider": "S3",
```

```
                            "version": "1"
                        },
                        "runOrder": 1,
                        "configuration": {
                            "PollForSourceChanges": "false",
                            "S3Bucket": "amzn-s3-demo-bucket",
                            "S3ObjectKey": "aws-codepipeline-s3-aws-
codedeploy_linux.zip"
                        },
                        "outputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "inputArtifacts": [],
                        "region": "us-west-2",
                        "namespace": "SourceVariables"
                    }
                ]
            },
            {
                "name": "Build",
                "actions": [
                    {
                        "name": "Build",
                        "actionTypeId": {
                            "category": "Build",
                            "owner": "AWS",
                            "provider": "CodeBuild",
                            "version": "1"
                        },
                        "runOrder": 1,
                        "configuration": {
                            "EnvironmentVariables": "[{\"name\":\"ETag\",\"value\":
\"#{SourceVariables.ETag}\",\"type\":\"PLAINTEXT\"}]",
                            "ProjectName": "my-build-project"
                        },
                        "outputArtifacts": [
                            {
                                "name": "BuildArtifact"
                            }
                        ],
                        "inputArtifacts": [
                            {
```

```
                            "name": "SourceArtifact"
                        }
                    ],
                    "region": "us-west-2",
                    "namespace": "BuildVariables"
                }
            ]

. . .
```

このプロバイダータイプに該当する configuration の例の詳細一覧については、「プロバイダータイプ別の有効な設定パラメータ」を参照してください。

アクション構造には、次の要件があります。

- ステージ内のすべてのアクション名は一意である必要がある
- 各パイプラインにはソースアクションが必要です。
- 接続を使用しないソースアクションでは、変更検出をオンに設定することも、オフに設定することもできます。「変更検出方法」を参照してください。
- これは、同じステージか、それ以降のステージかにかかわらず、すべてのアクションで当てはまりますが、入力アーティファクトは、出力アーティファクトを提供したアクションからの厳密なシーケンスにおける次のアクションである必要はありません。アクションは並行して、異なる出力アーティファクトバンドルを宣言することがあります。これらは、以下のアクションによって順番に消費されます。
- デプロイ場所として Amazon S3 バケットを使用するときは、オブジェクトキーも指定します。オブジェクトキーはファイル名 (オブジェクト) にするか、プレフィックス (フォルダパス) とファイル名の組み合わせにすることができます。変数を使用して、パイプラインで使用する場所の名前を指定できます。Amazon S3 のデプロイアクションでは、Amazon S3 オブジェクトキーでの以下の変数の使用がサポートされます。

  Amazon S3 での変数の使用

変数	コンソール入力の例	Output
datetime	js-application/{datetime}.zip	この形式の UTC タイムスタンプ: <YYYY>-<MM>-DD>_<HH>-<MM>-<SS>

変数	コンソール入力の例	Output
		例: js-application/2019-01-10_07-39-57.zip
uuid	js-application/{uuid}.zip	UUID は、他のすべての識別子と異なることが保証されるグローバル一意識別子です。この形式の UUID (すべての桁は 16 進形式): <8-digits>-<4-digits>-4-digits>-<4-digits>-<12-digits> 例: js-application/54a60075-b96a-4bf3-9013-db3a9EXAMPLE.zip

name

アクションの名前。

region

プロバイダーが であるアクションの場合 AWS のサービス、リソース AWS リージョン の 。

クロスリージョンアクションの場合は、[Region] フィールドを使用してアクションを作成する AWS リージョン を指定します。このアクション用に作成された AWS リソースは、 regionフィールドで指定されたのと同じリージョンで作成する必要があります。以下のアクションタイプのクロスリージョンアクションは作成できません。

- ソースアクション
- サードパーティープロバイダーによるアクション
- カスタムプロバイダーによるアクション

roleArn

宣言されたアクションを実行する IAM サービスロールの ARN。このアクションを引き受けるには、パイプラインレベルで指定した roleArn を使用します。

namespace

アクションは変数で設定できます。namespace フィールドを使用して、実行変数の名前空間と変数の情報を設定します。実行変数とアクション出力変数のリファレンス情報については、「変数リファレンス」を参照してください。

> **Note**
>
> Amazon ECR、Amazon S3、または CodeCommit ソースの場合、入力変換エントリを使用してソースオーバーライドを作成し、パイプラインイベントの EventBridge で revisionValue を使用することもできます。ここで、revisionValue はオブジェクトキー、コミット、またはイメージ ID のソースイベント変数から派生します。詳細については、Amazon ECR ソースアクションと EventBridge リソース、イベントに対して有効にしたソースを使用して Amazon S3 ソースアクションに接続する、または CodeCommit ソースアクションと EventBridge の手順に含まれる入力変換エントリのオプション手順を参照してください。

actionTypeId

アクションタイプ ID は、以下の 4 つのフィールドを組み合わせて識別します。

category

ソースアクションなど、パイプライン内のアクションまたはステップのタイプ。アクションタイプ別に有効なプロバイダーのセットがあります。アクションタイプ別の有効なプロバイダーのリストについては、「アクション構造リファレンス」を参照してください。

CodePipeline の有効な actionTypeId カテゴリ (アクションタイプ) は、以下のとおりです。

- Source
- Build
- Approval
- Deploy
- Test
- Invoke
- Compute

owner

現在サポートされているすべてのアクションタイプで、有効な所有者の文字列は、AWS、ThirdParty、または Custom のみです。アクション別の有効な所有者の文字列については、「アクション構造リファレンス」を参照してください。

詳細については、CodePipeline API リファレンスを参照してください。

version

アクションのバージョン。

provider

アクションプロバイダー (CodeDeploy など)。

- アクションカテゴリの有効なプロバイダータイプは、カテゴリによって異なります。例えば、ソースアクションカテゴリの場合、有効なプロバイダータイプは S3、CodeStarSourceConnection、CodeCommit、または Amazon ECR です。この例は、S3 プロバイダーのソースアクションの構造を示しています。

```
"actionTypeId": {
  "category": "Source",
  "owner": "AWS",
  "version": "1",
  "provider": "S3"},
```

InputArtifacts

このフィールドには、アクションカテゴリでサポートされている場合、入力アーティファクト構造が含まれます。アクションの入力アーティファクトは、前述のアクションで宣言された出力アーティファクトと完全に一致する必要がある 例えば、前述のアクションに次の宣言が含まれているとします。

```
"outputArtifacts": [
    {
    "MyApp"
    }
],
```

それ以外に出力アーティファクトが存在しない場合、次のアクションの入力アーティファクトは以下のようになります。

```
"inputArtifacts": [
    {
    "MyApp"
    }
],
```

例えば、ソースアクションはパイプラインの最初のアクションであるため、入力アーティファクトを持つことはできません。ただし、ソースアクションには、後続のアクションによって処理される出力アーティファクトが常に含まれます。ソースアクションの出力アーティファクトは、ソースリポジトリからのアプリケーションファイルで、アーティファクトバケットを介して圧縮して提供されます。これらは、ビルドコマンドを使用してアプリケーションファイルに対して実行する CodeBuild アクションなど、後続のアクションによって処理されます。

出力アーティファクトを持つことができないアクションの例として、デプロイアクションがあります。通常、デプロイアクションは最後のアクションであるため、出力アーティファクトを持ちません。

name

アクションの入力アーティファクトのアーティファクト名。

outputArtifacts

出力アーティファクト名は、パイプライン内で一意である必要があります。例えば、パイプラインには出力アーティファクト "MyApp" を含むアクションと、出力アーティファクト "MyBuiltApp" を含む別のアクションが含まれる場合があります。ただし、いずれも出力アーティファクト "MyApp" を持つ 2 つのアクションを、パイプラインに含めることはできません。

このフィールドには、アクションカテゴリでサポートされている場合、出力アーティファクト構造が含まれます。アクションの入力アーティファクトは、前のアクションで宣言した出力アーティファクトと完全に一致する必要があります。例えば、前述のアクションに次の宣言が含まれているとします。

```
"outputArtifacts": [
    {
    "MyApp"
    }
```

```
],
```

それ以外に出力アーティファクトが存在しない場合、次のアクションの入力アーティファクトは以下のようになります。

```
"inputArtifacts": [
    {
    "MyApp"
    }
],
```

例えば、ソースアクションはパイプラインの最初のアクションであるため、入力アーティファクトを持つことはできません。ただし、ソースアクションには、後続のアクションによって処理される出力アーティファクトが常に含まれます。ソースアクションの出力アーティファクトは、ソースリポジトリからのアプリケーションファイルで、アーティファクトバケットを介して圧縮して提供されます。これらは、ビルドコマンドを使用してアプリケーションファイルに対して実行する CodeBuild アクションなど、後続のアクションによって処理されます。

出力アーティファクトを持つことができないアクションの例として、デプロイアクションがあります。通常、デプロイアクションは最後のアクションであるため、出力アーティファクトを持ちません。

name

アクションの出力アーティファクトのアーティファクト名。

configuration (アクションプロバイダー別)

アクション設定には、プロバイダータイプに適した詳細とパラメータが含まれています。以下のセクションで示すアクション設定パラメータの例は S3 ソースアクションに固有のものです。

アクション設定と入力/出力アーティファクトの制限は、アクションプロバイダー別に異なる場合があります。アクションプロバイダー別のアクション設定の例のリストについては、「アクション構造リファレンス」と、「プロバイダータイプ別の有効な設定パラメータ」の表を参照してください。この表には、各アクションの設定パラメータの詳細を示す、プロバイダータイプ別のアクションリファレンスへのリンクがあります。アクションプロバイダー別の入力/出力アーティファクトの制限を示す表については、「アクションタイプ別の有効な入力/出力アーティファクトの数」を参照してください。

アクションの使用には、以下の考慮事項が適用されます。

- ソースアクションには入力アーティファクトがなく、デプロイアクションには出力アーティファクトがありません。
- 接続を使用しないソースアクションプロバイダー (S3 など) の場合は、変更の検出時にパイプラインを自動的に開始するかどうかを PollForSourceChanges パラメータで指定する必要があります。「PollForSourceChanges パラメータの有効な設定」を参照してください。
- 自動変更検出を設定してパイプラインを開始するか、変更検出を無効にするには、「ソースアクションと変更検出方法」を参照してください。
- フィルタリングを使用してトリガーを設定するには、接続のソースアクションを使用し、「トリガーとフィルタリングを使用してパイプラインを自動的に開始する」を参照してください。
- アクション別の出力変数については、「変数リファレンス」を参照してください。

> **Note**
>
> Amazon ECR、Amazon S3、または CodeCommit ソースの場合、入力変換エントリを使用してソースオーバーライドを作成し、パイプラインイベントの EventBridge で `revisionValue` を使用することもできます。ここで、`revisionValue` はオブジェクトキー、コミット、またはイメージ ID のソースイベント変数から派生します。詳細については、Amazon ECR ソースアクションと EventBridge リソース、イベントに対して有効にしたソースを使用して Amazon S3 ソースアクションに接続する、または CodeCommit ソースアクションと EventBridge の手順に含まれる入力変換エントリのオプション手順を参照してください。

> **Important**
>
> 30 日以上非アクティブなパイプラインでは、パイプラインのポーリングが無効になります。詳細については、パイプイライン構造リファレンスの「pollingDisabledAt」を参照してください。パイプラインをポーリングからイベントベースの変更検出に移行する手順については、「変更検出方法」を参照してください。

> **Note**
>
> CodeCommit および S3 ソースアクションには、設定済みの変更検出リソース (EventBridge ルール)、またはオプションを使用してソースの変更をリポジトリにポーリングする必要があります。Bitbucket、GitHub、または GitHub Enterprise Server のソースアクションを持つパ

イプラインの場合、ウェブフックを設定したり、デフォルトでポーリングを行う必要はありません。接続アクションは、変更検出を管理します。

runOrder

ステージ内のアクションの実行順序を示す正の整数。ステージ内の並列アクションは、同じ整数を持つアクションとして表示されます。例えば、実行順序が 2 の 2 つのアクションは、ステージ内の最初のアクションの実行後に並列して実行されます。

アクションの runOrder のデフォルト値は 1 です。値は、正の整数 (自然数) にする必要があります。分数、10 進数、負の数値、ゼロを使用することはできません。アクションのシリアルシーケンスを指定するには、最初のアクションに最小値を使用し、シーケンスの残りの各アクションにそれより大きい数値を使用します。並列アクションを指定するには、並列に実行する各アクションに同一の整数を使用します。コンソールで、実行するステージのレベルで [アクショングループを追加する] を選択して、アクションのシリアルシーケンスを指定できます。[アクションを追加する] を選択して、並列シーケンスを指定することもできます。[アクショングループ] は、同じレベルにある 1 つ以上のアクションの実行順序を指します。

例えば、3 つのアクションをステージのシーケンスで実行する場合、最初のアクションの runOrder 値には 1 を、2 番目のアクションの runOrder 値には 2 を、3 番目のアクションの runOrder 値には 3 を指定します。ただし、2 番目と 3 番目のアクションを並列に実行する場合、最初のアクションの runOrder 値には 1 を、2 番目と 3 番目のアクションの runOrder 値にはいずれも 2 を指定します。

> **Note**
>
> シリアルアクションの番号付けは、厳密な順序である必要はありません。例えば、シーケンスに 3 つのアクションがあり、2 番目のアクションを削除する場合、3 番目のアクションの runOrder 値の番号付けをやり直す必要はありません。アクション (3) の runOrder の値は、最初のアクション (1) の runOrder の値より大きいため、ステージの最初のアクションが実行されてから順番に実行されます。

CodePipeline の有効なアクションプロバイダー

パイプライン構造の形式は、パイプラインのアクションとステージをビルドするために使用します。アクションタイプは、アクションカテゴリとアクションプロバイダータイプの組み合わせで構成されます。

アクションカテゴリ別に有効なアクションプロバイダーのリストがあります。アクションカテゴリ別の有効なアクションプロバイダーについては、「アクション構造リファレンス」を参照してください。

アクションカテゴリごとにプロバイダーのセットが指定されています。Amazon S3 など、各アクションプロバイダーには、パイプライン構造のアクションカテゴリの `Provider` フィールドで使用する必要があるプロバイダー名 (`S3` など) があります。

パイプライン構造のアクションカテゴリセクションの `Owner` フィールドには、`AWS`、`ThirdParty`、`Custom` の 3 つの有効な値があります。

アクションプロバイダーのプロバイダー名と所有者情報については、「アクション構造リファレンス」または「アクションタイプ別の有効な入力/出力アーティファクトの数」を参照してください。

次の表は、アクションタイプ別の有効なプロバイダーのリストです。

> **Note**
>
> Bitbucket、GitHub、または GitHub Enterprise Server アクションについては、CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアクションの場合) アクションのリファレンストピックを参照してください。

アクションタイプ別の有効なアクションプロバイダー

アクションカテゴリ	有効なアクションプロバイダー	サポートされているパイプラインタイプ	アクションリファレンス
ソース	Amazon S3	V1, V2	Amazon S3 ソースアクションリファレンス

アクションカテゴリ	有効なアクションプロバイダー	サポートされているパイプラインタイプ	アクションリファレンス
	Amazon ECR	V1, V2	Amazon ECR ソースアクションリファレンス
	CodeCommit	V1, V2	CodeCommit ソースアクションリファレンス
	CodeStarSourceConnection (Bitbucket、GitHub、GitHub Enterprise Server アクションの場合)	V1, V2	CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアクションの場合)
構築	Amazon ECR `ECRBuildAndPublish` アクション	V2 のみ	`ECRBuildAndPublish` ビルドアクションリファレンス
	CodeBuild	V1, V2	AWS CodeBuild ビルドおよびテストアクションリファレンス
	コマンドアクション (「コンピューティング」を参照)	V2 のみ	
	カスタム CloudBees	V1, V2	アクションタイプ別の有効な入力/出力アーティファクトの数
	カスタム Jenkins	V1, V2	アクションタイプ別の有効な入力/出力アーティファクトの数

アクションカテゴリ	有効なアクションプロバイダー	サポートされているパイプラインタイプ	アクションリファレンス
	カスタム TeamCity	V1, V2	アクションタイプ別の有効な入力/出力アーティファクトの数
テスト	CodeBuild	V1, V2	AWS CodeBuild ビルドおよびテストアクションリファレンス
	AWS Device Farm	V1, V2	アクションタイプ別の有効な入力/出力アーティファクトの数
	カスタム BlazeMeter	V1, V2	アクションタイプ別の有効な入力/出力アーティファクトの数
	サードパーティー GhostInspector		アクションタイプ別の有効な入力/出力アーティファクトの数
	カスタム Jenkins		アクションタイプ別の有効な入力/出力アーティファクトの数
	サードパーティー Micro Focus StormRunner Load		アクションタイプ別の有効な入力/出力アーティファクトの数
	サードパーティー Nouvola		アクションタイプ別の有効な入力/出力アーティファクトの数
	サードパーティー Runscope		アクションタイプ別の有効な入力/出力アーティファクトの数

アクションカテゴリ	有効なアクションプロバイダー	サポートされているパイプラインタイプ	アクションリファレンス
デプロイ	Amazon S3		Amazon S3 デプロイアクションリファレンス
	CloudFormation		CloudFormation デプロイアクションリファレンス
	CodeDeploy		アクションタイプ別の有効な入力/出力アーティファクトの数
	EC2 デプロイアクション	V2 のみ	Amazon EC2 アクションリファレンス
	Amazon ECS		アクションタイプ別の有効な入力/出力アーティファクトの数
	Amazon ECS (Blue/Green) (これは `CodeDeployToECS` アクションです)		アクションタイプ別の有効な入力/出力アーティファクトの数
	Amazon EKS アクション	V2 のみ	Amazon Elastic Kubernetes Service の EKS デプロイアクションリファレンス
	(Elastic Beanstalk)		アクションタイプ別の有効な入力/出力アーティファクトの数
	AWS AppConfig		AWS AppConfig デプロイアクションリファレンス

アクションカテゴリ	有効なアクションプロバイダー	サポートされているパイプラインタイプ	アクションリファレンス
	OpsWorks		アクションタイプ別の有効な入力/出力アーティファクトの数
	Service Catalog		アクションタイプ別の有効な入力/出力アーティファクトの数
	Amazon Alexa		アクションタイプ別の有効な入力/出力アーティファクトの数
	カスタム XebiaLabs		アクションタイプ別の有効な入力/出力アーティファクトの数
承認	手動		アクションタイプ別の有効な入力/出力アーティファクトの数
Invoke	CodePipeline 呼び出しアクション		AWS CodePipeline アクションリファレンスを呼び出す
	AWS Lambda		AWS Lambda アクションリファレンスを呼び出す
	AWS Step Functions		AWS Step Functions アクションリファレンスを呼び出す

アクションカテゴリ	有効なアクションプロバイダー	サポートされているパイプラインタイプ	アクションリファレンス
	InspectorScan		Amazon Inspector InspectorScan 呼び出しアクションリファレンス
コンピューティング	コマンドアクション		コマンドアクションリファレンス

CodePipeline の一部のアクションタイプは、一部の AWS リージョンでのみ使用できます。アクションタイプは AWS リージョンで使用できますが、そのアクションタイプの AWS プロバイダーは利用できません。

各アクションプロバイダーの詳細については、「CodePipeline アクションタイプとの統合」を参照してください。

PollForSourceChanges パラメータの有効な設定

PollForSourceChanges パラメータのデフォルト設定は、以下の表に示すように、パイプラインの作成に使用した方法によって決まります。多くの場合、PollForSourceChanges パラメータはデフォルトで true になるため、無効にする必要があります。

PollForSourceChanges パラメータがデフォルトで true になる場合は、以下の操作を行う必要があります。

- PollForSourceChanges パラメータを JSON ファイルまたは CloudFormation テンプレートに追加します。
- 変更検出リソース (必要に応じて CloudWatch Events ルール) を作成します。
- PollForSourceChanges パラメータを false に設定します。

> **Note**
>
> CloudWatch Events ルールまたはウェブフックを作成する場合は、このパラメータを false に設定してパイプラインが複数回トリガーされるのを避ける必要があります。

PollForSourceChanges パラメータは、Amazon ECR ソースアクションには使用されません。

- **PollForSourceChanges** パラメータのデフォルト

ソース	作成方法	JSON 構造の出力の設定例
CodeCommit	パイプラインはコンソールで作成されます (変更検出リソースもコンソールで作成されます)。パラメータは、パイプライン構造の出力に表示され、デフォルトで false になります。	`BranchName": "main", "PollForSourceChanges": "false", "RepositoryName": "my-repo"`
	パイプラインは CLI または で作成され CloudFormation、 PollForSourceChanges パラメータは JSON 出力に表示されませんが、 は .2 に設定されますtrue。	`BranchName": "main", "RepositoryName": "my-repo"`
Amazon S3	パイプラインはコンソールで作成されます (変更検出リソースもコンソールで作成されます)。パラメータは、パイプライン構造の出力に表示され、デフォルトで false になります。	`"S3Bucket": "my-bucket", "S3ObjectKey": "object.zip", "PollForSourceChanges": "false"`
	パイプラインは CLI または で作成され CloudFormation、 PollForSourceChanges パラメータは JSON 出力に表示されませんが、 は .2 に設定されますtrue。	`"S3Bucket": "my-bucket", "S3ObjectKey": "object.zip"`

ソース	作成方法	JSON 構造の出力の設定例
GitHub	パイプラインはコンソールで作成されます (変更検出リソースもコンソールで作成されます)。パラメータは、パイプライン構造の出力に表示され、デフォルトで false になります。	"Owner": "*MyGitHubAccountName* ", "Repo": " *MyGitHubRepositoryName* " "PollForSourceChanges": "false", "Branch": " *main*" "OAuthToken": " ****"
	パイプラインは CLI または で作成され CloudFormation、 PollForSourceChanges パラメータは JSON 出力に表示されませんが、 は .2 に設定されますtrue。	"Owner": "*MyGitHubAccountName* ", "Repo": "*MyGitHubRepositoryName* ", "Branch": " *main*", "OAuthToken": " ****"
	2 PollForSourceChanges が JSON 構造または CloudFormation テンプレートに任意の時点で追加されている場合は、次のように表示されます。 "PollForSourceChanges": "true",	
	[3] 各ソースプロバイダーに適用される変更検出リソースの詳細については、「変更検出方法」を参照してください。	

アクションタイプ別の有効な入力/出力アーティファクトの数

アクションタイプおよびプロバイダーに応じて、入力/出力アーティファクトを以下の数にすることができます。

アーティファクトのアクションタイプの制約

所有者	アクションのタイプ	プロバイダー	入力アーティファクト有効な数	出力アーティファクトの有効な数
AWS	ソース	S3	0	1
AWS	ソース	CodeCommit	0	1
AWS	ソース	ECR	0	1
ThirdParty	ソース	CodeStarSourceConnection	0	1
AWS	構築	CodeBuild	1～5	0～5
AWS	テスト	CodeBuild	1～5	0～5
AWS	テスト	DeviceFarm	1	0
AWS	承認	ThirdParty	0	0
AWS	デプロイ	S3	1	0
AWS	デプロイ	CloudFormation	0～10	0～1
AWS	デプロイ	CodeDeploy	1	0
AWS	デプロイ	ElasticBeanstalk	1	0
AWS	デプロイ	OpsWorks	1	0
AWS	デプロイ	ECS	1	0
AWS	デプロイ	ServiceCatalog	1	0

所有者	アクションのタイプ	プロバイダー	入力アーティファクト有効な数	出力アーティファクトの有効な数
AWS	Invoke	Lambda	0～5	0～5
ThirdParty	デプロイ	AlexaSkillsKit	1～2	0
Custom	構築	Jenkins	0～5	0～5
Custom	テスト	Jenkins	0～5	0～5
Custom	サポートされているすべてのカテゴリ	カスタムアクションで指定	0～5	0～5

プロバイダータイプ別の有効な設定パラメータ

このセクションでは、アクションプロバイダー別の有効な configuration パラメータを示します。

各アクションには有効なアクション設定が必要です。この設定は、アクションのプロバイダータイプによって異なります。次の表は、有効な各プロバイダータイプで必要なアクション設定要素を示しています。

プロバイダータイプのアクション設定プロパティ

プロバイダー名	アクションタイプでのプロバイダー名	設定プロパティ	必須/オプション
Amazon S3 (デプロイアクションプロバイダー)	Amazon S3 デプロイアクションパラメータに関連する例などの詳細については、「Amazon S3 デプロイアクションリファレンス」を参照してください。		
Amazon S3 (ソースアク	Amazon S3 ソースアクションパラメータに関連する例などの詳細については、「Amazon S3 ソースアクションリファレンス」を参照してください。		

プロバイダー名	アクションタイプでのプロバイダー名	設定プロパティ	必須/オプション
ションプロバイダー)			
Amazon ECR	Amazon ECR パラメータに関連する例などの詳細については、「Amazon ECR ソースアクションリファレンス」を参照してください。		
CodeCommit	CodeCommit パラメータに関連する例などの詳細については、「CodeCommit ソースアクションリファレンス」を参照してください。		
Bitbucket、GitHub (GitHub アプリ経由)、GHES、GitLab の CodeStarSourceConnection アクション	アクション設定の例などの詳細については、「設定パラメータ」を参照してください。		
GitHub (OAuth アプリ経由)	GitHub パラメータに関連する例などの詳細については、「GitHub (OAuth アプリ経由) ソースアクションリファレンス」を参照してください。これはバージョン 1 の GitHub アクションです。		
AWS CloudFormation	AWS CloudFormation パラメータに関連する例を含む詳細については、「」を参照してくださいCloudFormation デプロイアクションリファレンス。		
CodeBuild	CodeBuild パラメータに関連する詳細な説明と例については、「AWS CodeBuild ビルドおよびテストアクションリファレンス」を参照してください。		
CodeDeploy	CodeDeploy パラメータに関連する詳細な説明と例については、「AWS CodeDeploy デプロイアクションリファレンス」を参照してください。		

プロバイダー名	アクションタイプでのプロバイダー名	設定プロパティ	必須/オプション
AWS Device Farm	AWS Device Farm パラメータに関する説明と例については、「」を参照してくださいAWS Device Farm テストアクションリファレンス。		
AWS Elastic Beanstalk	`ElasticBeanstalk`	`ApplicationName`	必須
		`EnvironmentName`	必須
AWS Lambda	AWS Lambda パラメータに関連する例を含む詳細については、「」を参照してくださいAWS Lambda アクションリファレンスを呼び出す。		
AWS OpsWorks Stacks	`OpsWorks`	`Stack`	必須
		`Layer`	オプションです。
		`App`	必須
Amazon ECS	Amazon ECS パラメータに関連する詳細な説明と例については、「Amazon Elastic Container Service デプロイアクションリファレンス」を参照してください。		
Amazon ECS と CodeDeploy (Blue/Green)	Amazon ECS および CodeDeploy blue/green パラメータに関連する詳細な説明と例については、「Amazon ECS および CodeDeploy ブルー/グリーンデプロイアクションリファレンス」を参照してください。		
Service Catalog	`ServiceCatalog`	`TemplateFilePath`	必須
		`ProductVersionName`	必須
		`ProductType`	必須
		`ProductVersionDescription`	オプションです。
		`ProductId`	必須

プロバイダー名	アクションタイプでのプロバイダー名	設定プロパティ	必須/オプション
Alexa Skills Kit	`AlexaSkillsKit`	`ClientId`	必須
		`ClientSecret`	必須
		`RefreshToken`	必須
		`SkillId`	必須
Jenkins	CodePipeline Plugin for Jenkins で指定したアクションの名前 (*`MyJenkins ProviderName`* など)	`ProjectName`	必須
手動承認	`Manual`	`CustomData`	オプションです。
		`ExternalEntityLink`	オプションです。
		`NotificationArn`	オプションです。

次の例は、Alexa Skills Kit を使用するデプロイアクションの有効な設定を示しています。

```
"configuration": {
  "ClientId": "amzn1.application-oa2-client.aadEXAMPLE",
  "ClientSecret": "****",
  "RefreshToken": "****",
  "SkillId": "amzn1.ask.skill.22649d8f-0451-4b4b-9ed9-bfb6cEXAMPLE"
}
```

次の例は、手動承認の有効な設定を示しています。

```
"configuration": {
  "CustomData": "Comments on the manual approval",
  "ExternalEntityLink": "http://my-url.com",
  "NotificationArn": "arn:aws:sns:us-west-2:12345EXAMPLE:Notification"
```

```
}
```

アクション構造リファレンス

このセクションは、アクション設定のみのリファレンスです。パイプライン構造の概念的な概要については、「CodePipeline パイプライン構造リファレンス」を参照してください。

CodePipeline の各アクションプロバイダーは、パイプライン構造の必須設定項目およびオプションの設定項目セットを使用します。このセクションでは、アクションプロバイダー別の以下のリファレンス情報を提供します。

- `Owner` や `Provider` などのパイプライン構造アクションブロックに含まれる `ActionType` フィールドの有効な値。
- パイプライン構造のアクションセクションに含まれる `Configuration` パラメータの説明およびその他のリファレンス情報（必須およびオプション）。
- JSON および YAML アクションフィールドの有効な例。

このセクションは、より多くのアクションプロバイダーで定期的に更新されます。リファレンス情報は現在、次のアクションプロバイダーで利用できます。

トピック

- Amazon EC2 アクションリファレンス
- Amazon ECR ソースアクションリファレンス
- ECRBuildAndPublish ビルドアクションリファレンス
- Amazon ECS および CodeDeploy ブルー/グリーンデプロイアクションリファレンス
- Amazon Elastic Container Service デプロイアクションリファレンス
- Amazon Elastic Kubernetes Service の EKS デプロイアクションリファレンス
- AWS Lambda デプロイアクションリファレンス
- Amazon S3 デプロイアクションリファレンス
- Amazon S3 ソースアクションリファレンス
- AWS AppConfig デプロイアクションリファレンス
- CloudFormation デプロイアクションリファレンス
- CloudFormation StackSets デプロイアクションリファレンス
- AWS CodeBuild ビルドおよびテストアクションリファレンス

- AWS CodePipeline アクションリファレンスを呼び出す
- CodeCommit ソースアクションリファレンス
- AWS CodeDeploy デプロイアクションリファレンス
- CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアクションの場合)
- コマンドアクションリファレンス
- AWS Device Farm テストアクションリファレンス
- Elastic Beanstalk のデプロイアクションリファレンス
- Amazon Inspector InspectorScan 呼び出しアクションリファレンス
- AWS Lambda アクションリファレンスを呼び出す
- AWS OpsWorks デプロイアクションリファレンス
- AWS Service Catalog デプロイアクションリファレンス
- AWS Step Functions アクションリファレンスを呼び出す

Amazon EC2 アクションリファレンス

Amazon EC2 EC2 アクションを使用して、デプロイフリートにアプリケーションコードをデプロイします。デプロイフリートは、Amazon EC2 Linux インスタンスまたは Linux SSM マネージドノードで構成できます。インスタンスに SSM 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。

> **Note**
>
> このアクションは Linux インスタンスタイプのみをサポートします。サポートされているフリートの最大サイズは 500 インスタンスです。

アクションは、指定された最大値に基づいてインスタンスの数を選択します。以前のインスタンスで失敗したインスタンスが最初に選択されます。以前に失敗したアクションなど、インスタンスが既に同じ入力アーティファクトのデプロイを受信している場合、アクションは特定のインスタンスでのデプロイをスキップします。

> **Note**
>
> このアクションは V2 タイプのパイプラインでのみサポートされています。

トピック

- アクションタイプ
- 設定パラメータ
- 入力アーティファクト
- 出力アーティファクト
- EC2 デプロイアクションのサービスロールポリシーのアクセス許可
- 仕様ファイルのリファレンスをデプロイする
- アクションの宣言
- デプロイ仕様を使用したアクション宣言の例
- 関連情報

アクションタイプ

- カテゴリ:`Deploy`
- 所有者: `AWS`
- プロバイダー: `EC2`
- バージョン: `1`

設定パラメータ

InstanceTagKey

必須: はい

Amazon EC2 で作成したインスタンスのタグキー (`Name` など)。

InstanceTagValue

必須: いいえ

Amazon EC2 で作成したインスタンスのタグの値 (`my-instances` など)。

この値を指定しない場合、InstanceTagKey を持つすべてのインスタンスがマッチします。

InstanceType

必須: はい

Amazon EC2 で作成されたインスタンスまたは SSM ノードのタイプ。有効な値は `EC2` および `SSM_MANAGED_NODE` です。

すべてのインスタンスで SSM エージェントを作成、タグ付け、インストールしておく必要があります。

> **Note**
>
> インスタンスを作成する際は、既存の EC2 インスタンスロールを作成または使用します。`Access Denied` エラーを回避するには、インスタンスロールに S3 バケットアクセス許可を追加して、CodePipeline アーティファクトバケットにインスタンスアクセス許可を付与する必要があります。デフォルトのロールを作成するか、パイプラインのリージョンのアーティファクトバケットにスコープダウンされた `s3:GetObject` アクセス許可で既存のロールを更新します。

TargetDirectory

必須: はい (スクリプトが指定されている場合)

スクリプトを実行するために Amazon EC2 インスタンスで使用されるディレクトリ。

DeploySpec

必須: はい (デプロイ仕様が指定されている場合)

デプロイのインストールイベントとライフサイクルイベントを設定するために使用されるファイル。デプロイ仕様フィールドの説明と情報については、「仕様ファイルのリファレンスをデプロイする」を参照してください。指定されたデプロイ仕様ファイルを含むアクション設定を表示するには、「デプロイ仕様を使用したアクション宣言の例」の例を参照してください。

MaxBatch

必須: いいえ

並列でデプロイできるインスタンスの最大数。

MaxError

必須: いいえ

デプロイ中に許可されるインスタンスエラーの最大数。

TargetGroupNameList

必須: いいえ

デプロイのターゲットグループ名のリスト。ターゲットグループは作成済みである必要があります。

ターゲットグループは、特定のリクエストを処理するための一連のインスタンスを提供します。ターゲットグループが指定されている場合、インスタンスはデプロイ前にターゲットグループから削除され、デプロイ後にターゲットグループに追加されます。

PreScript

必須: いいえ

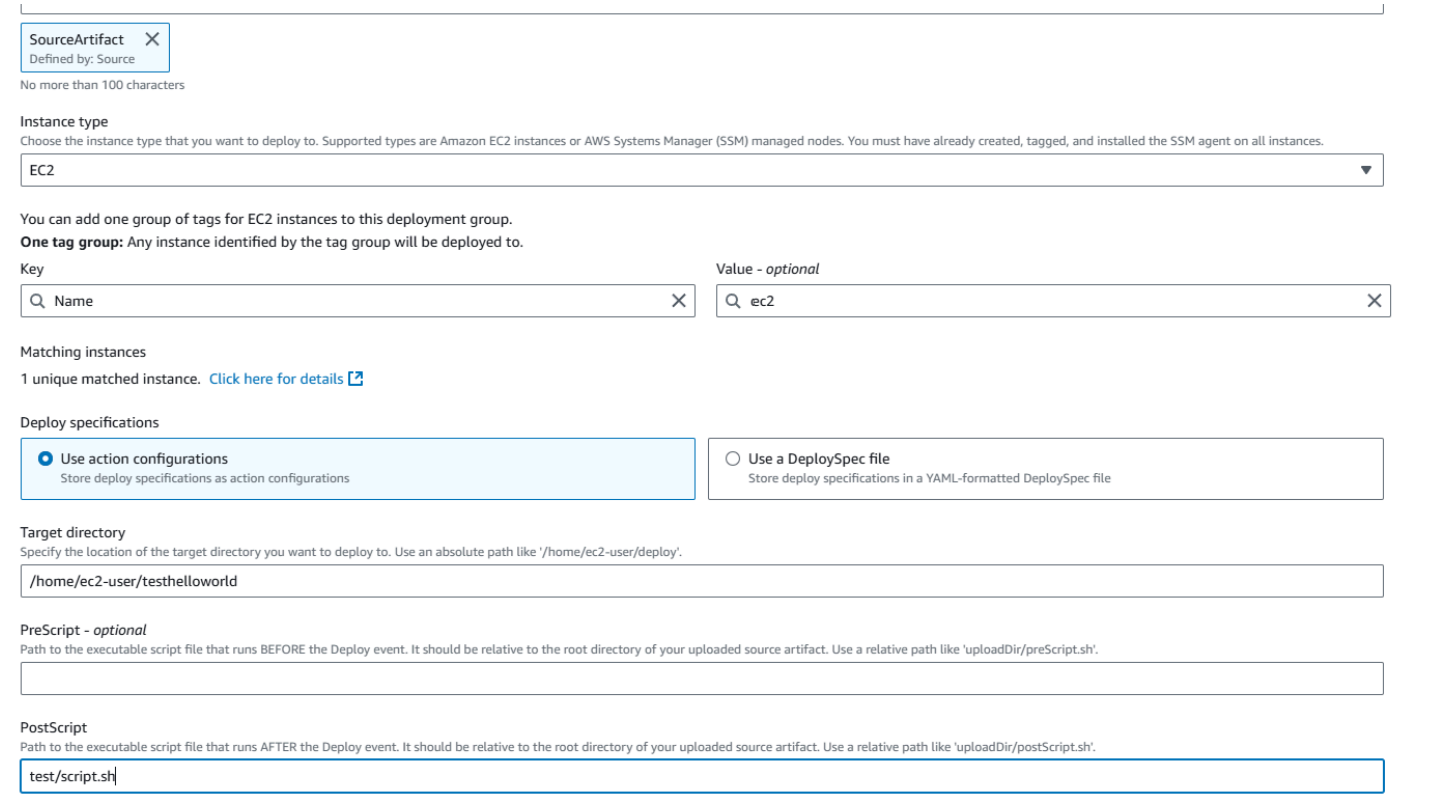
アクションデプロイフェーズの前に実行するスクリプト。

PostScript

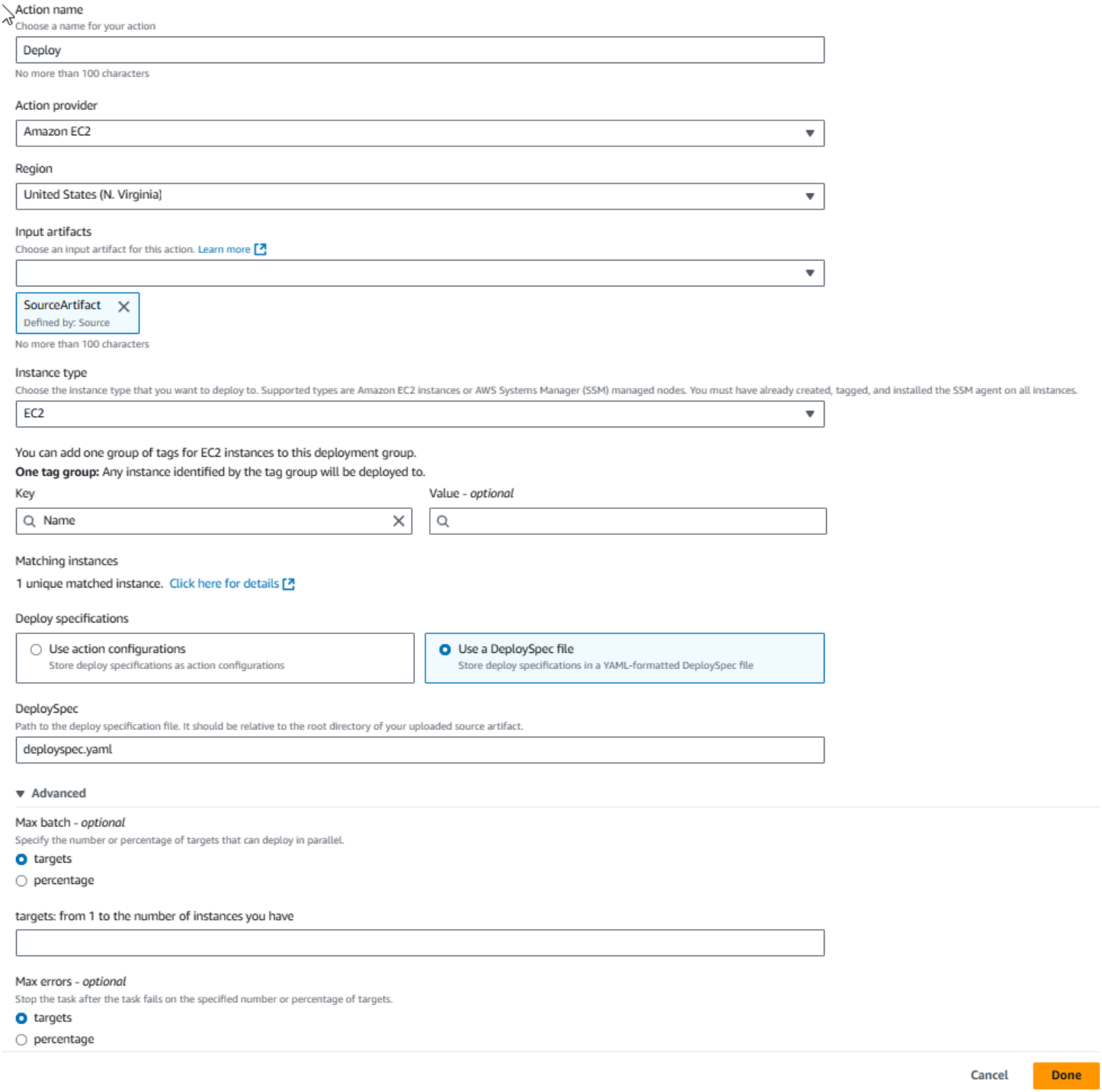
必須: はい

アクションデプロイフェーズの後に実行されるスクリプト。

次の図は、[アクションの設定を使用] が選択されているアクションの [編集] ページの例を示しています。



次の図は、[DeploySpec ファイルを使用] が選択されているアクションの [編集] ページの例を示しています。



入力アーティファクト

- アーティファクトの数: 1
- 説明: デプロイ中にスクリプトアクションをサポートするために提供されたファイル。

出力アーティファクト

- アーティファクトの数: `0`
- 説明: 出力アーティファクトは、このアクションタイプには適用されません。

EC2 デプロイアクションのサービスロールポリシーのアクセス許可

CodePipeline がアクションを実行する際、CodePipeline サービスロールには、最小特権でのアクセスに適切にスコープダウンされた次のアクセス許可が必要です。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "StatementWithAllResource",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeInstances",
                "elasticloadbalancing:DescribeTargetGroupAttributes",
                "elasticloadbalancing:DescribeTargetGroups",
                "elasticloadbalancing:DescribeTargetHealth",
                "ssm:CancelCommand",
                "ssm:DescribeInstanceInformation",
                "ssm:ListCommandInvocations"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "StatementForLogs",
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": [
```

```
                "arn:aws:logs:us-east-1:111122223333:log-group:/aws/codepipeline/
{{pipelineName}}:*"
            ]
        },
        {
            "Sid": "StatementForElasticloadbalancing",
            "Effect": "Allow",
            "Action": [
                "elasticloadbalancing:DeregisterTargets",
                "elasticloadbalancing:RegisterTargets"
            ],
            "Resource": [
                "arn:aws:elasticloadbalancing:us-east-1:111122223333:targetgroup/
[[targetGroupName]]/*"
            ]
        },
        {
            "Sid": "StatementForSsmOnTaggedInstances",
            "Effect": "Allow",
            "Action": [
                "ssm:SendCommand"
            ],
            "Resource": [
                "arn:aws:ec2:us-east-1:111122223333:instance/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/{{tagKey}}": "{{tagValue}}"
                }
            }
        },
        {
            "Sid": "StatementForSsmApprovedDocuments",
            "Effect": "Allow",
            "Action": [
                "ssm:SendCommand"
            ],
            "Resource": [
                "arn:aws:ssm:us-east-1::document/AWS-RunPowerShellScript",
                "arn:aws:ssm:us-east-1::document/AWS-RunShellScript"
            ]
        }
    ]
```

```
}
```

CloudWatch ログのパイプラインのロググループ

CodePipeline は、アクションを実行するときに、次のようにパイプライン名を使用してロググループを作成します。これにより、パイプライン名を使用してアクセス許可の範囲をリソースのログ記録に絞り込むことができます。

```
/aws/codepipeline/MyPipelineName
```

ログ記録のための以下のアクセス許可は、サービスロールの上記の更新に含まれています。

- logs:CreateLogGroup
- logs:CreateLogStream
- logs:PutLogEvents

アクションの詳細ダイアログページを使用してコンソールでログを表示するには、ログを表示するアクセス許可をコンソールロールに追加する必要があります。詳細については、「コンソールでコンピューティングログを表示するために必要なアクセス許可」でコンソールのアクセス許可ポリシーの例を参照してください。

CloudWatch ログのサービスロールポリシーのアクセス許可

CodePipeline は、アクションを実行するときに、次のようにパイプライン名を使用してロググループを作成します。これにより、パイプライン名を使用してアクセス許可の範囲をリソースのログ記録に絞り込むことができます。

```
/aws/codepipeline/MyPipelineName
```

アクションの詳細ダイアログページを使用してコンソールでログを表示するには、ログを表示するアクセス許可をコンソールロールに追加する必要があります。詳細については、「コンソールでコンピューティングログを表示するために必要なアクセス許可」でコンソールのアクセス許可ポリシーの例を参照してください。

仕様ファイルのリファレンスをデプロイする

CodePipeline がアクションを実行する際に、インスタンスへのデプロイを設定する仕様ファイルを指定できます。デプロイ仕様ファイルは、デプロイライフサイクルイベントに応じてインストールす

る内容と実行するライフサイクルイベントフックを指定します。デプロイ仕様ファイルは常に YAML 形式です。デプロイ仕様ファイルは、以下の目的で使用されます。

- 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。

デプロイ仕様ファイルは、AppSpec ファイルを使用して CodeDeploy でサポートされている特定のデプロイ設定パラメータをサポートします。既存の AppSpec ファイルを直接使用でき、サポートされていないパラメータは無視されます。CodeDeploy の AppSpec ファイルの詳細については、「CodeDeploy ユーザーガイド」のアプリケーション仕様ファイルリファレンスを参照してください。

ファイルデプロイパラメータは次のように指定されます。

- `files` - デプロイ仕様ファイルは、デプロイファイルの `source:` と `destination:` を指定します。
- `scripts` - デプロイのスクリプト化されたイベント。`BeforeDeploy` と `AfterDeploy` の 2 つのイベントがサポートされています。
- `hooks` - イベントのライフサイクルフック。サポートされるフックは、`ApplicationStop`、`BeforeInstall`、`AfterInstall`、`ApplicationStart`、および `ValidateService` です。

> **Note**
>
> フックパラメータは CodeDeploy との AppSpec の互換性のために使用でき、バージョン 0.0 (AppSpec 形式) でのみ使用できます。この形式では、CodePipeline はイベントのベストエフォートマッピングを実行します。

仕様ファイルでは正しい YAML スペースを使用する必要があります。デプロイ仕様ファイル内の場所とスペース数が正しくない場合、エラーが発生します。間隔の詳細については、YAML の仕様を参照してください。

デプロイ仕様ファイルの例を以下に示します。

```
version: 0.1
files:
  - source: /index.html
    destination: /var/www/html/
scripts:
  BeforeDeploy:
    - location: scripts/install_dependencies
      timeout: 300
      runas: myuser
  AfterDeploy:
    - location: scripts/start_server
      timeout: 300
      runas: myuser
```

指定されたデプロイ仕様ファイルを含むアクション設定を表示するには、「デプロイ仕様を使用したアクション宣言の例」の例を参照してください。

アクションの宣言

YAML

```
name: DeployEC2
actions:
- name: EC2
  actionTypeId:
    category: Deploy
    owner: AWS
    provider: EC2
    version: '1'
  runOrder: 1
  configuration:
    InstanceTagKey: Name
    InstanceTagValue: my-instances
    InstanceType: EC2
    PostScript: "test/script.sh",
    TargetDirectory: "/home/ec2-user/deploy"
  outputArtifacts: []
  inputArtifacts:
  - name: SourceArtifact
  region: us-east-1
```

JSON

```
{
    "name": "DeployEC2",
    "actions": [
        {
            "name": "EC2Deploy",
            "actionTypeId": {
                "category": "Deploy",
                "owner": "AWS",
                "provider": "EC2",
                "version": "1"
            },
            "runOrder": 1,
            "configuration": {
                "InstanceTagKey": "Name",
                "InstanceTagValue": "my-instances",
                "InstanceType": "EC2",
                "PostScript": "test/script.sh",
                "TargetDirectory": "/home/ec2-user/deploy"
            },
            "outputArtifacts": [],
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-east-1"
        }
    ]
},
```

デプロイ仕様を使用したアクション宣言の例

YAML

```
name: DeployEC2
actions:
- name: EC2
  actionTypeId:
    category: Deploy
    owner: AWS
```

```
    provider: EC2
    version: '1'
  runOrder: 1
  configuration:
    DeploySpec: "deployspec.yaml"
    InstanceTagKey: Name
    InstanceTagValue: my-instances
    InstanceType: EC2
  outputArtifacts: []
  inputArtifacts:
  - name: SourceArtifact
  region: us-east-1
```

JSON

```
{
    "name": "DeployEC2",
    "actions": [
        {
            "name": "EC2Deploy",
            "actionTypeId": {
                "category": "Deploy",
                "owner": "AWS",
                "provider": "EC2",
                "version": "1"
            },
            "runOrder": 1,
            "configuration": {
                "DeploySpec": "deployspec.yaml",
                "InstanceTagKey": "Name",
                "InstanceTagValue": "my-instances",
                "InstanceType": "EC2"
            },
            "outputArtifacts": [],
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-east-1"
        }
    ]
},
```

関連情報

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- チュートリアル: CodePipeline を使用して Amazon EC2 インスタンスにデプロイする – このチュートリアルでは、スクリプトファイルをデプロイする EC2 インスタンスの作成と、EC2 アクションを使用したパイプラインの作成について説明します。
- EC2 デプロイアクションがエラーメッセージ `No such file` で失敗する – このトピックでは、EC2 アクションでファイルが見つからないエラーのトラブルシューティングについて説明します。

Amazon ECR ソースアクションリファレンス

新規イメージが Amazon ECR リポジトリにプッシュされた場合、パイプラインをトリガーします。このアクションは、Amazon ECR にプッシュされたイメージの URI を参照するイメージ定義ファイルを提供します。このソースアクションは、他のソースアーティファクトのソース場所を許可するために、CodeCommit などの他のソースアクションと組み合わせて使用されることがよくあります。詳細については、「チュートリアル: Amazon ECR ソース、ECS - CodeDeploy 間のデプロイでパイプラインを作成する」を参照してください。

コンソールを使用してパイプラインを作成または編集すると、CodePipeline はリポジトリで変更が生じた場合にパイプラインを開始する EventBridge ルールを作成します。

> **Note**
>
> Amazon ECR、Amazon S3、または CodeCommit ソースの場合、入力変換エントリを使用してソースオーバーライドを作成し、パイプラインイベントの EventBridge で `revisionValue` を使用することもできます。ここで、`revisionValue` はオブジェクトキー、コミット、またはイメージ ID のソースイベント変数から派生します。詳細については、Amazon ECR ソースアクションと EventBridge リソース、イベントに対して有効にしたソースを使用して Amazon S3 ソースアクションに接続する、または CodeCommit ソースアクションと EventBridge の手順に含まれる入力変換エントリのオプション手順を参照してください。

Amazon ECR アクションを介してパイプラインを接続する前に、Amazon ECR リポジトリを作成し、イメージをプッシュしておく必要があります。

トピック

- アクションタイプ
- 設定パラメータ
- 入力アーティファクト
- 出力アーティファクト
- 出力変数
- サービスロールのアクセス許可: Amazon ECR アクション
- アクションの宣言 (Amazon ECR の例)
- 関連情報

アクションタイプ

- カテゴリ:`Source`
- 所有者: `AWS`
- プロバイダー: `ECR`
- バージョン: `1`

設定パラメータ

RepositoryName

必須: はい

イメージがプッシュされた Amazon ECR リポジトリの名前。

ImageTag

必須: いいえ

イメージに使用するタグ。

> **Note**
>
> `ImageTag` の値を指定しない場合、デフォルト値は `latest` になります。

入力アーティファクト

- アーティファクトの数: `0`
- 説明: 入力アーティファクトは、このアクションタイプには適用されません。

出力アーティファクト

- アーティファクトの数: `1`
- 説明: このアクションは、パイプライン実行をトリガーしたイメージの URI を含む `imageDetail.json` ファイルがあるアーティファクトを生成します。`imageDetail.json` ファイルの詳細については、「Amazon ECS Blue/Green デプロイアクション用の imageDetail.json ファイル」を参照してください。

出力変数

このアクションを設定すると、パイプライン内のダウンストリームアクションのアクション設定によって参照できる変数が生成されます。このアクションは、アクションに名前空間がない場合でも、出力変数として表示できる変数を生成します。名前空間を使用してアクションを設定し、これらの変数をダウンストリームアクションの設定で使用できるようにします。

詳細については、「変数リファレンス」を参照してください。

RegistryId

リポジトリを含むレジストリに関連付けられた AWS アカウント ID。

RepositoryName

イメージがプッシュされた Amazon ECR リポジトリの名前。

ImageTag

イメージに使用するタグ。

> **Note**
> ソースリビジョンが上書きされると、`ImageTag`出力変数は出力されません。

ImageDigest

イメージマニフェストの sha256 ダイジェスト。

ImageURI

イメージの URL。

サービスロールのアクセス許可: Amazon ECR アクション

Amazon ECR がサポートされるように、以下をポリシーステートメントに追加します。

```
{
    "Effect": "Allow",
    "Action": [
        "ecr:DescribeImages"
    ],
    "Resource": "resource_ARN"
},
```

このアクションの詳細については、「Amazon ECR ソースアクションリファレンス」を参照してください。

アクションの宣言 (Amazon ECR の例)

YAML

```
Name: Source
Actions:
  - InputArtifacts: []
    ActionTypeId:
      Version: '1'
      Owner: AWS
      Category: Source
      Provider: ECR
    OutputArtifacts:
      - Name: SourceArtifact
    RunOrder: 1
    Configuration:
      ImageTag: latest
      RepositoryName: my-image-repo
```

```
    Name: ImageSource
```

JSON

```
{
    "Name": "Source",
    "Actions": [
        {
            "InputArtifacts": [],
            "ActionTypeId": {
                "Version": "1",
                "Owner": "AWS",
                "Category": "Source",
                "Provider": "ECR"
            },
            "OutputArtifacts": [
                {
                    "Name": "SourceArtifact"
                }
            ],
            "RunOrder": 1,
            "Configuration": {
                "ImageTag": "latest",
                "RepositoryName": "my-image-repo"
            },
            "Name": "ImageSource"
        }
    ]
},
```

関連情報

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- チュートリアル: Amazon ECR ソース、ECS - CodeDeploy 間のデプロイでパイプラインを作成する – このチュートリアルでは、サンプル AppSpec ファイルとサンプル CodeDeploy アプリケーションおよびデプロイグループを提供し、Amazon ECS インスタンスにデプロイする CodeCommit と Amazon ECR ソースを使用してパイプラインを作成します。

ECRBuildAndPublish ビルドアクションリファレンス

このビルドアクションにより、ソースで変更が発生したときに新しいイメージの構築とプッシュを自動化できます。このアクションは、指定された Dockerfile の場所に基づいて構築され、イメージをプッシュします。このビルドアクションは、Amazon ECR ソースリポジトリで変更が発生したときにパイプラインをトリガーする CodePipeline の Amazon ECR ソースアクションとは異なります。このアクションの詳細については、「Amazon ECR ソースアクションリファレンス」を参照してください。

これは、パイプラインをトリガーするソースアクションではありません。このアクションはイメージをビルドし、Amazon ECR イメージリポジトリにプッシュします。

アクションをパイプラインに追加する前に、Amazon ECR リポジトリを作成し、GitHub などのソースコードリポジトリに Dockerfile を追加しておく必要があります。

> **Important**
>
> このアクションでは、CodePipeline マネージド CodeBuild コンピューティングを使用して、ビルド環境でコマンドを実行します。コマンドアクションを実行すると、 AWS CodeBuildで別途料金が発生します。

> **Note**
>
> このアクションは V2 タイプのパイプラインでのみ使用できます。

トピック

- アクションタイプ
- 設定パラメータ
- 入力アーティファクト
- 出力アーティファクト
- 出力変数
- サービスロールのアクセス許可: ECRBuildAndPublish アクション
- アクションの宣言

- 関連情報

アクションタイプ

- カテゴリ:Build
- 所有者: AWS
- プロバイダー: ECRBuildAndPublish
- バージョン: 1

設定パラメータ

ECRRepositoryName

必須: はい

イメージがプッシュされた Amazon ECR リポジトリの名前。

DockerFilePath

必須: いいえ

イメージの構築に使用される Dockerfile の場所。必要に応じて、ルートレベルでない場合は、代替の Dockerfile の場所を指定できます。

> **Note**
>
> DockerFilePath の値を指定しない場合、値はデフォルトでソースリポジトリのルートレベルになります。

ImageTags

必須: いいえ

イメージに使用するタグ。複数のタグを文字列のカンマ区切りリストとして入力できます。

> **Note**
>
> ImageTags の値を指定しない場合、デフォルト値は latest になります。

RegistryType

必須: いいえ

リポジトリがパブリックかプライベートかを指定します。有効な値は `private | public` です。

> **Note**
>
> `RegistryType` の値を指定しない場合、デフォルト値は `private` になります。

入力アーティファクト

- アーティファクトの数: `1`
- 説明: イメージのビルドに必要な Dockerfile を含むソースアクションによって生成されるアーティファクト。

出力アーティファクト

- アーティファクトの数: `0`

出力変数

このアクションを設定すると、パイプライン内のダウンストリームアクションのアクション設定によって参照できる変数が生成されます。このアクションは、アクションに名前空間がない場合でも、出力変数として表示できる変数を生成します。名前空間を使用してアクションを設定し、これらの変数をダウンストリームアクションの設定で使用できるようにします。

詳細については、「変数リファレンス」を参照してください。

ECRImageDigestId

イメージマニフェストの `sha256` ダイジェスト。

ECRRepositoryName

イメージがプッシュされた Amazon ECR リポジトリの名前。

サービスロールのアクセス許可: **ECRBuildAndPublish** アクション

ECRBuildAndPublish アクションをサポートするには、以下をポリシーステートメントに追加します。

```
{
    "Statement": [
         {
            "Sid": "ECRRepositoryAllResourcePolicy",
            "Effect": "Allow",
            "Action": [
                "ecr:DescribeRepositories",
                "ecr:GetAuthorizationToken",
                "ecr-public:DescribeRepositories",
                "ecr-public:GetAuthorizationToken"
            ],
        "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ecr:GetAuthorizationToken",
                "ecr:InitiateLayerUpload",
                "ecr:UploadLayerPart",
                "ecr:CompleteLayerUpload",
                "ecr:PutImage",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchCheckLayerAvailability"
            ],
            "Resource": "PrivateECR_Resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ecr-public:GetAuthorizationToken",
                "ecr-public:DescribeRepositories",
                "ecr-public:InitiateLayerUpload",
                "ecr-public:UploadLayerPart",
                "ecr-public:CompleteLayerUpload",
                "ecr-public:PutImage",
                "ecr-public:BatchCheckLayerAvailability",
                "sts:GetServiceBearerToken"
            ],
```

```
            "Resource": "PublicECR_Resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sts:GetServiceBearerToken"
            ],
            "Resource": "*"
        }
    ]
}
```

さらに、Commands アクションにまだ追加されていない場合は、CloudWatch ログを表示するために、サービスロールに次のアクセス許可を追加します。

```
{
    "Effect": "Allow",
    "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
    ],
    "Resource": "resource_ARN"
},
```

> **Note**
>
> サービスロールポリシーステートメントでリソースベースのアクセス許可を使用して、アクセス許可の範囲をパイプラインリソースレベルに絞り込みます。

このアクションの詳細については、「ECRBuildAndPublish ビルドアクションリファレンス」を参照してください。

アクションの宣言

YAML

```
name: ECRBuild
actionTypeId:
  category: Build
```

```
  owner: AWS
  provider: ECRBuildAndPublish
  version: '1'
runOrder: 1
configuration:
  ECRRepositoryName: actions/my-imagerepo
outputArtifacts: []
inputArtifacts:
- name: SourceArtifact
region: us-east-1
namespace: BuildVariables
```

JSON

```
{
    "name": "ECRBuild",
    "actionTypeId": {
        "category": "Build",
        "owner": "AWS",
        "provider": "ECRBuildAndPublish",
        "version": "1"
    },
    "runOrder": 1,
    "configuration": {
        "ECRRepositoryName": "actions/my-imagerepo"
    },
    "outputArtifacts": [],
    "inputArtifacts": [
        {
            "name": "SourceArtifact"
        }
    ],
    "region": "us-east-1",
    "namespace": "BuildVariables"
},
```

関連情報

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- チュートリアル: CodePipeline (V2 タイプ) を使用して Docker イメージを構築し Amazon ECR にプッシュする – このチュートリアルでは、サンプル Dockerfile と、ソースリポジトリへの変更時

にイメージを ECR にプッシュし、Amazon ECS にデプロイするパイプラインを作成する手順について説明します。

Amazon ECS および CodeDeploy ブルー/グリーンデプロイアクションリファレンス

Blue/Green デプロイを使用してコンテナアプリケーションをデプロイ AWS CodePipeline するパイプラインを で設定できます。Blue/Green デプロイでは、古いバージョンと一緒に新しいバージョンのアプリケーションを起動し、トラフィックを再ルーティングする前に新しいバージョンをテストできます。また、デプロイプロセスをモニタリングし、問題がある場合は迅速にロールバックすることもできます。

完成したパイプラインは、イメージまたはタスク定義ファイルへの変更を検出し、CodeDeploy を使用してトラフィックを Amazon ECS クラスターとロードバランサーにルーティングおよびデプロイします。CodeDeploy は、特別なポートを介して新しいタスクをターゲットにできる新しいリスナーをロードバランサーに作成します。また、Amazon ECS タスク定義が保存されている CodeCommit リポジトリなどのソース場所を使用するようにパイプラインを設定することもできます。

パイプラインを作成する前に、Amazon ECS リソース、CodeDeploy リソース、ロードバランサーとターゲットグループをすでに作成しておく必要があります。イメージリポジトリにイメージにタグを付けて保存し、タスク定義と AppSpec ファイルをファイルリポジトリにアップロードしておく必要があります。

> **Note**
>
> このトピックでは、CodePipeline の Amazon ECS から CodeDeploy Blue/Green デプロイアクションについて説明します。CodePipeline での Amazon ECS 標準デプロイアクションの詳細については、「Amazon Elastic Container Service デプロイアクションリファレンス」を参照してください。

トピック

- アクションタイプ
- 設定パラメータ
- 入力アーティファクト
- 出力アーティファクト

- サービスロールのアクセス許可: CodeDeployToECS アクション
- アクションの宣言
- 関連情報

アクションタイプ

- カテゴリ:`Deploy`
- 所有者: `AWS`
- プロバイダー: `CodeDeployToECS`
- バージョン: `1`

設定パラメータ

ApplicationName

必須: はい

CodeDeploy 内のアプリケーションの名前。パイプラインを作成する前に、CodeDeploy でアプリケーションをすでに作成しておく必要があります。

DeploymentGroupName

必須: はい

CodeDeploy アプリケーション用に作成した Amazon ECS タスクセットに指定されているデプロイグループ。パイプラインを作成する前に、CodeDeploy でデプロイグループをすでに作成しておく必要があります。

TaskDefinitionTemplateArtifact

必須: はい

デプロイアクションにタスク定義ファイルを提供する入力アーティファクトの名前。これは通常、ソースアクションからの出力アーティファクトの名前です。コンソールを使用する場合、ソースアクションの出力アーティファクトのデフォルト名は `SourceArtifact` になります。

AppSpectemplateArtifact

必須: はい

デプロイアクションに AppSpec ファイルを提供する入力アーティファクトの名前。この値は、パイプラインの実行時に更新されます。これは通常、ソースアクションからの出力アーティファクトの名前です。コンソールを使用する場合、ソースアクションの出力アーティファクトのデフォルト名は SourceArtifact になります。AppSpec ファイル内の TaskDefinition では、こちらに示されているように <TASK_DEFINITION> プレースホルダーテキストを使用できます。

AppSpecTemplatePath

必須: いいえ

パイプラインの CodeCommit リポジトリなど、パイプラインのソースファイルの口ケーションに保存されている AppSpec ファイルのファイル名。デフォルトのファイル名は appspec.yaml です。AppSpec ファイルが同じ名前で、ファイルリポジトリのルートレベルに保存されている場合は、ファイル名を指定する必要はありません。パスがデフォルトでない場合は、パスとファイル名を入力します。

TaskDefinitionTemplatePath

必須: いいえ

パイプラインの CodeCommit リポジトリなど、パイプラインのファイルソースの口ケーションに保存されているタスク定義のファイル名。デフォルトのファイル名は taskdef.json です。タスク定義ファイルが同じ名前で、ファイルリポジトリのルートレベルに保存されている場合は、ファイル名を指定する必要はありません。パスがデフォルトでない場合は、パスとファイル名を入力します。

イメージ<Number>ArtifactName

必須: いいえ

デプロイアクションにイメージを提供する入力アーティファクトの名前。これは一般に、Amazon ECR ソースアクションからの出力など、イメージリポジトリの出力アーティファクトです。

<Number> に指定できる値は 1 から 4 までです。

イメージ<Number>ContainerName

必須: いいえ

Amazon ECR ソースリポジトリなど、イメージリポジトリから使用可能なイメージの名前。

<Number> に指定できる値は 1 から 4 までです。

入力アーティファクト

- アーティファクトの数: 1 to 5
- 説明: CodeDeployToECSアクションは、最初にソースファイルリポジトリ内のタスク定義ファイルと AppSpec ファイルを検索し、次にイメージリポジトリ内のイメージを検索して、タスク定義の新しいリビジョンを動的に生成し、最後に AppSpec コマンドを実行してタスクセットとコンテナをクラスターにデプロイします。

  CodeDeployToECS アクションは、イメージ URI をイメージにマップする imageDetail.json ファイルを検索します。Amazon ECR イメージリポジトリへの変更をコミットすると、ECR ソースアクションのパイプラインはそのコミット用に imageDetail.json ファイルを作成します。アクションが自動化されていないパイプラインの場合、手動で imageDetail.json ファイルを追加することもできます。imageDetail.json ファイルの詳細については、「Amazon ECS Blue/Green デプロイアクション用の imageDetail.json ファイル」を参照してください。

  CodeDeployToECS アクションは、タスク定義の新しいリビジョンを動的に生成します。このフェーズでは、このアクションがタスク定義ファイル内のプレースホルダーを、imageDetail.json ファイルから取得したイメージ URI に置き換えます。例えば、IMAGE1_NAME を Image1ContainerName パラメータとして設定する場合、タスク定義ファイルのイメージフィールドの値としてプレースホルダー <IMAGE1_NAME> を指定する必要があります。この場合、CodeDeployToECS アクションはプレースホルダー <IMAGE1_NAME> を、Image1ArtifactName として指定したアーティファクト内の imageDetail.json から取得した実際のイメージ URI に置き換えます。

  タスク定義の更新では、CodeDeploy の AppSpec.yaml ファイルに TaskDefinition プロパティを含めます。

  ```
  TaskDefinition: <TASK_DEFINITION>
  ```

  このプロパティは、新しいタスク定義が作成された後、CodeDeployToECS アクションによって更新されます。

  TaskDefinition フィールドの値として、プレースホルダーテキストは <TASK_DEFINITION> である必要があります。CodeDeployToECS アクションは、このプレースホルダーを、動的に生成されたタスク定義の実際の ARN に置き換えます。

出力アーティファクト

- アーティファクトの数: `0`
- 説明: 出力アーティファクトは、このアクションタイプには適用されません。

サービスロールのアクセス許可: **CodeDeployToECS** アクション

CodeDeployToECS アクション (Blue/Green デプロイ) では、CodeDeploy から Amazon ECS への Blue/Green デプロイアクションを持つパイプラインを作成するために必要な最低限のアクセス許可は以下のとおりです。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCodeDeployDeploymentActions",
            "Action": [
                "codedeploy:CreateDeployment",
                "codedeploy:GetDeployment"
            ],
            "Resource": [
                "arn:aws:codedeploy:*:111122223333:deploymentgroup:
[[ApplicationName]]/*"
            ],
            "Effect": "Allow"
        },
        {
            "Sid": "AllowCodeDeployApplicationActions",
            "Action": [
                "codedeploy:GetApplication",
                "codedeploy:GetApplicationRevision",
                "codedeploy:RegisterApplicationRevision"
            ],
            "Resource": [
                "arn:aws:codedeploy:*:111122223333:application:
[[ApplicationName]]",
                "arn:aws:codedeploy:*:111122223333:application:
[[ApplicationName]]/*"
```

```
            ],
            "Effect": "Allow"
        },
        {
            "Sid": "AllowCodeDeployDeploymentConfigAccess",
            "Action": [
                "codedeploy:GetDeploymentConfig"
            ],
            "Resource": [
                "arn:aws:codedeploy:*:111122223333:deploymentconfig:*"
            ],
            "Effect": "Allow"
        },
        {
            "Sid": "AllowECSRegisterTaskDefinition",
            "Action": [
                "ecs:RegisterTaskDefinition"
            ],
            "Resource": [
                "*"
            ],
            "Effect": "Allow"
        },
        {
            "Sid": "AllowPassRoleToECS",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "arn:aws:iam::111122223333:role/[[PassRoles]]"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "ecs.amazonaws.com",
                        "ecs-tasks.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

Amazon ECS でタグ付け承認の使用をオプトインできます。オプトインする場合、ecs:TagResource というアクセス許可を付与する必要があります。オプトインの方法、必要になるアクセス許可、タグ付け承認の詳細については、Amazon Elastic Container Service 開発者ガイドの「タグ付け承認のタイムライン」を参照してください。

また、タスクに IAMロールを使用するための iam:PassRole 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 ECS タスク実行 IAM ロールそしてタスクの IAM ロール」を参照してください。

上の例に示すように、ecs-tasks.amazonaws.com を iam:PassedToService 条件下のサービスのリストへ追加することができます。

アクションの宣言

YAML

```
Name: Deploy
Actions:
  - Name: Deploy
    ActionTypeId:
      Category: Deploy
      Owner: AWS
      Provider: CodeDeployToECS
      Version: '1'
    RunOrder: 1
    Configuration:
      AppSpecTemplateArtifact: SourceArtifact
      ApplicationName: ecs-cd-application
      DeploymentGroupName: ecs-deployment-group
      Image1ArtifactName: MyImage
      Image1ContainerName: IMAGE1_NAME
      TaskDefinitionTemplatePath: taskdef.json
      AppSpecTemplatePath: appspec.yaml
      TaskDefinitionTemplateArtifact: SourceArtifact
    OutputArtifacts: []
    InputArtifacts:
      - Name: SourceArtifact
      - Name: MyImage
    Region: us-west-2
    Namespace: DeployVariables
```

JSON

```
{
    "Name": "Deploy",
    "Actions": [
        {
            "Name": "Deploy",
            "ActionTypeId": {
                "Category": "Deploy",
                "Owner": "AWS",
                "Provider": "CodeDeployToECS",
                "Version": "1"
            },
            "RunOrder": 1,
            "Configuration": {
                "AppSpecTemplateArtifact": "SourceArtifact",
                "ApplicationName": "ecs-cd-application",
                "DeploymentGroupName": "ecs-deployment-group",
                "Image1ArtifactName": "MyImage",
                "Image1ContainerName": "IMAGE1_NAME",
                "TaskDefinitionTemplatePath": "taskdef.json",
                "AppSpecTemplatePath": "appspec.yaml",
                "TaskDefinitionTemplateArtifact": "SourceArtifact"
            },
            "OutputArtifacts": [],
            "InputArtifacts": [
                {
                    "Name": "SourceArtifact"
                },
                {
                    "Name": "MyImage"
                }
            ],
            "Region": "us-west-2",
            "Namespace": "DeployVariables"
        }
    ]
}
```

関連情報

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- チュートリアル: Amazon ECR ソース、ECS - CodeDeploy 間のデプロイでパイプラインを作成する - このチュートリアルでは、Blue/Green のデプロイに必要な CodeDeploy リソースと Amazon ECS リソースの作成を順を追って説明します。このチュートリアルでは、Docker イメージを Amazon ECR にプッシュし、Docker イメージ名、コンテナ名、Amazon ECS サービス名、およびロードバランサーの設定を一覧表示する Amazon ECS タスク定義を作成する方法について説明します。このチュートリアルでは、デプロイ用の AppSpec ファイルとパイプラインの作成について順を追って説明します。

  > **Note**
  >
  > このトピックとチュートリアルでは、CodePipeline の CodeDeploy および ECS Blue/Green アクションについて説明します。CodePipeline の ECS 標準アクションの詳細については、「チュートリアル: CodePipeline を使用した継続的デプロイ」を参照してください。

- AWS CodeDeploy ユーザーガイド – Blue/Green デプロイでロードバランサー、本番リスナー、ターゲットグループ、Amazon ECS アプリケーションを使用する方法については、「チュートリアル: Amazon ECS サービスをデプロイする」を参照してください。AWS CodeDeploy ユーザーガイドのこのリファレンス情報は、Amazon ECS および を使用した Blue/Green デプロイの概要を示しています AWS CodeDeploy。
- Amazon Elastic Container Service デベロッパーガイド - Docker イメージとコンテナ、ECS サービスとクラスター、および ECS タスクセットの操作については、「Amazon ECS とは」を参照してください。

Amazon Elastic Container Service デプロイアクションリファレンス

Amazon ECS アクションを使用して、Amazon ECS サービスとタスクセットをデプロイできます。Amazon ECS サービスは、Amazon ECS クラスターにデプロイされるコンテナアプリケーションです。Amazon ECS クラスターは、クラウドでコンテナアプリケーションをホストするインスタンスの集まりです。デプロイには、Amazon ECS で作成したタスク定義と、CodePipeline がイメージをデプロイするために使用するイメージ定義ファイルが必要です。

> **Important**
>
> CodePipeline の Amazon ECS 標準デプロイアクションは、Amazon ECS サービスで使用されるリビジョンに基づいて、タスク定義の独自のリビジョンを作成します。Amazon ECS サービスを更新せずにタスク定義の新しいリビジョンを作成した場合、デプロイアクションはそれらのリビジョンを無視します。

パイプラインを作成する前に、Amazon ECS リソースを作成し、イメージリポジトリにイメージをタグ付けして保存し、buildspec ファイルをファイルリポジトリにアップロードしておく必要があります。

> **Note**
>
> このリファレンスのトピックでは、CodePipeline の Amazon ECS 標準デプロイアクションについて説明します。CodePipeline における Amazon ECS から CodeDeploy の blue/green デプロイアクションのリファレンス情報については、Amazon ECS および CodeDeploy ブルー/グリーンデプロイアクションリファレンス を参照してください。

トピック

- アクションタイプ
- 設定パラメータ
- 入力アーティファクト
- 出力アーティファクト
- サービスロールのアクセス許可: Amazon ECS 標準アクション
- アクションの宣言
- 関連情報

アクションタイプ

- カテゴリ:`Deploy`
- 所有者: `AWS`
- プロバイダー: `ECS`
- バージョン: `1`

設定パラメータ

ClusterName

必須: はい

Amazon ECS 内の Amazon ECS クラスター。

ServiceName

必須: はい

Amazon ECS で作成した Amazon ECS サービス。

FileName

必須: いいえ

イメージ定義ファイルの名前は、サービスのコンテナ名およびイメージとタグを説明する JSON ファイルです。このファイルは ECS 標準デプロイに使用します。詳細については、「入力アーティファクト」および「Amazon ECS 標準デプロイアクション用の imagedefinitions.json ファイル」を参照してください。

デプロイメントタイムアウト

必須: いいえ

Amazon ECS デプロイアクションのタイムアウト (分)。タイムアウトは、このアクションの最大デフォルトタイムアウトまで設定できます。例:

```
"DeploymentTimeout": "15"
```

入力アーティファクト

- アーティファクトの数: 1
- 説明: アクションはパイプラインのソースファイルリポジトリ内の `imagedefinitions.json` ファイル。イメージ定義ドキュメントは、Amazon ECS のコンテナ名およびイメージとタグについて説明する JSON ファイルです。CodePipeline はそのファイルを使用して、Amazon ECR などのイメージリポジトリからイメージを取得します。アクションが自動化されていないパイプラインの場合、手動で `imagedefinitions.json` ファイルを追加することもできます。`imagedefinitions.json` ファイルの詳細については、「Amazon ECS 標準デプロイアクション用の imagedefinitions.json ファイル」を参照してください。

アクションには、イメージリポジトリにすでにプッシュされている既存のイメージが必要です。イメージマッピングは `imagedefinitions.json` ファイルの場合、アクションで Amazon ECR ソースをソースアクションとしてパイプラインに含める必要はありません。

出力アーティファクト

- アーティファクトの数: `0`
- 説明: 出力アーティファクトは、このアクションタイプには適用されません。

サービスロールのアクセス許可: Amazon ECS 標準アクション

Amazon ECS では、Amazon ECS デプロイアクションを使用してパイプラインを作成するために必要な最小限のアクセス権限は次のとおりです。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "TaskDefinitionPermissions",
            "Effect": "Allow",
            "Action": [
                "ecs:DescribeTaskDefinition",
                "ecs:RegisterTaskDefinition"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "ECSServicePermissions",
            "Effect": "Allow",
            "Action": [
                "ecs:DescribeServices",
                "ecs:UpdateService"
            ],
            "Resource": [
                "arn:aws:ecs:*:111122223333:service/[[clusters]]/*"
```

```
            ]
        },
        {
            "Sid": "ECSTagResource",
            "Effect": "Allow",
            "Action": [
                "ecs:TagResource"
            ],
            "Resource": [
                "arn:aws:ecs:*:111122223333:task-definition/
[[taskDefinitions]]:*"
            ],
            "Condition": {
                "StringEquals": {
                    "ecs:CreateAction": [
                        "RegisterTaskDefinition"
                    ]
                }
            }
        },
        {
            "Sid": "IamPassRolePermissions",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "arn:aws:iam::111122223333:role/[[passRoles]]"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "ecs.amazonaws.com",
                        "ecs-tasks.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

Amazon ECS でタグ付け承認の使用をオプトインできます。オプトインする場合、`ecs:TagResource` というアクセス許可を付与する必要があります。オプトインの方法、必要

になるアクセス許可、タグ付け承認の詳細については、Amazon Elastic Container Service 開発者ガイドの「タグ付け承認のタイムライン」を参照してください。

タスクに IAM ロールを使用するための `iam:PassRole` 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 ECS タスク実行 IAM ロールそしてタスクの IAM ロール」を参照してください。以下のポリシーテキストを使用します。

アクションの宣言

YAML

```
Name: DeployECS
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Provider: ECS
  Version: '1'
RunOrder: 2
Configuration:
  ClusterName: my-ecs-cluster
  ServiceName: sample-app-service
  FileName: imagedefinitions.json
  DeploymentTimeout: '15'
OutputArtifacts: []
InputArtifacts:
  - Name: my-image
```

JSON

```
{
    "Name": "DeployECS",
    "ActionTypeId": {
        "Category": "Deploy",
        "Owner": "AWS",
        "Provider": "ECS",
        "Version": "1"
    },
    "RunOrder": 2,
    "Configuration": {
        "ClusterName": "my-ecs-cluster",
        "ServiceName": "sample-app-service",
        "FileName": "imagedefinitions.json",
```

```
        "DeploymentTimeout": "15"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "my-image"
        }
    ]
},
```

関連情報

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- ECRBuildandPublish アクションを使用してイメージをプッシュし、ECS 標準アクションを使用して Amazon ECS にデプロイする方法を示すチュートリアルについては、「チュートリアル: CodePipeline (V2 タイプ) を使用して Docker イメージを構築し Amazon ECR にプッシュする」を参照してください。
- チュートリアル: CodePipeline を使用した継続的なデプロイ — このチュートリアルでは、CodeCommit などのソースファイルリポジトリに格納する Dockerfile を作成する方法を説明します。次に、チュートリアルでは、Docker イメージを構築して Amazon ECR にプッシュし、imagedefinitions.json ファイルを作成する CodeBuild buildspec ファイルを組み込む方法を示します。最後に、Amazon ECS サービスとタスク定義を作成し、Amazon ECS デプロイアクションを使用してパイプラインを作成します。

  > **Note**
  >
  > このトピックとチュートリアルでは、CodePipeline の Amazon ECS 標準デプロイアクションについて説明します。CodePipeline における Amazon ECS から CodeDeploy の blue/green デプロイアクションの情報については、チュートリアル: Amazon ECR ソース、ECS - CodeDeploy 間のデプロイでパイプラインを作成する を参照してください。

- Amazon Elastic Container Service デベロッパーガイド - Docker イメージとコンテナ、Amazon ECS サービスとクラスター、および Amazon ECS タスクセットの操作については、「Amazon ECS とは」を参照してください。

Amazon Elastic Kubernetes Service の EKS デプロイアクションリファレンス

EKSDeploy アクションを使用して、Amazon EKS サービスをデプロイできます。デプロイには、CodePipeline がイメージのデプロイに使用する Kubernetes マニフェストファイルが必要です。

パイプラインを作成する前に、Amazon EKS リソースを作成し、イメージリポジトリにイメージを保存しておく必要があります。必要に応じて、クラスターの VPC 情報を指定できます。

> **Important**
>
> このアクションでは、CodePipeline マネージド CodeBuild コンピューティングを使用して、ビルド環境でコマンドを実行します。コマンドアクションを実行すると、 AWS CodeBuildで別途料金が発生します。

> **Note**
>
> EKS デプロイアクションは V2 タイプのパイプラインでのみ使用できます。

EKS アクションは、パブリック EKS クラスターとプライベート EKS クラスターの両方をサポートします。プライベートクラスターは EKS で推奨されるタイプですが、どちらのタイプもサポートされています。

EKS アクションは、クロスアカウントアクションでサポートされています。クロスアカウント EKS アクションを追加するには、アクション宣言でターゲットアカウントから actionRoleArn を追加します。

トピック

- アクションタイプ
- 設定パラメータ
- 入力アーティファクト
- 出力アーティファクト
- 環境変数
- 出力変数

- サービスロールのポリシーのアクセス許可
- アクションの宣言
- 関連情報

アクションタイプ

- カテゴリ:Deploy
- 所有者: AWS
- プロバイダー: EKS
- バージョン: 1

設定パラメータ

ClusterName

必須: はい

Amazon EKS の Amazon EKS クラスター。

Helm のオプション

Helm が選択したデプロイツールである場合に使用できるオプションは次のとおりです。

HelmReleaseName

必須: はい (Helm タイプでのみ必須)

デプロイのリリース名。

HelmChartLocation

必須: はい (Helm タイプでのみ必須)

デプロイのチャートの場所。

HelmValuesFiles

必須: いいえ (Helm タイプでのみオプション)

Helm 値のファイルを上書きするには、Helm チャートの場所にカンマ区切りの Helm 値ファイルを入力します。

Kubectl のオプション

Kubectl が選択したデプロイツールである場合に使用できるオプションは次のとおりです。

ManifestFiles

必須: はい (Kubectl タイプでのみ必須)

マニフェストファイルの名前は、サービスのコンテナ名およびイメージとタグを説明するテキストファイルです。このファイルを使用して、イメージ URI とその他の情報をパラメータ化します。この目的のために 環境変数を使用できます。

このファイルは、パイプラインのソースリポジトリに保存します。

Namespace

必須: いいえ

`kubectl` または `helm` コマンドで使用される Kubernetes 名前空間。

サブネット

必須: いいえ

クラスターの VPC のサブネット。これらは、クラスターにアタッチされているのと同じ VPC の一部です。クラスターにまだアタッチされていないサブネットをここで指定することもできます。

SecurityGroupIds

必須: いいえ

クラスターの VPC のセキュリティグループ。これらは、クラスターにアタッチされているのと同じ VPC の一部です。クラスターにまだアタッチされていないセキュリティグループを指定して、ここで指定することもできます。

入力アーティファクト

- アーティファクトの数: 1
- 説明: アクションはパイプラインのソースファイルリポジトリ内の Kubernetes マニフェストファイルまたは Helm チャートを参照します。S3 バケットに保存されている .tgz 形式の Helm チャートを使用する場合は、S3 バケット/キーをソースアクションとして設定することで使用できます。例えば、指定されたオブジェクトキーは `my-chart-0.1.0.tgz` になります。

出力アーティファクト

- アーティファクトの数: `0`
- 説明: 出力アーティファクトは、このアクションタイプには適用されません。

環境変数

マニフェストファイルまたは Helm チャート値ファイルのイメージリポジトリやイメージタグなどの変数を置き換えるために使用されます。

キー

キーと値の環境変数ペアのキー (`$IMAGE_TAG` など)。

値

キーと値のペアの値 (`v1.0` など)。値は、パイプラインアクションまたはパイプライン変数からの出力変数でパラメータ化できます。例えば、パイプラインには `${codepipeline.PipelineExecutionId}` で ECR イメージを作成する ECRBuildAndPublish アクションがあり、EKS アクションは `${codepipeline.PipelineExecutionId}` を環境変数の値として使用してこのイメージを使用できます。

出力変数

EKSClusterName

Amazon EKS の Amazon EKS クラスター。

サービスロールのポリシーのアクセス許可

このアクションを実行するには、パイプラインのサービスロールポリシーで次のアクセス許可が使用可能である必要があります。

- EC2 アクション: CodePipeline が実行される際は、アクション EC2 インスタンスのアクセス許可が必要です。これは、EKS クラスターの作成に必要な EC2 インスタンスロールとは異なります。

  既存のサービスロールを使用している場合、このアクションを使用するには、サービスロールに以下のアクセス許可を追加する必要があります。

- ec2:CreateNetworkInterface
- ec2:DescribeDhcpOptions
- ec2:DescribeNetworkInterfaces
- ec2:DeleteNetworkInterface
- ec2:DescribeSubnets
- ec2:DescribeSecurityGroups
- ec2:DescribeVpcs

- EKS アクション: CodePipeline がアクションを実行する際は、EKS クラスターアクセス許可が必要です。これは、EKS クラスターの作成に必要な IAM EKS クラスターロールとは異なることに注意してください。

  既存のサービスロールを使用している場合、このアクションを使用するには、サービスロールに以下のアクセス許可を追加する必要があります。

  - eks:DescribeCluster
- ログストリームアクション: CodePipeline は、アクションを実行するときに、次のようにパイプライン名を使用してロググループを作成します。これにより、パイプライン名を使用してアクセス許可の範囲をリソースのログ記録に絞り込むことができます。

  ```
  /aws/codepipeline/MyPipelineName
  ```

  既存のサービスロールを使用している場合、このアクションを使用するには、サービスロールに以下のアクセス許可を追加する必要があります。

  - logs:CreateLogGroup
  - logs:CreateLogStream
  - logs:PutLogEvents

サービスロールポリシーステートメントで、次の例に示すように、アクセス許可の範囲をリソースレベルに絞り込みます。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "eks:DescribeCluster"
            ],
            "Resource": "arn:aws:eks:*:111122223333:cluster/YOUR_CLUSTER_NAME"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:DescribeDhcpOptions",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DeleteNetworkInterface",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeVpcs",
                "ec2:DescribeRouteTables"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:CreateLogGroup",
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:*:111122223333:log-group:/aws/
codepipeline/YOUR_PIPELINE_NAME",
                "arn:aws:logs:*:111122223333:log-group:/aws/
codepipeline/YOUR_PIPELINE_NAME:*"
            ]
        }
    ]
}
```

アクションの詳細ダイアログページを使用してコンソールでログを表示するには、ログを表示するアクセス許可をコンソールロールに追加する必要があります。詳細については、「コンソールでコン

ピューティングログを表示するために必要なアクセス許可」でコンソールのアクセス許可ポリシーの例を参照してください。

サービスロールをクラスターのアクセスエントリとして追加する

パイプラインのサービスロールポリシーでアクセス許可が利用可能になったら、CodePipeline サービスロールをクラスターのアクセスエントリとして追加して、クラスターのアクセス許可を設定します。

更新されたアクセス許可を持つアクションロールを使用することもできます。詳細については、「ステップ 4: CodePipeline サービスロールのアクセスエントリを作成する」のチュートリアルの例を参照してください。

アクションの宣言

YAML

```
Name: DeployEKS
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Provider: EKS
  Version: '1'
RunOrder: 2
Configuration:
  ClusterName: my-eks-cluster
  ManifestFiles: ManifestFile.json
OutputArtifacts: []
InputArtifacts:
  - Name: SourceArtifact
```

JSON

```
{
    "Name": "DeployECS",
    "ActionTypeId": {
        "Category": "Deploy",
        "Owner": "AWS",
        "Provider": "EKS",
        "Version": "1"
    },
    "RunOrder": 2,
```

```
    "Configuration": {
        "ClusterName": "my-eks-cluster",
        "ManifestFiles": "ManifestFile.json"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "SourceArtifact"
        }
    ]
},
```

関連情報

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- EKS クラスターと Kubernetes マニフェストファイルを作成してアクションをパイプラインに追加する方法を示すチュートリアルについては、「チュートリアル: CodePipeline を使用して Amazon EKS にデプロイする」を参照してください。

AWS Lambda デプロイアクションリファレンス

AWS Lambda デプロイアクションを使用して、サーバーレスデプロイ用のアプリケーションコードのデプロイを管理します。関数をデプロイし、次のようにトラフィックデプロイのデプロイ戦略を使用できます。

- トラフィックシフト用の Canary デプロイとリニアデプロイ
- All at Once デプロイ

> **Note**
>
> このアクションは V2 タイプのパイプラインでのみサポートされています。

トピック

- アクションタイプ
- 設定パラメータ

- 入力アーティファクト
- 出力アーティファクト
- 出力変数
- Lambda デプロイアクションのサービスロールポリシーのアクセス許可
- アクションの宣言
- 関連情報

アクションタイプ

- カテゴリ:Deploy
- 所有者: AWS
- プロバイダー: Lambda
- バージョン: 1

設定パラメータ

FunctionName

必須: はい

Lambda で作成した関数の名前 (MyLambdaFunction など)。

バージョンを作成しておく必要があります。

FunctionAlias

必須: いいえ

Lambda で作成した関数のエイリアス (live など)。この関数をデプロイします。エイリアスが存在し、アクション実行の開始時にその背後にあるバージョンが 1 つある必要があります。(ロールバックターゲットバージョンになります)。

指定しない場合、アクションはソースアーティファクトを $LATEST にデプロイし、新しいバージョンを作成します。このユースケースでは、デプロイ戦略とターゲットバージョンオプションは使用できません。

PublishedTargetVersion

必須: いいえ

FunctionAlias にデプロイする Lambda 関数のバージョン。パイプラインまたはアクションレベルの変数を使用できます (#{variables.lambdaTargetVersion} など)。アクション実行の開始時に、バージョンを公開する必要があります。

入力アーティファクトが指定されていない場合は必須です。

DeployStrategy

必須: いいえ (デフォルトは AllAtOnce)

Lambda デプロイアクションが、Lambda 関数の元のバージョンから FunctionAlias の新しいバージョンにトラフィックを移行するために採用するレートを決定します。使用可能なデプロイ戦略は Canary またはリニアです。使用できる形式:

- AllAtOnce -

  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。

  指定されなかった場合、デフォルト値は AllAtOnce です。
- Canary10Percent5Minutes - 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。

  パーセンテージと分の値の両方を変更できます。
- Linear10PercentEvery1Minute - 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。

  パーセンテージと分の値の両方を変更できます。

このフィールドについては、次の考慮事項に注意してください。

- 最大合計待機時間は 2 日です。
- FunctionAlias が指定されている場合にのみ使用できます。

アラーム

必須: いいえ

Lambda デプロイ用に設定されたアラーム名のカンマ区切りリスト。最大 10 のアラームを追加できます。モニタリングされたアラームが ALARM 状態になると、アクションは失敗します。

次の画像は、アクションの [編集] ページの例を示しています。

Edit action

Action name
Choose a name for your action
Deploy
No more than 100 characters

Action provider
AWS Lambda

Region
United States (N. Virginia)

Input artifacts
Choose an input artifact for this action. Learn more

SourceArtifact
Defined by: Source
No more than 100 characters

Function name
Choose a function that you have already created in the AWS Lambda console. Or create a function in the AWS Lambda console and then return to this task.
MyLambdaFunction

Function alias
Choose an alias of the Lambda Function to be deployed to, such as live. The alias must exist and have a version behind it when the action executions starts (it will be the rollback target version). If not provided, action deploys source artifact to $LATEST and creates a new version. In this use case, deploy strategy and target version options are not available.
aliasV1

Deploy strategy
Choose the rate at which the Lambda deploy action adopts to shift traffic from the original version of the Lambda function to the new version.

Deployment strategy type
Choose the deployment type for the Lambda deploy action.
Canary deployment

Traffic Percentage
Percentage of traffic to shift in each increment (1-100)
10

Time interval (minutes)
Time between traffic shifts
5

Alarms
A list of alarm names configured for the lambda deployment. A maximum of 10 alarms can be added. The Lambda deploy action fails when any monitored alarm goes into alarm state.
CloudWatch Alarms

Cancel Done

入力アーティファクト

- アーティファクトの数: 1
- 説明: デプロイ中にスクリプトアクションをサポートするために提供されたファイル。

出力アーティファクト

- アーティファクトの数: 0
- 説明: 出力アーティファクトは、このアクションタイプには適用されません。

出力変数

このアクションを設定すると、パイプライン内のダウンストリームアクションのアクション設定によって参照できる変数が生成されます。このアクションは、アクションに名前空間がない場合でも、出力変数として表示できる変数を生成します。名前空間を使用してアクションを設定し、これらの変数をダウンストリームアクションの設定で使用できるようにします。

詳細については、「変数リファレンス」を参照してください。

FunctionVersion

デプロイされた新しい Lambda 関数のバージョン。

Lambda デプロイアクションのサービスロールポリシーのアクセス許可

CodePipeline がアクションを実行する際、CodePipeline サービスロールには、最小特権でのアクセスに適切にスコープダウンされた次のアクセス許可が必要です。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "StatementForLambda",
            "Effect": "Allow",
            "Action": [
                "lambda:GetAlias",
```

```
                "lambda:GetFunctionConfiguration",
                "lambda:GetProvisionedConcurrencyConfig",
                "lambda:PublishVersion",
                "lambda:UpdateAlias",
                "lambda:UpdateFunctionCode"
            ],
            "Resource": [
                "arn:aws:lambda:us-east-1:111122223333:function:
{{FunctionName}}",
                "arn:aws:lambda:us-east-1:111122223333:function:
{{FunctionName}}:*"
            ]
        },
        {
            "Sid": "StatementForCloudWatch",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:DescribeAlarms"
            ],
            "Resource": [
                "arn:aws:cloudwatch:us-east-1:111122223333:alarm:{{AlarmNames}}"
            ]
        },
        {
            "Sid": "StatementForLogs1",
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:111122223333:log-group:/us-east-1/
codepipeline/{{pipelineName}}",
                "arn:aws:logs:us-east-1:111122223333:log-group:/us-east-1/
codepipeline/{{pipelineName}}:*"
            ]
        },
        {
            "Sid": "StatementForLogs2",
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": [
```

```
                "arn:aws:logs:us-east-1:111122223333:log-group:/us-east-1/
codepipeline/{{pipelineName}}:log-stream:*"
            ]
        }
    ]
}
```

アクションの宣言

YAML

```
name: Deploy
actionTypeId:
  category: Deploy
  owner: AWS
  provider: Lambda
  version: '1'
runOrder: 1
configuration:
  DeployStrategy: Canary10Percent5Minutes
  FunctionAlias: aliasV1
  FunctionName: MyLambdaFunction
outputArtifacts: []
inputArtifacts:
- name: SourceArtifact
region: us-east-1
namespace: DeployVariables
```

JSON

```
{
    "name": "Deploy",
    "actionTypeId": {
        "category": "Deploy",
        "owner": "AWS",
        "provider": "Lambda",
        "version": "1"
    },
    "runOrder": 1,
    "configuration": {
        "DeployStrategy": "Canary10Percent5Minutes",
        "FunctionAlias": "aliasV1",
```

```
        "FunctionName": "MyLambdaFunction"
    },
    "outputArtifacts": [],
    "inputArtifacts": [
        {
            "name": "SourceArtifact"
        }
    ],
    "region": "us-east-1",
    "namespace": "DeployVariables"
},
```

関連情報

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- チュートリアル: CodePipeline を使用した Lambda 関数のデプロイ – このチュートリアルでは、サンプル Lambda 関数の作成について説明します。ここでは、エイリアスとバージョンを作成し、圧縮された Lambda 関数をソースの場所に追加し、パイプラインで Lambda アクションを実行します。

Amazon S3 デプロイアクションリファレンス

Amazon S3 デプロイアクションを使用して、静的ウェブサイトホスティングまたはアーカイブ用にファイルを Amazon S3 バケットにデプロイします。バケットにアップロードする前にデプロイファイルを抽出するかどうかを指定できます。

> **Note**
>
> このリファレンストピックでは、CodePipeline の Amazon S3 デプロイアクションについて説明します。デプロイプラットフォームはホスティング用に設定された Amazon S3 バケットです。CodePipeline での Amazon S3 ソースアクションのリファレンス情報については、「Amazon S3 ソースアクションリファレンス」を参照してください。

トピック

- アクションタイプ
- 設定パラメータ

- 入力アーティファクト
- 出力アーティファクト
- サービスロールのアクセス許可: S3 デプロイアクション
- アクション設定の例
- 関連情報

アクションタイプ

- カテゴリ:Deploy
- 所有者: AWS
- プロバイダー: S3
- バージョン: 1

設定パラメータ

BucketName

必須: はい

ファイルがデプロイされる Amazon S3 バケットの名前。

Extract

必須: はい

true を指定すると、アップロード前にファイルが抽出されるようになります。それ以外の場合、ホストされた静的ウェブサイトの場合など、アプリケーションファイルはアップロード用に圧縮されたままになります。false を指定した場合は、ObjectKey が必要になります。

ObjectKey

条件付き。Extract = false の場合は必須

S3 バケット内のオブジェクトを一意に識別する Amazon S3 オブジェクトキーの名前。

KMSEncryptionKeyARN

必須: いいえ

ホストバケットの AWS KMS 暗号化キーの ARN。KMSEncryptionKeyARN パラメータは、提供された AWS KMS keyを使用してアップロードされたアーティファクトを暗号化します。 KMS キーの場合、キー ID 、キー ARN 、またはエイリアス ARN を使用できます。

> **Note**
>
> エイリアスは、カスタマーマスターキー (KMS) を作成したアカウントでのみ認識されます。クロスアカウントアクションの場合、キー ID またはキー ARN のみを使用してキーを識別できます。クロスアカウントアクションには他のアカウント (AccountB) のロールを使用するため、キー ID を指定すると他のアカウント (AccountB) のキーが使用されます。

> **Important**
>
> CodePipeline は、対称 KMS キーのみをサポートしています。非対称キーを使用して S3 bucket のデータを暗号化しないでください。

CannedACL

必須: いいえ

CannedACL パラメータは、指定された 既定 ACL を、Amazon S3 にデプロイされたオブジェクトに適用します。このオブジェクトに適用された既存の ACL が上書きされます。

CacheControl

必須: いいえ

CacheControl パラメータは、バケットのオブジェクトのリクエストやレスポンスのキャッシュ動作を制御します。有効な値のリストについては、HTTP オペレーションの Cache-Control ヘッダーフィールドを参照してください。CacheControl に複数の値を入力するには、各値の間にカンマを使用します。CLI の例に示すように、各カンマの後にスペースを追加できます (オプション)。

```
"CacheControl": "public, max-age=0, no-transform"
```

入力アーティファクト

- アーティファクトの数: 1
- 説明: デプロイまたはアーカイブ用のファイルは、CodePipeline によってソースリポジトリから取得され、圧縮されて、アップロードされます。

出力アーティファクト

- アーティファクトの数: 0
- 説明: 出力アーティファクトは、このアクションタイプには適用されません。

サービスロールのアクセス許可: S3 デプロイアクション

S3 デプロイアクションをサポートするには、以下をポリシーステートメントに追加します。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:PutObjectAcl",
        "s3:PutObjectVersionAcl",
        "s3:GetBucketVersioning",
        "s3:GetBucketAcl",
        "s3:GetBucketLocation"
      ],
      "Resource": [
        "arn:aws:s3:::[[s3DeployBuckets]]",
        "arn:aws:s3:::[[s3DeployBuckets]]/*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "111122223333"
        }
      }
```

```
    }
  ]
}
```

S3 デプロイアクションのサポートでは、S3 オブジェクトにタグがある場合は、ポリシーステートメントに次のアクセス許可も追加する必要があります。

```
"s3:GetObjectTagging",
"s3:GetObjectVersionTagging",
"s3:PutObjectTagging"
```

アクション設定の例

次に、アクションの設定例を示します。

Extract を **false** に設定する場合のアクションの設定例

以下の例で示しているのは、Extract フィールドを false に設定してアクションを作成した場合のアクションのデフォルト設定です。

YAML

```
Name: Deploy
Actions:
  - Name: Deploy
    ActionTypeId:
      Category: Deploy
      Owner: AWS
      Provider: S3
      Version: '1'
    RunOrder: 1
    Configuration:
      BucketName: website-bucket
      Extract: 'false'
      ObjectKey: MyWebsite
    OutputArtifacts: []
    InputArtifacts:
      - Name: SourceArtifact
    Region: us-west-2
    Namespace: DeployVariables
```

JSON

```
{
    "Name": "Deploy",
    "Actions": [
        {
            "Name": "Deploy",
            "ActionTypeId": {
                "Category": "Deploy",
                "Owner": "AWS",
                "Provider": "S3",
                "Version": "1"
            },
            "RunOrder": 1,
            "Configuration": {
                "BucketName": "website-bucket",
                "Extract": "false",
                "ObjectKey": "MyWebsite"
                },
            "OutputArtifacts": [],
            "InputArtifacts": [
                {
                    "Name": "SourceArtifact"
                }
            ],
            "Region": "us-west-2",
            "Namespace": "DeployVariables"
        }
    ]
},
```

Extract を **true** に設定する場合のアクションの設定例

以下の例で示しているのは、Extract フィールドを true に設定してアクションを作成した場合のアクションのデフォルト設定です。

YAML

```
Name: Deploy
Actions:
  - Name: Deploy
    ActionTypeId:
```

```
      Category: Deploy
      Owner: AWS
      Provider: S3
      Version: '1'
    RunOrder: 1
    Configuration:
      BucketName: website-bucket
      Extract: 'true'
    OutputArtifacts: []
    InputArtifacts:
      - Name: SourceArtifact
    Region: us-west-2
    Namespace: DeployVariables
```

JSON

```
{
    "Name": "Deploy",
    "Actions": [
        {
            "Name": "Deploy",
            "ActionTypeId": {
                "Category": "Deploy",
                "Owner": "AWS",
                "Provider": "S3",
                "Version": "1"
            },
            "RunOrder": 1,
            "Configuration": {
                "BucketName": "website-bucket",
                "Extract": "true"
                },
            "OutputArtifacts": [],
            "InputArtifacts": [
                {
                    "Name": "SourceArtifact"
                }
            ],
            "Region": "us-west-2",
            "Namespace": "DeployVariables"
        }
    ]
},
```

関連情報

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- チュートリアル: Amazon S3 をデプロイプロバイダとして使用するパイプラインを作成する - このチュートリアルでは、S3 デプロイアクションを使用してパイプラインを作成する 2 つの例を紹介します。サンプルファイルをダウンロードし、CodeCommit リポジトリにアップロードします。その後、S3 バケットを作成し、ホスティング用に設定します。次に、CodePipeline コンソールを使用してパイプラインを作成し、Amazon S3 デプロイ設定を指定します。
- Amazon S3 ソースアクションリファレンス - このアクションリファレンスでは、CodePipeline での Amazon S3 ソースアクションのリファレンス情報と例を提供しています。

Amazon S3 ソースアクションリファレンス

ソースアクションは、新しいオブジェクトが設定されたバケットとオブジェクトキーにアップロードされると、パイプラインをトリガーします。

このリファレンストピックでは、CodePipeline の Amazon S3 ソースアクションについて説明します。ソース場所は、バージョニング用に設定された Amazon S3 バケットです。CodePipeline での Amazon S3 デプロイアクションのリファレンス情報については、「Amazon S3 デプロイアクションリファレンス」を参照してください。

パイプラインを接続する前に、Amazon S3 ソースバケットを作成し、バージョニングを有効にして、ソースファイルを 1 つの ZIP ファイルとしてアップロードします。

Amazon S3 で新しいバケットを作成するか、アプリケーションファイルのソースの場所として使用する既存の Amazon S3 バケットを選択できます。バージョニングの有効化の詳細については、Amazon S3 ユーザーガイド」の「バージョニングの使用」を参照してください。

Amazon S3 がパイプラインのソースプロバイダーである場合は、ソースファイルを 1 つの .zip に圧縮し、.zip をソースバケットにアップロードできます。解凍した 1 つのファイルをアップロードすることもできますが、.zip ファイルを想定したダウンストリームアクションは失敗します。

コンソールを使用してパイプラインを作成または編集する場合、CodePipeline は Amazon S3 ソースバケットで変更が発生したときにパイプラインを開始する EventBridge ルールを作成します。

入力変換エントリを使用してソースオーバーライドを作成し、パイプラインイベントの EventBridge で `revisionValue` を使用することもできます。ここで、`revisionValue` はオブジェクトキー、コミット、またはイメージ ID のソースイベント変数から派生します。詳細については、「」の手順

に含まれる入力変換エントリのオプションステップを参照してください[イベントに対して有効にしたソースを使用して Amazon S3 ソースアクションに接続する]。

トピック

- アクションタイプ
- 設定パラメータ
- 入力アーティファクト
- 出力アーティファクト
- 出力変数
- サービスロールのアクセス許可: S3 ソースアクション
- アクションの宣言
- 関連情報

アクションタイプ

- カテゴリ:`Source`
- 所有者: `AWS`
- プロバイダー: `S3`
- バージョン: `1`

設定パラメータ

S3 バケット

必須: はい

ソースの変更が検出される Amazon S3 バケットの名前。

S3ObjectKey

必須: はい

ソースの変更が検出される Amazon S3 オブジェクトキーの名前。

AllowOverrideForS3ObjectKey

必須: いいえ

AllowOverrideForS3ObjectKey は、StartPipelineExecution からのソースの上書きにより、ソースアクションに設定済みの S3ObjectKey を上書きできるかどうかを制御します。S3 オブジェクトキーを使用したソースの上書きの詳細については、「ソースリビジョンオーバーライドでパイプラインを開始する」を参照してください。

> **Important**
>
> AllowOverrideForS3ObjectKey を省略すると、CodePipeline は、このパラメータを false に設定して、ソースアクションで S3 オブジェクトキーを上書きできないようにします。

このパラメータの有効な値:

- true: 設定すると、事前設定済みの S3 オブジェクトキーは、パイプラインの実行中にソースリビジョンの上書きによって上書きされる場合があります。

  > **Note**
  >
  > 新しいパイプラインの実行開始時に、事前設定済みの S3 オブジェクトキーを上書きすることをすべての CodePipeline ユーザーに許可する場合は、AllowOverrideForS3ObjectKey を true に設定する必要があります。

- false:

  設定すると、CodePipeline は、ソースリビジョンの上書きを使用して S3 オブジェクトキーを上書きすることを許可しません。これは、このパラメータのデフォルト値でもあります。

PollForSourceChanges

必須: いいえ

PollForSourceChanges は、ソースの変更について CodePipeline が Amazon S3 ソースバケットをポーリングするかどうかを制御します。代わりに CloudWatch Events と CloudTrail を使用してソースの変更を検出することをお勧めします。CloudWatch Events の設定についての詳細は、「S3 ソースと CloudTrail 証跡を使用してポーリングパイプラインを移行する (CLI)」または「S3 ソースと CloudTrail 証跡 (CloudFormation テンプレート) を使用してポーリングパイプラインを移行する」を参照してください。

> **Important**
>
> CloudWatch Events ルールを設定する場合、パイプラインの重複実行を避けるために `PollForSourceChanges` を `false` に設定する必要があります。

このパラメータの有効な値:

- `true`: 設定されている場合、CodePipeline はソースの変更についてソースの場所をポーリングします。

  > **Note**
  >
  > `PollForSourceChanges` を省略すると、 CodePipeline はソースの変更についてデフォルトでソースの場所をポーリングします。この動作は、`PollForSourceChanges` が含まれており、`true` に設定されている場合と同じです。

- `false`: 設定されている場合、CodePipeline はソースの変更についてソースの場所をポーリングしません。CloudWatch Events ルールを設定してソース変更を検出する場合は、この設定を使用します。

入力アーティファクト

- アーティファクトの数: `0`
- 説明: 入力アーティファクトは、このアクションタイプには適用されません。

出力アーティファクト

- アーティファクトの数: `1`
- 説明: パイプラインに接続するように設定されたソースバケットで使用可能なアーティファクトを提供します。バケットから生成されたアーティファクトは、Amazon S3 アクションの出力アーティファクトです。Amazon S3 オブジェクトメタデータ (ETag とバージョン ID) が、トリガーされたパイプライン実行のソースリビジョンとして CodePipeline で表示されます。

出力変数

このアクションを設定すると、パイプライン内のダウンストリームアクションのアクション設定によって参照できる変数が生成されます。このアクションは、アクションに名前空間がない場合でも、出力変数として表示できる変数を生成します。名前空間を使用してアクションを設定し、これらの変数をダウンストリームアクションの設定で使用できるようにします。

CodePipeline における変数の詳細については、変数リファレンス を参照してください。

BucketName

パイプラインをトリガーしたソース変更に関連する Amazon S3 バケットの名前。

ETag

パイプラインをトリガーしたソース変更に関連するオブジェクトのエンティティタグ。ETag は、オブジェクトの MD5 ハッシュです。ETag は、オブジェクトのコンテンツに加えた変更のみを反映し、メタデータに加えた変更は反映しません。

ObjectKey

パイプラインをトリガーしたソース変更に関連する Amazon S3 オブジェクトキーの名前。

VersionId

パイプラインをトリガーしたソース変更に関連するオブジェクトのバージョンのバージョン ID。

サービスロールのアクセス許可: S3 ソースアクション

S3 ソースアクションをサポートするには、以下をポリシーステートメントに追加します。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion",
        "s3:GetBucketVersioning",
        "s3:GetBucketAcl",
```

```
        "s3:GetBucketLocation",
        "s3:GetObjectTagging",
        "s3:GetObjectVersionTagging"
      ],
      "Resource": [
        "arn:aws:s3:::[[S3Bucket]]",
        "arn:aws:s3:::[[S3Bucket]]/*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "111122223333"
        }
      }
    }
  ]
}
```

アクションの宣言

YAML

```
Name: Source
Actions:
  - RunOrder: 1
    OutputArtifacts:
      - Name: SourceArtifact
    ActionTypeId:
      Provider: S3
      Owner: AWS
      Version: '1'
      Category: Source
    Region: us-west-2
    Name: Source
    Configuration:
      S3Bucket: amzn-s3-demo-source-bucket
      S3ObjectKey: my-application.zip
      PollForSourceChanges: 'false'
    InputArtifacts: []
```

JSON

```
{
```

```
    "Name": "Source",
    "Actions": [
        {
            "RunOrder": 1,
            "OutputArtifacts": [
                {
                    "Name": "SourceArtifact"
                }
            ],
            "ActionTypeId": {
                "Provider": "S3",
                "Owner": "AWS",
                "Version": "1",
                "Category": "Source"
            },
            "Region": "us-west-2",
            "Name": "Source",
            "Configuration": {
                "S3Bucket": "amzn-s3-demo-source-bucket",
                "S3ObjectKey": "my-application.zip",
                "PollForSourceChanges": "false"
            },
            "InputArtifacts": []
        }
    ]
},
```

関連情報

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- チュートリアル: シンプルなパイプラインを作成する (S3 バケット) – このチュートリアルでは、サンプル AppSpec ファイルとサンプル CodeDeploy アプリケーションおよびデプロイグループを提供します。このチュートリアルを参照して、Amazon EC 2 インスタンスにデプロイする Amazon S3 ソースを持つパイプラインを作成します。

AWS AppConfig デプロイアクションリファレンス

AWS AppConfig は の一機能です AWS Systems Manager。 AppConfig は、あらゆる規模のアプリケーションへの制御されたデプロイをサポートし、検証チェックとモニタリングの機能が組み込ま

れています。AppConfig は、Amazon EC2 インスタンス、コンテナ AWS Lambda、モバイルアプリケーション、または IoT デバイスでホストされているアプリケーションで使用できます。

AppConfig デプロイアクションは、パイプラインソースの場所に保存されている設定を、指定された AppConfig アプリケーション、環境、および設定プロファイルにデプロイする AWS CodePipeline アクションです。AppConfig デプロイ戦略 で定義されている環境設定を使用します。

アクションタイプ

- カテゴリ:Deploy
- 所有者: AWS
- プロバイダー: AppConfig
- バージョン: 1

設定パラメータ

アプリケーション

必須: はい

設定とデプロイの詳細を含む AWS AppConfig アプリケーションの ID。

環境

必須: はい

設定がデプロイされている AWS AppConfig 環境の ID。

設定プロファイル

必須: はい

デプロイする AWS AppConfig 設定プロファイルの ID。

Artifact 設定パスを入力

必須: はい

デプロイする入力アーティファクト内の構成データのファイルパス。

デプロイメントストラテジー

必須: いいえ

デプロイに使用する AWS AppConfig デプロイ戦略。

入力アーティファクト

- アーティファクトの数: 1
- 説明:デプロイアクションの入力アーティファクト。

出力アーティファクト

該当なし。

サービスロールのアクセス許可: **AppConfig** アクション

CodePipeline がアクションを実行する際、CodePipeline サービスロールポリシーには、最小特権でアクセスを維持するために、リソースレベルに適切にスコープダウンされた次のアクセス許可が必要です。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "appconfig:StartDeployment",
                "appconfig:StopDeployment",
                "appconfig:GetDeployment"
            ],
            "Resource": [
                "arn:aws:appconfig:*:111122223333:application/[[Application]]",
                "arn:aws:appconfig:*:111122223333:application/[[Application]]/*",
                "arn:aws:appconfig:*:111122223333:deploymentstrategy/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

アクション設定の例

YAML

```
name: Deploy
actions:
  - name: Deploy
    actionTypeId:
      category: Deploy
      owner: AWS
      provider: AppConfig
      version: '1'
    runOrder: 1
    configuration:
      Application: 2s2qv57
      ConfigurationProfile: PvjrpU
      DeploymentStrategy: frqt7ir
      Environment: 9tm27yd
      InputArtifactConfigurationPath: /
    outputArtifacts: []
    inputArtifacts:
      - name: SourceArtifact
    region: us-west-2
    namespace: DeployVariables
```

JSON

```
{
    "name": "Deploy",
    "actions": [
        {
            "name": "Deploy",
            "actionTypeId": {
                "category": "Deploy",
                "owner": "AWS",
                "provider": "AppConfig",
                "version": "1"
            },
            "runOrder": 1,
            "configuration": {
                "Application": "2s2qv57",
                "ConfigurationProfile": "PvjrpU",
                "DeploymentStrategy": "frqt7ir",
```

```
                "Environment": "9tm27yd",
                "InputArtifactConfigurationPath": "/"
            },
            "outputArtifacts": [],
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-west-2",
            "namespace": "DeployVariables"
        }
    ]
}
```

関連情報

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- AWS AppConfig – AWS AppConfig デプロイの詳細については、AWS Systems Manager 「ユーザーガイド」を参照してください。
- チュートリアル: AWS AppConfig をデプロイプロバイダーとして使用するパイプラインを作成する – このチュートリアルでは、シンプルなデプロイ設定ファイルと AppConfig リソースの設定を開始し、コンソールを使用して AWS AppConfig デプロイアクションでパイプラインを作成する方法を示します。

CloudFormation デプロイアクションリファレンス

CloudFormation スタックに対して オペレーションを実行します。スタックは、単一のユニットとして管理できる AWS リソースのコレクションです。スタック内のすべてのリソースは、スタックの CloudFormation テンプレートで定義されます。変更セットにより、元のスタックを変更せずに表示できる比較が作成されます。スタックと変更セットで実行できる CloudFormation アクションのタイプについては、 `ActionMode`パラメータを参照してください。

スタックオペレーションが失敗した CloudFormation アクションのエラーメッセージを作成するには、CodePipeline は API を CloudFormation `DescribeStackEvents`呼び出します。アクション IAM ロールにその API にアクセスするアクセス許可がある場合、最初に失敗したリソースの詳細が CodePipeline エラーメッセージに含まれます。そうでなく、ロールポリシーに適切なアク

セス許可がない場合、CodePipeline は API へのアクセスを無視し、代わりに一般的なエラーメッセージを表示します。そのためには、パイプラインのサービスロールまたは他の IAM ロールに `cloudformation:DescribeStackEvents` アクセス許可を追加する必要があります。

リソースの詳細がパイプラインのエラーメッセージに表示されないようにするには、`cloudformation:DescribeStackEvents` アクセス許可を削除することによって、アクション IAM ロールに対するこのアクセス許可を取り消すことができます。

トピック

- アクションタイプ
- 設定パラメータ
- 入力アーティファクト
- 出力アーティファクト
- 出力変数
- サービスロールのアクセス許可: CloudFormation アクション
- アクションの宣言
- 関連情報

アクションタイプ

- カテゴリ:`Deploy`
- 所有者: `AWS`
- プロバイダー: `CloudFormation`
- バージョン: `1`

設定パラメータ

ActionMode

必須: はい

`ActionMode` は、スタックまたは変更セットで CloudFormation 実行されるアクションの名前です。以下のアクティベーションモードを使用できます。

- CHANGE_SET_EXECUTE は、指定された一連のリソース更新に基づくリソーススタックの変更セットを実行します。このアクションにより、 はスタックの変更 CloudFormation を開始します。
- CHANGE_SET_REPLACE は、変更セットが存在しない場合、指定されたスタック名とテンプレートに基づいて変更セットを作成します。変更セットが存在する場合、 はそれ CloudFormation を削除し、新しい変更セットを作成します。
- スタックが存在しない場合は、CREATE_UPDATE がスタックを作成します スタックが存在する場合、 はスタック CloudFormation を更新します。既存のスタックを更新するには、このアクションを使用します。REPLACE_ON_FAILURE とは異なり、スタックが存在し、失敗した状態の場合、CodePipeline はスタックを削除して置き換えることはありません。
- DELETE_ONLY は、スタックを削除します。存在しないスタックを指定した場合は、アクションはスタックを削除せずに正常に終了します。
- スタックが存在しない場合は、REPLACE_ON_FAILURE がスタックを作成します。スタックが存在し、障害状態にある場合、 はスタック CloudFormation を削除し、新しいスタックを作成します。スタックが失敗状態でない場合、 はスタック CloudFormation を更新します。

  CloudFormationに次のいずれかのステータスタイプが表示されている場合、スタックは失敗状態になります。

  - ROLLBACK_FAILED
  - CREATE_FAILED
  - DELETE_FAILED
  - UPDATE_ROLLBACK_FAILED

  失敗したスタックをリカバリーまたはトラブルシューティングせずに自動的に置き換えるには、このアクションを使用します。

  > **Important**
  >
  > REPLACE_ON_FAILURE は、スタックが削除される可能性があるため、テスト目的でのみ使用することをお勧めします。

StackName

必須: はい

StackName は、既存のスタックの名前、または作成するスタックの名前です。

機能

必須: 条件による

Capabilities を使用すると、テンプレートに一部のリソースを単独で作成および更新する機能があり、これらの機能はテンプレート内のリソースのタイプに基づいて決定されることが確認されます。

このプロパティは、スタックテンプレートに IAM リソースがある場合、またはマクロを含むテンプレートから直接スタックを作成する場合に必要です。 CloudFormation アクションをこの方法で正常に動作させるには、次のいずれかの機能を使用してアクションが正常に動作することを明示的に承認する必要があります。

- CAPABILITY_IAM
- CAPABILITY_NAMED_IAM
- CAPABILITY_AUTO_EXPAND

機能間にカンマ (スペースなし) を使用して、複数の機能を指定できます。アクションの宣言 の例は、CAPABILITY_IAM プロパティと CAPABILITY_AUTO_EXPAND プロパティの両方を持つエントリを示しています。

Capabilities の詳細については、[AWS CloudFormation API リファレンス]の [UpdateStack] のプロパティを参照してください。

ChangeSetName

必須: 条件による

ChangeSetName は、既存の変更セットの名前、または指定されたスタック用に作成する新しい変更セットの名前です。

このプロパティは、次のアクションモードに必要です。CHANGE_SET_REPLACE および CHANGE_SET_EXECUTE。他のすべてのアクションモードでは、このプロパティは無視されます。

RoleArn

必須: 条件による

RoleArn は、指定されたスタックのリソースを操作するときに CloudFormation が引き受ける IAM サービスロールの ARNです。RoleArn は、変更セットを実行するときには適用されません。変更セットの作成に CodePipeline を使用しない場合は、変更セットまたはスタックにロールが関連付けられていることを確認します。

> **Note**
>
> このロールは、アクション宣言 RoleArn で設定された、実行中のアクションのロールと同じアカウントである必要があります。

このプロパティは、以下のアクションモードでは必須です。

- CREATE_UPDATE
- REPLACE_ON_FAILURE
- DELETE_ONLY
- CHANGE_SET_REPLACE

> **Note**
>
> CloudFormation にはテンプレートへの S3-signed付き URL が付与されるため、アーティファクトバケットにアクセスするためのアクセス許可RoleArnは必要ありません。ただし、アクション RoleArn は、署名付き URL を生成するために、アーティファクト バケットへアクセスする許可を必要と [します]。

TemplatePath

必須: 条件による

TemplatePath は CloudFormation テンプレートファイルを表します。このアクションへの入力アーティファクトにファイルを含めます。ファイル名の形式は次のとおりです:

Artifactname::*TemplateFileName*

Artifactname は、CodePipeline に表示される入力アーティファクト名です。たとえば、アーティファクト名 SourceArtifact と template-export.json ファイル名を持つソースステージでは、次の例に示すような TemplatePath の名前が作成されます。

```
"TemplatePath": "SourceArtifact::template-export.json"
```

このプロパティは、以下のアクションモードでは必須です。

- CREATE_UPDATE
- REPLACE_ON_FAILURE

- CHANGE_SET_REPLACE

他のすべてのアクションモードでは、このプロパティは無視されます。

> **Note**
>
> CloudFormation テンプレート本文を含むテンプレートファイルの最小長は 1 バイト、最大長は 1 MB です。CodePipeline の CloudFormation デプロイアクションの場合、入力アーティファクトの最大サイズは常に 256 MB です。詳細については、AWS CodePipeline のクォータ および [CloudFormation の制限] を参照してください。

OutputFileName

必須: いいえ

`OutputFileName` を使用して、CodePipeline がこのアクションのパイプライン出力アーティファクトに追加する `CreateStackOutput.json` などの出力ファイル名を指定します。JSON ファイルには、 CloudFormation スタックの `Outputs`セクションの内容が含まれています。

名前を指定しない場合、CodePipeline は出力ファイルやアーティファクトを生成しません。

ParameterOverrides

必須: いいえ

パラメータはスタックテンプレートで定義され、スタックの作成時または更新時にそれらの値を指定できます。JSON オブジェクトを使用して、テンプレートにパラメータ値を設定できます。（これらの値は、テンプレート設定ファイルに設定された値を上書きします。）パラメータオーバーライドの使用の詳細については、設定プロパティ（JSON オブジェクト）を参照してください。

パラメータ値のほとんどは、テンプレート設定ファイルを使用して指定することをお勧めします。パラメータの上書きは、パイプラインが実行されるまで不明な値にのみ使用します。詳細については、[AWS CloudFormation ユーザーガイド] の [CodePipeline パイプラインでのパラメータオーバーライド関数の使用] を参照してください。

> **Note**
>
> すべてのパラメータ名がスタックテンプレートに存在する必要があります。

TemplateConfiguration

必須: いいえ

TemplateConfiguration はテンプレート構成ファイルです。このアクションへの入力アーティファクトにファイルを含めます。テンプレート設定ファイルには、テンプレートのパラメータ値およびスタックポリシーを含めることができます。テンプレート設定ファイル形式の詳細については、「AWS CloudFormation アーティファクト」を参照してください。

テンプレート構成ファイル名は以下の形式に従います。

Artifactname::*TemplateConfigurationFileName*

Artifactname は、CodePipeline に表示される入力アーティファクト名です。たとえば、アーティファクト名 SourceArtifact と test-configuration.json ファイル名を持つソースステージでは、次の例に示すような TemplateConfiguration の名前が作成されます。

```
"TemplateConfiguration": "SourceArtifact::test-configuration.json"
```

入力アーティファクト

- アーティファクトの数: 0 to 10
- 説明: 入力として、 CloudFormation アクションはオプションで以下の目的でアーティファクトを受け入れます。
  - 実行するスタックテンプレートファイルを提供するため。(TemplatePath パラメータを参照。)
  - 使用するテンプレート設定ファイルを提供するため。(TemplateConfiguration パラメータを参照。) テンプレート設定ファイル形式の詳細については、「AWS CloudFormation アーティファクト」を参照してください。
  - CloudFormation スタックの一部としてデプロイする Lambda 関数のアーティファクトを提供します。

出力アーティファクト

- アーティファクトの数: 0 to 1

- 説明: `OutputFileName` パラメータが指定された場合、このアクションによって生成される出力アーティファクトには、指定された名前の JSON ファイルが含まれます。JSON ファイルには、CloudFormation スタックの「出力」セクションの内容が含まれています。

  CloudFormation アクション用に作成できる出力セクションの詳細については、出力を参照してください。

出力変数

このアクションを設定すると、パイプライン内のダウンストリームアクションのアクション設定によって参照できる変数が生成されます。名前空間を使用してアクションを設定し、これらの変数をダウンストリームアクションの設定で使用できるようにします。

CloudFormation アクションの場合、変数はスタックテンプレートの `Outputs`セクションで指定された値から生成されます。CloudFormation アクションモードのうち、出力を生成するのは、スタックの作成、スタックの更新、変更セットの実行など、スタックの作成や更新を伴うアクションモードのみです。変数を生成するアクションモードは次のとおりです。

- `CHANGE_SET_EXECUTE`
- `CHANGE_SET_REPLACE`
- `CREATE_UPDATE`
- `REPLACE_ON_FAILURE`

詳細については、「変数リファレンス」を参照してください。CloudFormation 出力変数を使用するパイプラインを CloudFormation デプロイアクションで作成するためのチュートリアルについては、「チュートリアル: AWS CloudFormation デプロイアクションの変数を使用するパイプラインを作成する」を参照してください。

サービスロールのアクセス許可: CloudFormation アクション

CodePipeline がアクションを実行する際、CodePipeline サービスロールポリシーには、最小特権でアクセスを維持するために、パイプラインリソース ARN に適切にスコープダウンされた次のアクセス許可が必要です。例えば、以下をポリシーステートメントに追加します。

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCFNStackAccess",
            "Effect": "Allow",
            "Action": [
                "cloudformation:CreateStack",
                "cloudformation:UpdateStack",
                "cloudformation:DeleteStack",
                "cloudformation:DescribeStacks",
                "cloudformation:DescribeStackResources",
                "cloudformation:DescribeStackEvents",
                "cloudformation:GetTemplate",
                "cloudformation:DescribeChangeSet",
                "cloudformation:CreateChangeSet",
                "cloudformation:DeleteChangeSet",
                "cloudformation:ExecuteChangeSet"
            ],
            "Resource": [
                "arn:aws:cloudformation:*:111122223333:stack/
[[cfnDeployStackNames]]/*"
            ]
        },
        {
            "Sid": "ValidateTemplate",
            "Effect": "Allow",
            "Action": [
                "cloudformation:ValidateTemplate"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowIAMPassRole",
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::111122223333:role/[[cfnExecutionRoles]]"
            ],
            "Condition": {
                "StringEqualsIfExists": {
                    "iam:PassedToService": [
                        "cloudformation.amazonaws.com"
```

```
                    ]
                }
            }
        }
    ]
}
```

cloudformation:DescribeStackEvents アクセス許可はオプションであることに注意してください。これにより、 CloudFormation アクションはより詳細なエラーメッセージを表示できます。パイプラインのエラーメッセージにリソースの詳細を表示したくない場合は、このアクセス許可を IAM ロールから取り消すことができます。

アクションの宣言

YAML

```
Name: ExecuteChangeSet
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Provider: CloudFormation
  Version: '1'
RunOrder: 2
Configuration:
  ActionMode: CHANGE_SET_EXECUTE
  Capabilities: CAPABILITY_NAMED_IAM,CAPABILITY_AUTO_EXPAND
  ChangeSetName: pipeline-changeset
  ParameterOverrides: '{"ProjectId": "my-project","CodeDeployRole":
 "CodeDeploy_Role_ARN"}'
  RoleArn: CloudFormation_Role_ARN
  StackName: my-project--lambda
  TemplateConfiguration: 'my-project--BuildArtifact::template-configuration.json'
  TemplatePath: 'my-project--BuildArtifact::template-export.yml'
OutputArtifacts: []
InputArtifacts:
  - Name: my-project-BuildArtifact
```

JSON

```
{
    "Name": "ExecuteChangeSet",
```

```
    "ActionTypeId": {
        "Category": "Deploy",
        "Owner": "AWS",
        "Provider": "CloudFormation",
        "Version": "1"
    },
    "RunOrder": 2,
    "Configuration": {
        "ActionMode": "CHANGE_SET_EXECUTE",
        "Capabilities": "CAPABILITY_NAMED_IAM,CAPABILITY_AUTO_EXPAND",
        "ChangeSetName": "pipeline-changeset",
        "ParameterOverrides": "{\"ProjectId\": \"my-project\",\"CodeDeployRole\":
 \"CodeDeploy_Role_ARN\"}",
        "RoleArn": "CloudFormation_Role_ARN",
        "StackName": "my-project--lambda",
        "TemplateConfiguration": "my-project--BuildArtifact::template-
configuration.json",
        "TemplatePath": "my-project--BuildArtifact::template-export.yml"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
             "Name": "my-project-BuildArtifact"
        }
    ]
},
```

関連情報

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- [設定プロパティ リファレンス] - [AWS CloudFormation ユーザーガイド] のリファレンスの章では、これらの CodePipeline パラメータの詳細な説明と例をご覧いただけます。
- AWS CloudFormation API リファレンス – AWS CloudFormation API リファレンスの CreateStack パラメータは、 テンプレートの CloudFormation スタックパラメータを記述します。

CloudFormation StackSets デプロイアクションリファレンス

CodePipeline は、CI/CD プロセスの一部として CloudFormation StackSets オペレーションを実行する機能を提供します。スタックセットを使用して、単一の CloudFormation テンプレートを使用して

AWS リージョン間で AWS アカウントにスタックを作成します。各スタックに含まれるすべてのリソースは、スタックセットの CloudFormation テンプレートによって定義されます。スタックセットを作成する際、使用するテンプレートに加え、そのテンプレートで必要なパラメータや機能を指定します。

CloudFormation StackSets の概念の詳細については、「 AWS CloudFormation ユーザーガイド」の「StackSets の概念」を参照してください。

パイプラインを CloudFormation StackSets と統合するには、一緒に使用する 2 つの異なるアクションタイプを使用します。

- `CloudFormationStackSet` アクションはスタックセット、またはパイプラインのソース場所に保存されているテンプレートからのスタックインスタンスを作成または更新します。スタックセットが作成または更新されるたびに、指定したインスタンスへの変更のデプロイが開始されます。コンソールでは、パイプラインを作成または編集するとき、[CloudFormation スタックセット] アクションプロバイダーを選択します。
- `CloudFormationStackInstances` アクションは、指定したインスタンスへの `CloudFormationStackSet` アクションからの変更のデプロイ、新しいスタックインスタンスの作成、および指定されたインスタンスに対するパラメータの上書きを定義します。コンソールでは、パイプラインを作成または編集するとき、[CloudFormation スタックインスタンス] アクションプロバイダーを選択します。

これらのアクションを使用して、ターゲット AWS アカウントまたはターゲット AWS の Organizations 組織単位 IDs にデプロイできます。

> **Note**
>
> Organizations アカウントまたは組織単位 IDs AWS をターゲットにデプロイし、サービスマネージドアクセス許可モデルを使用するには、 AWS CloudFormation StackSets と AWS Organizations 間の信頼されたアクセスを有効にする必要があります。詳細については、「Stacksets AWS CloudFormation による信頼されたアクセスの有効化」を参照してください。

トピック

- Stack CloudFormation StackSets アクションの仕組み
- パイプラインで StackSets アクションを構成する方法

- CloudFormationStackSet アクション
- CloudFormationStackInstances アクション
- サービスロールのアクセス許可: CloudFormationStackSet アクション
- サービスロールのアクセス許可: CloudFormationStackInstances アクション
- スタックセットオペレーションのアクセス許可モデル
- テンプレートパラメータのデータタイプ
- 関連情報

Stack CloudFormation StackSets アクションの仕組み

`CloudFormationStackSet` アクションは、アクションが初めて実行されているかどうかに応じて、リソースを作成または更新します。

`CloudFormationStackSet` アクションはスタックセットを 作成 または 更新 して、指定されたインスタンスに変更をデプロイします。

> **Note**
>
> このアクションを使用してスタックインスタンスの追加を含む更新を行う場合、新しいインスタンスが最初にデプロイされ、更新は最後に完了します。新しいインスタンスは最初に古いバージョンを受け取り、次に更新がすべてのインスタンスに適用されます。

- 作成: インスタンスが指定されておらず、スタックセットが存在しない場合は、CloudFormationsStackSet アクション は、インスタンスを作成せずにスタックセットを作成します。
- 更新 : CloudFormationsStackSet アクションがすでに作成されたスタックセットに対して実行されているとき、アクションはスタックセットを更新します。インスタンスが指定されておらず、スタックセットがすでに存在する場合は、すべてのインスタンスが更新されます。このアクションが特定の インスタンスの更新に使用されている場合、残りのすべてのインスタンスが OUTDATED ステータスに移行します。

  CloudFormationsStackSet アクションを使用して、次の方法でスタックセットを更新します。
  - 一部またはすべてのインスタンスでテンプレートを更新します。
  - 一部またはすべてのインスタンスのパラメータを更新します。

- スタックセット の実行ロールを更新します。(これは、管理者ロールで指定された 実行 ロール と一致しなければなりません)。
- アクセス許可 モデルを変更します (インスタンスが作成されていない場合のみ)。
- スタックセットのアクセス許可モデルが Service Managed の場合、AutoDeployment を有効または無効にします。
- スタックセットのアクセス許可モデルが Service Managed の場合は、メンバーアカウントの委任管理者となります。
- 管理者ロール を更新します。
- スタックセット の説明を更新します。
- デプロイターゲットをスタックセットの更新に追加して、新しいスタックインスタンスを作成します。

CloudFormationStackInstances アクションは、新しいスタックインスタンスを作成するか、古いスタックインスタンスを更新します。スタックセットが更新されると、インスタンスは古くなりますが、その中のすべてのインスタンスが更新されるわけではありません。

- 作成: スタックがすでに存在する場合には、CloudFormationStackInstances アクションはインスタンスの更新のみを行い、スタックインスタンスは作成しません。
- 更新: CloudFormationStackSet アクションが実行された後に、テンプレートまたはパラメータが一部のインスタンスでのみ更新された場合、残りは OUTDATED とマークされます。後期のパイプラインのステージでは、CloudFormationStackInstances は断続的にスタックセット内の残りのインスタンスを更新して、すべてのインスタンスが CURRENT とマークされるようにします。このアクションは、インスタンスを追加、または新しいインスタンスまたは既存のインスタンスでパラメータをオーバーライドするために使用できます。

アップデートの一環として、CloudFormationStackSet そして CloudFormationStackInstances アクションは、新しいデプロイターゲットを指定して、新しいスタックインスタンスを作成します。

アップデートの一環として、CloudFormationStackSet そして CloudFormationStackInstances アクションは、スタックセット、インスタンス、またはリソースを削除しません。アクションがスタックを更新する一方、すべてのインスタンスが更新されないよう指定する場合、更新を指定しなかったインスタンスは更新から除外され、ステータスが OUTDATED に設定されます。

デプロイ中、インスタンスへのデプロイが失敗した場合、スタックインスタンスは、OUTDATED のステータスを表示することもできます。

パイプラインで StackSets アクションを構成する方法

ベストプラクティスとして、スタックセットが作成され、最初にサブセットまたは単一のインスタンスにデプロイされるようにパイプラインを構築する必要があります。デプロイをテストし、生成されたスタックセットを表示したら、CloudFormationStackInstances アクションを追加し、残りのインスタンスが作成および更新されるようにします。

コンソールまたは CLI を使用して、次のように推奨されるパイプライン構造を作成します。

1. ソースアクションを持つパイプラインを作成し (必須)、CloudFormationStackSet アクションをデプロイアクションとして指定します。パイプラインを実行します。
2. パイプラインが最初に実行されると、CloudFormationStackSet アクションがスタックセットと少なくとも 1 つの初期インスタンスを作成します。スタックセットの作成を確認し、初期インスタンスへのデプロイを確認します。例えば、us-east-1 が指定されたリージョンであるアカウント Account-A のスタックセットの初期作成では、スタックインスタンスは次のスタックセットで作成されます。

スタックインスタンス	リージョン	ステータス
StackInstanceID-1	us-east-1	CURRENT

3. パイプラインを編集して、指定したターゲットのスタックインスタンスを作成または更新する 2 番目のデプロイアクションとして CloudFormationStackInstances を追加します。例えば、us-east-2 およびeu-central-1 リージョンが指定されるアカウント Account-A のスタックインスタンスの作成の場合、残りのスタックインスタンスが作成され、初期インスタンスは次のように更新されます。

スタックインスタンス	リージョン	ステータス
StackInstanceID-1	us-east-1	CURRENT
StackInstanceID-2	us-east-2	CURRENT
StackInstanceID-3	eu-central-1	CURRENT

4. 必要に応じてパイプラインを実行して、スタックセットを更新し、スタックインスタンスを更新または作成します。

アクション設定からデプロイターゲットを削除したスタックの更新を開始すると、更新用に指定されていなかったスタックインスタンスがデプロイから削除され、OUTDATED ステータスに移行します。たとえば、Account-A リージョンがアクション設定から削除されたアカウント us-east-2 のスタックインスタンスの更新の場合、残りのスタックインスタンスが作成され、削除されたインスタンスは OUTDATED に設定されます。

スタックインスタンス	リージョン	ステータス
StackInstanceID-1	us-east-1	CURRENT
StackInstanceID-2	us-east-2	OUTDATED
StackInstanceID-3	eu-central-1	CURRENT

スタックセットのデプロイのベストプラクティスの詳細については、AWS CloudFormation ユーザーガイドで StackSets のベストプラクティスを参照してください。

CloudFormationStackSet アクション

アクションはパイプラインのソース場所に保存されているテンプレートからのスタックセットを作成または更新します。

スタックセットを定義したら、設定パラメータに指定されたターゲットアカウントや リージョンでスタックを作成、更新、削除できるようになります。スタックの作成、更新、削除の際に、オペレーションを実行するリージョンの順序、スタックオペレーションを停止するフォールトトレランスパーセンテージ、スタックでオペレーションを同時に実行するアカウント数など、その他の環境設定を指定することができます。

スタックセットはリージョンのリソースです。1 つの AWS リージョンでスタックセットを作成する場合、他のリージョンからスタックセットにアクセスすることはできません。

このアクションをスタックセットに対する更新アクションとして使用する場合、少なくとも 1 つのスタック インスタンスにデプロイがないと、スタックの更新は許可されません。

トピック

- アクションタイプ
- 設定パラメータ
- 入力アーティファクト
- 出力アーティファクト
- 出力変数
- 例 CloudFormationStackSet アクションの設定

アクションタイプ

- カテゴリ:Deploy
- 所有者: AWS
- プロバイダー: CloudFormationStackSet
- バージョン: 1

設定パラメータ

StackSetName

必須: はい

スタックセットに関連付ける名前。この名前は作成されるリージョンで一意であることが必要です。

名前には、英数字とハイフンのみを使用することができます。アルファベットで始まり、また 128 文字以下である必要があります。

説明

必須: いいえ

スタックセットの説明。これを使用して、スタックセットの目的やその他の関連情報を記述できます。

TemplatePath

必須: はい

スタックセット内のリソースを定義するテンプレートの口ケーション。これは、最大サイズ 460,800 byte のテンプレートを指定する必要があります。

フォーマット "InputArtifactName::TemplateFileName" で、ソースアーティファクト名とテンプレートファイルへのパスを次の例に示すように入力します。

```
SourceArtifact::template.txt
```

パラメータ

必須: いいえ

デプロイ中に更新されるスタックセットのテンプレートパラメータのリスト。

パラメータはリテラルリストまたはファイルパスとして提供できます。

- パラメータは、次の ParameterKey=string,ParameterValue=string,UsePreviousValue=boolean,ResolvedV ParameterKey=string,ParameterValue=string,UsePreviousValue=boolean,ResolvedV のような省略構文のフォーマットで入力できます。これらのデータタイプの詳細については、「テンプレートパラメータのデータタイプ」を参照してください。

  次の例は、amzn-s3-demo-source-bucket という名前のパラメータの値 BucketName を示しています。

  ```
  ParameterKey=BucketName,ParameterValue=amzn-s3-demo-source-bucket
  ```

  次の例は、複数のパラメータを持つエントリを示しています。

  ```
  ParameterKey=BucketName,ParameterValue=amzn-s3-demo-source-bucket
  ParameterKey=Asset1,ParameterValue=true
  ParameterKey=Asset2,ParameterValue=true
  ```

- 次の例で示すように、フォーマット "InputArtifactName::ParametersFileName" で入力されたテンプレートパラメータのオーバーライドのリストを含むファイルのロケーションを入力できます。

  ```
  SourceArtifact::parameters.txt
  ```

  次の例では、parameters.txt のファイルの内容を示します。

  ```
  [
  ```

```
    {
        "ParameterKey": "KeyName",
        "ParameterValue": "true"
    },
    {
        "ParameterKey": "KeyName",
        "ParameterValue": "true"
    }
]
```

機能

必須: いいえ

テンプレート内のリソースのタイプに応じて、テンプレートがリソースを作成および更新できることを示します。

このプロパティは、スタックテンプレートに IAM リソースがある場合、またはマクロを含むテンプレートから直接スタックを作成する場合に使用する必要があります。 CloudFormation アクションをこの方法で正常に動作させるには、次のいずれかの機能を使用する必要があります。

- `CAPABILITY_IAM`
- `CAPABILITY_NAMED_IAM`

機能間にカンマ (スペースなし) を使用して、複数の機能を指定できます。例 CloudFormationStackSet アクションの設定 の例は、複数の機能を持つエントリを示しています。

PermissionModel

必須: いいえ

IAM ロールの作成および管理方法を決定します。フィールドを指定しない場合、デフォルトが使用されます。詳細については、「スタックセットオペレーションのアクセス許可モデル」を参照してください。

次の値を指定できます。

- `SELF_MANAGED` (デフォルト): ターゲットアカウントにデプロイするには、管理者ロールと実行ロールを作成する必要があります。
- `SERVICE_MANAGED`: CloudFormation StackSets は、 AWS Organizations が管理するアカウントにデプロイするために必要な IAM ロールを自動的に作成します。これには、アカウントが組織のメンバーである必要があります。

> **Note**
>
> このパラメータは、スタックセットにスタックインスタンスが存在しない場合にのみ変更できます。

AdministrationRoleARN

> **Note**
>
> CloudFormation StackSets は複数のアカウントでオペレーションを実行するため、スタックセットを作成する前に、それらのアカウントで必要なアクセス許可を定義する必要があります。

必須: いいえ

> **Note**
>
> このパラメータは SELF_MANAGED アクセス許可モデルの場合はオプションで、SERVICE_MANAGED アクセス許可モデルには使用されません。

スタックセットオペレーションの実行に使用される管理者アカウントの IAM ロールの ARN。

名前には、英数字と記号 _+=,.@-, を使用できます。スペースは使用できません。名前では、大文字と小文字は区別されません。このロール名は、最小 20 文字、最大 2048 文字の長さでなければなりません。ロール名はアカウント内で一意である必要があります。ここで指定するロール名は、既存のロール名である必要があります。ロール名を指定しない場合は、AWSCloudFormationStackSetAdministrationRole に設定されます。ServiceManaged を指定する場合は、ロール名を定義してはいけません。

ExecutionRoleName

> **Note**
>
> CloudFormation StackSets は複数のアカウントでオペレーションを実行するため、スタックセットを作成する前に、それらのアカウントで必要なアクセス許可を定義する必要があります。

必須: いいえ

> **Note**
>
> このパラメータは SELF_MANAGED アクセス許可モデルの場合はオプションで、SERVICE_MANAGED アクセス許可モデルには使用されません。

スタックセットオペレーションの実行に使用されるターゲットアカウントの IAM ロールの名前。名前には、英数字と記号 _+=,.@-, を使用できます。スペースは使用できません。名前では、大文字と小文字は区別されません。このロール名は、最小 1 文字、最大 64 文字の長さでなければなりません。ロール名はアカウント内で一意である必要があります。ここで指定するロール名は、既存のロール名である必要があります。カスタマイズされた実行ロールを使用している場合は、このロールを指定しないでください。ロール名を指定しない場合は、`AWSCloudFormationStackSetExecutionRole` に設定されます。Service_Managed を true に設定する場合は、ロール名を定義してはいけません。

OrganizationsAutoDeployment

必須: いいえ

> **Note**
>
> このパラメータは SERVICE_MANAGED アクセス許可モデルの場合はオプションで、SELF_MANAGED アクセス許可モデルには使用されません。

ターゲット組織または組織単位 (OU) に追加された Organizations アカウントに CloudFormation StackSets が自動的にデプロイ AWS されるかどうかについて説明します。もし `OrganizationsAutoDeployment` が指定されている場合は、`DeploymentTargets` そして `Regions` を指定しないでください。

> **Note**
>
> `OrganizationsAutoDeployment` に入力が指定されていない場合、デフォルト値は `Disabled` です。

次の値を指定できます。

- `Enabled`。必須: いいえ

  StackSets は、指定したリージョンのターゲット組織または組織単位 (OU) に追加した AWS Organizations アカウントに追加のスタックインスタンスを自動的にデプロイします。アカウントがターゲット組織または OU から削除された場合、 CloudFormation StackSets は指定されたリージョンのアカウントからスタックインスタンスを削除します。

- `Disabled`。必須: いいえ

  StackSets は、指定されたリージョンのターゲット組織または組織単位 (OU) に追加された AWS Organizations アカウントに、追加のスタックインスタンスを自動的にデプロイしません。

- `EnabledWithStackRetention`。必須: いいえ

  ターゲット組織または OU からアカウントを削除したときに スタックリソースは保持されます。

DeploymentTargets

必須: いいえ

> **Note**
>
> SERVICE_MANAGED アクセス許可モデルの場合、デプロイターゲットに組織ルート ID または組織単位 ID を提供できます。SELF_MANAGED アクセス許可モデルの場合、アカウントのみを提供できます。

> **Note**
>
> このパラメータを選択する場合は、リージョン も選択する必要があります。

スタックセットインスタンスを作成/更新する AWS アカウントまたは組織単位 IDs のリスト。

- Accounts:

  アカウントは、リテラルリストまたはファイルパスとして指定できます。

  - リテラル: 次の例に示すように、パラメータを省略構文のフォーマット `account_ID,account_ID` で入力します。

```
111111222222,333333444444
```

- ファイルパス: スタックセットインスタンスを作成/更新する AWS アカウントのリストを含むファイルの場所。 形式で入力しますInputArtifactName::AccountsFileName。ファイルパスを使用して accounts または OrganizationalUnitIds のいずれかを指定する場合、以下の例に示すように、ファイル形式は JSON であることが必要です。

  ```
  SourceArtifact::accounts.txt
  ```

  次の例では、accounts.txt のファイルの内容を示します。

  ```
  [
      "111111222222"
  ]
  ```

  以下の例では、複数のアカウントを一覧表示したときの accounts.txt のファイルの内容を示しています。

  ```
  [
      "111111222222","333333444444"
  ]
  ```

- OrganizationalUnitIds

  > **Note**
  > このパラメータは SERVICE_MANAGED アクセス許可モデルの場合はオプションで、SELF_MANAGED アクセス許可モデルには使用されません。OrganizationsAutoDeployment を選択した場合は、これを使用しないでください。

  関連付けられたスタックインスタンスを更新する AWS 組織単位。

  組織単位 ID は、リテラルリストまたはファイルパスとして提供できます。

  - リテラル: 次の例に示すように、カンマで区切って文字列の配列を入力します。

    ```
    ou-examplerootid111-exampleouid111,ou-examplerootid222-exampleouid222
    ```

- ファイルパス: スタックセットインスタンスを作成または更新する OrganizationalUnitIds のリストを含むファイルの場所。ファイルパスを使用して accounts または OrganizationalUnitIds のいずれかを指定する場合、以下の例に示すように、ファイル形式は JSON であることが必要です。

  ファイルのパスをフォーマット `InputArtifactName::OrganizationalUnitIdsFileName` で入力します。

  ```
  SourceArtifact::OU-IDs.txt
  ```

  次の例では、`OU-IDs.txt` のファイルの内容を示します。

  ```
  [
      "ou-examplerootid111-exampleouid111","ou-examplerootid222-exampleouid222"
  ]
  ```

リージョン

必須: いいえ

> **Note**
> このパラメータを選択する場合は、DeploymentTargets も選択する必要があります。

スタックセットインスタンスが作成または更新される AWS リージョンのリスト。リージョンは、入力された順序で更新されます。

次の例に示すように`Region1,Region2`、有効な AWS リージョンのリストを の形式で入力します。

```
us-west-2,us-east-1
```

FailureTolerancePercentage

必須: いいえ

このスタックオペレーションが失敗するリージョンあたりのアカウントの割合。 はそのリージョンでオペレーションを CloudFormation 停止します。オペレーションがリージョンで停止してい

る場合、 CloudFormation は後続のリージョンでオペレーションを試みません。指定された割合に基づいてアカウント数を計算する場合、 は次の整数に CloudFormation 切り下げられます。

MaxConcurrentPercentage

必須: いいえ

このオペレーションを一度に実行するアカウントの最大の割合。指定された割合に基づいてアカウント数を計算する場合、 は次の整数に CloudFormation 切り下げられます。切り捨てるとゼロになる場合、 は代わりに数値を 1 として CloudFormation 設定します。この設定で 最大値 を指定する場合でも、大規模なデプロイでは、同時に処理される実際のアカウント数はサービスのスロットリングのために低くなる可能性があります。

RegionConcurrencyType

必須: いいえ

リージョン同時デプロイパラメータを設定することで、スタックセットを AWS リージョン 全体に順次デプロイするか、並列デプロイするかを指定できます。複数の にスタックを並行 AWS リージョン してデプロイするようにリージョンの同時実行を指定すると、全体的なデプロイ時間が短縮される可能性があります。

- 並列: スタックセットのデプロイは、指定された失敗許容回数をリージョンのデプロイ失敗が超えない限り、同時に行われます。
- 順次: スタックセットのデプロイは、リージョンのデプロイ失敗が指定された失敗許容回数を超えない限り、一度に 1 つずつ行われます。デフォルトでは、順次デプロイが選択されています。

ConcurrencyMode

必須: いいえ

同時実行モードでは、スタックセットオペレーション時の同時実行レベルの動作を、厳密な耐障害性またはソフトな耐障害性のいずれかを選択できます。厳密な障害耐性では、障害が発生するたびに同時実行性が低下するため、スタックセットの操作に障害が発生するため、デプロイ速度が低下します。ソフト障害耐性は、 CloudFormation 安全機能を活用しながら、デプロイ速度を優先します。

- STRICT_FAILURE_TOLERANCE: このオプションでは、失敗したアカウントの数が特定の耐障害性を超えないように、同時実行レベルを動的に下げます。これがデフォルトの動作です。
- SOFT_FAILURE_TOLERANCE: このオプションは実際の同時実行性から耐障害性を切り離します。これにより、障害の数に関係なく、スタックセットの操作を設定された同時実行レベルで実行できます。

CallAs

必須: いいえ

> **Note**
>
> このパラメータは、SERVICE_MANAGED アクセス許可モデルではオプションであり、SELF_MANAGED アクセス許可モデルでは使用しません。

組織の管理アカウントで行動するか、メンバーアカウントで委任管理者として行動するかを指定します。

> **Note**
>
> このパラメータを DELEGATED_ADMIN に設定する場合は、パイプライン IAM ロールに organizations:ListDelegatedAdministrators アクセス許可があることを確認してください。それ以外の場合、アクションは実行中に失敗し、「Account used is not a delegated administrator」のようなエラーが発生します。

- SELF: スタックセットのデプロイでは、管理アカウントにサインインしている間、サービスマネージドアクセス許可を使用します。
- DELEGATED_ADMIN: スタックセットのデプロイでは、委任管理者アカウントにサインインしている間、サービスマネージドアクセス許可を使用します。

入力アーティファクト

CloudFormationStackSet アクションでスタックセットのテンプレートを含む入力アーティファクトを少なくとも 1 つ含める必要があります。デプロイターゲット、アカウント、およびパラメータのリストには、より多くの入力アーティファクトを含めることができます。

- アーティファクトの数: 1 to 3
- 説明: アーティファクトを含めて、以下を提供できます。
  - スタックテンプレートファイル（TemplatePath パラメータを参照。）
  - パラメータファイル（Parameters パラメータを参照。）
  - アカウントファイル（DeploymentTargets パラメータを参照。）

出力アーティファクト

- アーティファクトの数: `0`
- 説明: 出力アーティファクトは、このアクションタイプには適用されません。

出力変数

このアクションを設定すると、パイプライン内のダウンストリームアクションのアクション設定によって参照できる変数が生成されます。名前空間を使用してアクションを設定し、これらの変数をダウンストリームアクションの設定で使用できるようにします。

- StackSetId: スタックセットの ID。
- OperationId: スタックセットオペレーションの ID。

詳細については、「変数リファレンス」を参照してください。

例 CloudFormationStackSet アクションの設定

次の例は、CloudFormationStackSet アクションのアクション設定を示しています。

自己管理型のアクセス許可モデルの例

次の例は、入力されたデプロイターゲットが AWS アカウント ID である CloudFormationStackSet アクションを示しています。

YAML

```
Name: CreateStackSet
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Provider: CloudFormationStackSet
  Version: '1'
RunOrder: 1
Configuration:
  DeploymentTargets: '111111222222'
  FailureTolerancePercentage: '20'
  MaxConcurrentPercentage: '25'
  PermissionModel: SELF_MANAGED
  Regions: us-east-1
  StackSetName: my-stackset
```

```
  TemplatePath: 'SourceArtifact::template.json'
OutputArtifacts: []
InputArtifacts:
  - Name: SourceArtifact
Region: us-west-2
Namespace: DeployVariables
```

JSON

```
{
    "Name": "CreateStackSet",
    "ActionTypeId": {
        "Category": "Deploy",
        "Owner": "AWS",
        "Provider": "CloudFormationStackSet",
        "Version": "1"
    },
    "RunOrder": 1,
    "Configuration": {
        "DeploymentTargets": "111111222222",
        "FailureTolerancePercentage": "20",
        "MaxConcurrentPercentage": "25",
        "PermissionModel": "SELF_MANAGED",
        "Regions": "us-east-1",
        "StackSetName": "my-stackset",
        "TemplatePath": "SourceArtifact::template.json"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "SourceArtifact"
        }
    ],
    "Region": "us-west-2",
    "Namespace": "DeployVariables"
}
```

サービス管理型のアクセス許可モデルの例

次の例は、 AWS Organizations への自動デプロイのオプションがスタック保持で有効になっている、サービスマネージドアクセス許可モデルの CloudFormationStackSet アクションを示しています。

YAML

```
Name: Deploy
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Provider: CloudFormationStackSet
  Version: '1'
RunOrder: 1
Configuration:
  Capabilities: 'CAPABILITY_IAM,CAPABILITY_NAMED_IAM'
  OrganizationsAutoDeployment: EnabledWithStackRetention
  PermissionModel: SERVICE_MANAGED
  StackSetName: stacks-orgs
  TemplatePath: 'SourceArtifact::template.json'
OutputArtifacts: []
InputArtifacts:
  - Name: SourceArtifact
Region: eu-central-1
Namespace: DeployVariables
```

JSON

```
{
    "Name": "Deploy",
    "ActionTypeId": {
        "Category": "Deploy",
        "Owner": "AWS",
        "Provider": "CloudFormationStackSet",
        "Version": "1"
    },
    "RunOrder": 1,
    "Configuration": {
        "Capabilities": "CAPABILITY_IAM,CAPABILITY_NAMED_IAM",
        "OrganizationsAutoDeployment": "EnabledWithStackRetention",
        "PermissionModel": "SERVICE_MANAGED",
        "StackSetName": "stacks-orgs",
        "TemplatePath": "SourceArtifact::template.json"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "SourceArtifact"
```

```
        }
    ],
    "Region": "eu-central-1",
    "Namespace": "DeployVariables"
}
```

CloudFormationStackInstances アクション

このアクションは 新しいインスタンスを作成し、指定されたインスタンスに スタックセットをデプロイします。スタックインスタンスは、リージョン内のターゲットアカウントのスタックへのリファレンスです。スタックインスタンスは、スタックがなくても存在することができます。たとえば、スタックの作成が失敗した場合は、スタック作成の失敗理由がスタックインスタンスに表示されます。スタックインスタンスに関連付けられるスタックセットは、1 つのみです。

スタックセットの最初の作成後、CloudFormationStackInstances を使用して新しいスタックインスタンスを追加できます。テンプレートパラメータ値は、スタックセットインスタンスの作成または更新オペレーション中にスタックインスタンスレベルでオーバーライドできます。

各スタックセットには、1 つのテンプレートとテンプレートパラメータのセットがあります。テンプレートまたはテンプレートパラメータを更新すると、セット全体のパラメータが更新されます。次に、変更がそのインスタンスにデプロイされるまですべてのインスタンスステータスが OUTDATED に設定されます。

特定のインスタンスのパラメータ値をオーバーライドするには、たとえばテンプレートに stage のパラメータが prod の値で含まれている場合、そのパラメータ値を beta または gamma にオーバーライドできます。

トピック

- アクションタイプ
- 設定パラメータ
- 入力アーティファクト
- 出力アーティファクト
- 出力変数
- アクション設定の例

アクションタイプ

- カテゴリ:`Deploy`
- 所有者: `AWS`
- プロバイダー: `CloudFormationStackInstances`
- バージョン: `1`

設定パラメータ

StackSetName

必須: はい

スタックセットに関連付ける名前。この名前は作成されるリージョンで一意であることが必要です。

名前には、英数字とハイフンのみを使用することができます。アルファベットで始まり、また 128 文字以下である必要があります。

DeploymentTargets

必須: いいえ

> **Note**
>
> SERVICE_MANAGED アクセス許可モデルの場合、デプロイターゲットに組織ルート ID または組織単位 ID を提供できます。SELF_MANAGED アクセス許可モデルの場合、アカウントのみを提供できます。

> **Note**
>
> このパラメータを選択する場合は、リージョン も選択する必要があります。

スタックセットインスタンスを作成/更新する AWS アカウントまたは組織単位 IDs のリスト。

- Accounts:

  アカウントは、リテラルリストまたはファイルパスとして指定できます。

- リテラル: 次の例に示すように、パラメータを省略構文のフォーマット `account_ID,account_ID` で入力します。

```
111111222222,333333444444
```

- ファイルパス: スタックセットインスタンスを作成/更新する AWS アカウントのリストを含むファイルの場所。 形式で入力します`InputArtifactName::AccountsFileName`。ファイルパスを使用して accounts または OrganizationalUnitIds のいずれかを指定する場合、以下の例に示すように、ファイル形式は JSON であることが必要です。

```
SourceArtifact::accounts.txt
```

次の例では、`accounts.txt` のファイルの内容を示します。

```
[
    "111111222222"
]
```

以下の例では、複数のアカウントを一覧表示したときの `accounts.txt` のファイルの内容を示しています。

```
[
    "111111222222","333333444444"
]
```

- OrganizationalUnitIds

> **Note**
>
> このパラメータは SERVICE_MANAGED アクセス許可モデルの場合はオプションで、SELF_MANAGED アクセス許可モデルには使用されません。OrganizationsAutoDeployment を選択した場合は、これを使用しないでください。

関連付けられたスタックインスタンスを更新する AWS 組織単位。

組織単位 ID は、リテラルリストまたはファイルパスとして提供できます。

- リテラル: 次の例に示すように、カンマで区切って文字列の配列を入力します。

```
ou-examplerootid111-exampleouid111,ou-examplerootid222-exampleouid222
```

- ファイルパス: スタックセットインスタンスを作成または更新する OrganizationalUnitIds のリストを含むファイルの場所。ファイルパスを使用して accounts または OrganizationalUnitIds のいずれかを指定する場合、以下の例に示すように、ファイル形式は JSON であることが必要です。

  ファイルのパスをフォーマット `InputArtifactName::OrganizationalUnitIdsFileName` で入力します。

  ```
  SourceArtifact::OU-IDs.txt
  ```

  次の例では、`OU-IDs.txt` のファイルの内容を示します。

  ```
  [
      "ou-examplerootid111-exampleouid111","ou-examplerootid222-exampleouid222"
  ]
  ```

リージョン

必須: はい

> **Note**
> このパラメータを選択する場合は、DeploymentTargets も選択する必要があります。

スタックセットインスタンスが作成または更新される AWS リージョンのリスト。リージョンは、入力された順序で更新されます。

次の例に示すように`Region1,Region2`、有効な AWS リージョンのリストを の形式で入力します。

```
us-west-2,us-east-1
```

ParameterOverrides

必須: いいえ

選択したスタックインスタンスでオーバーライドしたいスタックセットパラメータのリスト。オーバーライドされたパラメータ値は、指定されたアカウントおよびリージョン内のすべてのスタックインスタンスに適用されます。

パラメータはリテラルリストまたはファイルパスとして提供できます。

- パラメータは、次の `ParameterKey=string,ParameterValue=string,UsePreviousValue=boolean,ResolvedV ParameterKey=string,ParameterValue=string,UsePreviousValue=boolean,ResolvedV` のような省略構文のフォーマットで入力できます。これらのデータタイプの詳細については、「テンプレートパラメータのデータタイプ」を参照してください。

  次の例は、`amzn-s3-demo-source-bucket` という名前のパラメータの値 `BucketName` を示しています。

  ```
  ParameterKey=BucketName,ParameterValue=amzn-s3-demo-source-bucket
  ```

  次の例は、複数のパラメータを持つエントリを示しています。

  ```
  ParameterKey=BucketName,ParameterValue=amzn-s3-demo-source-bucket
  ParameterKey=Asset1,ParameterValue=true
  ParameterKey=Asset2,ParameterValue=true
  ```

- 次の例で示すように、フォーマット `InputArtifactName::ParameterOverridesFileName` で入力されたテンプレートパラメータのオーバーライドのリストを含むファイルのロケーションを入力できます。

  ```
  SourceArtifact::parameter-overrides.txt
  ```

  次の例では、`parameter-overrides.txt` のファイルの内容を示します。

  ```
  [
      {
          "ParameterKey": "KeyName",
          "ParameterValue": "true"
      },
      {
          "ParameterKey": "KeyName",
          "ParameterValue": "true"
  ```

```
    }
]
```

FailureTolerancePercentage

必須: いいえ

このスタックオペレーションが失敗するリージョンあたりのアカウントの割合。 はそのリージョンでオペレーションを CloudFormation 停止します。オペレーションがリージョンで停止している場合、 CloudFormation は後続のリージョンでオペレーションを試みません。指定された割合に基づいてアカウント数を計算する場合、 は次の整数に CloudFormation 切り下げられます。

MaxConcurrentPercentage

必須: いいえ

このオペレーションを一度に実行するアカウントの最大の割合。指定された割合に基づいてアカウント数を計算する場合、 は次の整数に CloudFormation 切り下げられます。切り捨てるとゼロになる場合、 は代わりに数値を 1 として CloudFormation 設定します。最大値 を指定する場合でも、大規模なデプロイでは、同時に処理される実際のアカウント数はサービスのスロットリングのために低くなる可能性があります。

RegionConcurrencyType

必須: いいえ

リージョン同時デプロイパラメータを設定することで、スタックセットを AWS リージョン 全体に順次デプロイするか、並列デプロイするかを指定できます。複数の にスタックを並行 AWS リージョン してデプロイするようにリージョンの同時実行を指定すると、全体的なデプロイ時間が短縮される可能性があります。

- 並列: スタックセットのデプロイは、指定された失敗許容回数をリージョンのデプロイ失敗が超えない限り、同時に行われます。
- 順次: スタックセットのデプロイは、リージョンのデプロイ失敗が指定された失敗許容回数を超えない限り、一度に 1 つずつ行われます。デフォルトでは、順次デプロイが選択されています。

ConcurrencyMode

必須: いいえ

同時実行モードでは、スタックセットオペレーション時の同時実行レベルの動作を、厳密な耐障害性またはソフトな耐障害性のいずれかを選択できます。厳密な障害耐性では、障害が発生するたびに同時実行性が低下するため、スタックセットの操作に障害が発生するため、デプロイ速度

が低下します。ソフト障害耐性は、 CloudFormation 安全機能を活用しながら、デプロイ速度を優先します。

- STRICT_FAILURE_TOLERANCE: このオプションでは、失敗したアカウントの数が特定の耐障害性を超えないように、同時実行レベルを動的に下げます。これがデフォルトの動作です。
- SOFT_FAILURE_TOLERANCE: このオプションは実際の同時実行性から耐障害性を切り離します。これにより、障害の数に関係なく、スタックセットの操作を設定された同時実行レベルで実行できます。

CallAs

必須: いいえ

> **Note**
>
> このパラメータは、SERVICE_MANAGED アクセス許可モデルではオプションであり、SELF_MANAGED アクセス許可モデルでは使用しません。

組織の管理アカウントで行動するか、メンバーアカウントで委任管理者として行動するかを指定します。

> **Note**
>
> このパラメータを DELEGATED_ADMIN に設定する場合は、パイプライン IAM ロールに organizations:ListDelegatedAdministrators アクセス許可があることを確認してください。それ以外の場合、アクションは実行中に失敗し、「Account used is not a delegated administrator」のようなエラーが発生します。

- SELF: スタックセットのデプロイでは、管理アカウントにサインインしている間、サービスマネージドアクセス許可を使用します。
- DELEGATED_ADMIN: スタックセットのデプロイでは、委任管理者アカウントにサインインしている間、サービスマネージドアクセス許可を使用します。

入力アーティファクト

CloudFormationStackInstances にデプロイターゲットとパラメータをリストするアーティファクトを含めることができます。

- アーティファクトの数: `0 to 2`
- 説明: 入力として、 スタックセットアクションはオプションでこれらの目的でアーティファクトを受け入れます。
  - 使用するパラメータファイルを提供するには（`ParameterOverrides` パラメータを参照。）
  - 使用するターゲットアカウントファイルを提供するには（`DeploymentTargets` パラメータを参照。）

出力アーティファクト

- アーティファクトの数: `0`
- 説明: 出力アーティファクトは、このアクションタイプには適用されません。

出力変数

このアクションを設定すると、パイプライン内のダウンストリームアクションのアクション設定によって参照できる変数が生成されます。名前空間を使用してアクションを設定し、これらの変数をダウンストリームアクションの設定で使用できるようにします。

- StackSetId: スタックセットの ID。
- OperationId: スタックセットオペレーションの ID。

詳細については、「変数リファレンス」を参照してください。

アクション設定の例

次の例は、CloudFormationStackInstances アクションのアクション設定を示しています。

自己管理型のアクセス許可モデルの例

次の例は、入力されたデプロイターゲットが AWS アカウント ID である CloudFormationStackInstances アクションを示しています111111222222。

YAML

```
Name: my-instances
ActionTypeId:
  Category: Deploy
```

```
  Owner: AWS
  Provider: CloudFormationStackInstances
  Version: '1'
RunOrder: 2
Configuration:
  DeploymentTargets: '111111222222'
  Regions: 'us-east-1,us-east-2,us-west-1,us-west-2'
  StackSetName: my-stackset
OutputArtifacts: []
InputArtifacts:
  - Name: SourceArtifact
Region: us-west-2
```

JSON

```
{
    "Name": "my-instances",
    "ActionTypeId": {
        "Category": "Deploy",
        "Owner": "AWS",
        "Provider": "CloudFormationStackInstances",
        "Version": "1"
    },
    "RunOrder": 2,
    "Configuration": {
        "DeploymentTargets": "111111222222",
        "Regions": "us-east-1,us-east-2,us-west-1,us-west-2",
        "StackSetName": "my-stackset"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "SourceArtifact"
        }
    ],
    "Region": "us-west-2"
}
```

サービス管理型のアクセス許可モデルの例

次の例は、デプロイターゲットが AWS Organizations 組織単位 ID であるサービスマネージドアクセス許可モデルの CloudFormationStackInstances アクションを示していますou-1111-1example。

YAML

```
Name: Instances
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Provider: CloudFormationStackInstances
  Version: '1'
RunOrder: 2
Configuration:
  DeploymentTargets: ou-1111-1example
  Regions: us-east-1
  StackSetName: my-stackset
OutputArtifacts: []
InputArtifacts:
  - Name: SourceArtifact
Region: eu-central-1
```

JSON

```
{
    "Name": "Instances",
    "ActionTypeId": {
        "Category": "Deploy",
        "Owner": "AWS",
        "Provider": "CloudFormationStackInstances",
        "Version": "1"
    },
    "RunOrder": 2,
    "Configuration": {
        "DeploymentTargets": "ou-1111-1example",
        "Regions": "us-east-1",
        "StackSetName": "my-stackset"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "SourceArtifact"
        }
    ],
    "Region": "eu-central-1"
}
```

サービスロールのアクセス許可: CloudFormationStackSet アクション

CloudFormation StackSets アクションには、次の最小限のアクセス許可が必要です。

CloudFormationStackSet のアクションについては、以下をポリシーステートメントに追加します。

```
{
    "Effect": "Allow",
    "Action": [
        "cloudformation:CreateStackSet",
        "cloudformation:UpdateStackSet",
        "cloudformation:CreateStackInstances",
        "cloudformation:DescribeStackSetOperation",
        "cloudformation:DescribeStackSet",
        "cloudformation:ListStackInstances"
    ],
    "Resource": "resource_ARN"
},
```

サービスロールのアクセス許可: CloudFormationStackInstances アクション

CloudFormationStackInstances のアクションについては、以下をポリシーステートメントに追加します。

```
{
    "Effect": "Allow",
    "Action": [
        "cloudformation:CreateStackInstances",
        "cloudformation:DescribeStackSetOperation"
    ],
    "Resource": "resource_ARN"
},
```

スタックセットオペレーションのアクセス許可モデル

CloudFormation StackSets は複数のアカウントでオペレーションを実行するため、スタックセットを作成する前に、それらのアカウントで必要なアクセス許可を定義する必要があります。アクセス許可は、自己管理型のアクセス許可またはサービス管理型のアクセス許可を使用して定義できます。

自己管理アクセス許可を使用して、StackSets で必要な 2 つの IAM ロールを作成します。これは、スタックセットを定義するアカウント内の AWSCloudFormationStackSetAdministrationRole などの管理者ロールと、スタックセットインスタンスをデプロイする各アカウントの AWSCloudFormationStackSetExecutionRole などの実行ロールです。このアクセス許可モデルを使用すると、StackSets は、ユーザーが IAM ロールを作成するアクセス許可を持つ任意の AWS アカウントにデプロイできます。詳細については、[AWS CloudFormation ユーザーガイド] の「自己管理型のアクセス許可の承認」を参照してください。

> **Note**
>
> CloudFormation StackSets は複数のアカウントでオペレーションを実行するため、スタックセットを作成する前に、それらのアカウントで必要なアクセス許可を定義する必要があります。

サービスマネージド型のアクセス許可を使用すると、 AWS Organizations が管理するアカウントにスタックインスタンスをデプロイできます。このアクセス許可モデルを使用すると、必要な IAM ロールを作成する必要はありません。ユーザーに代わって StackSets が IAM ロールを作成します。このモデルでは、将来組織に追加されるアカウントへの自動デプロイを有効にすることもできます。AWS CloudFormation 「 ユーザーガイド」のAWS 「Organizations で信頼されたアクセスを有効にする」を参照してください。

テンプレートパラメータのデータタイプ

スタックセットオペレーションで使用されるテンプレートパラメータには、次のデータタイプが含まれます。詳細については、「DescribeStackset」を参照してください。

ParameterKey

- 説明: パラメータに関連付けられたキー。特定のパラメータにキーと値を指定しない場合、 はテンプレートで指定されたデフォルト値 AWS CloudFormation を使用します。
- 例:

```
"ParameterKey=BucketName,ParameterValue=amzn-s3-demo-source-bucket"
```

ParameterValue

- 説明: パラメータに関連付けられた入力値。
- 例:

```
"ParameterKey=BucketName,ParameterValue=amzn-s3-demo-source-bucket"
```

UsePreviousValue

- スタックの更新中に、スタックが特定のパラメータキーに使用している既存のパラメータ値を使用します。`true` を指定した場合は、パラメータ値を指定しないでください。
- 例:

```
"ParameterKey=Asset1,UsePreviousValue=true"
```

各スタックセットには、1 つのテンプレートとテンプレートパラメータのセットがあります。テンプレートまたはテンプレートパラメータを更新すると、セット全体のパラメータが更新されます。次に、変更がそのインスタンスにデプロイされるまですべてのインスタンスステータスが OUTDATED に設定されます。

特定のインスタンスのパラメータ値をオーバーライドするには、たとえばテンプレートに `stage` のパラメータが `prod` の値で含まれている場合、そのパラメータ値を `beta` または `gamma` にオーバーライドできます。

関連情報

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- パラメータタイプ - [AWS CloudFormation ユーザーガイド] 内のこのリファレンスチャプターでは、CloudFormation テンプレートパラメータの詳細と例をより詳しく提供します。
- ベストプラクティス - スタックセットのデプロイのベストプラクティスの詳細については、AWS CloudFormation ユーザーガイドの「https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/stacksets-bestpractices.html」を参照してください。
- AWS CloudFormation API リファレンス – スタックセットオペレーションで使用されるパラメータの詳細については、 AWS CloudFormation API リファレンスの次の CloudFormation アクションを参照してください。

  - CreateStackSet アクションでスタックセットを作成します。
  - UpdateStackSet アクションで、指定されたアカウントおよびリージョン内のスタックセットおよび関連するスタックインスタンスを更新します。スタックセットの更新で作成されたスタックセットオペレーションが (完全または部分的に、指定されたフォルトトレランス値以下または以上となり) 失敗しても、スタックセットはこれらの変更で更新されます。指定されたスタック

セットに対する後続の CreateStackInstances 呼び出しでは、更新されたスタックセットが使用されます。

- CreateStackInstances アクションで、自己管理アクセス許可モデルで指定されたすべてのアカウント内、またはサービス管理アクセス許可モデルで指定されたすべてのデプロイターゲット内に、指定されたすべてのリージョンのスタックインスタンスを作成します。このアクションによって作成されたインスタンスのパラメータをオーバーライドできます。インスタンスがすでに存在する場合、CreateStackInstances は同じ入力パラメータで UpdateStackInstances を呼び出します。このアクションを使用してインスタンスを作成しても、他のスタックインスタンスのステータスは変更されません。
- UpdateStackInstances アクションで、自己管理アクセス許可モデルで指定されたすべてのアカウント内、またはサービス管理アクセス許可モデルで指定されたすべてのデプロイターゲット内で、指定されたすべてのリージョンのスタックセットによってスタックインスタンスを最新にします。このアクションによって更新されたインスタンスのパラメータをオーバーライドできます。このアクションを使用してインスタンスのサブセットを更新しても、他のスタックインスタンスのステータスは変更されません。
- DescribeStackSetOperation アクションで、指定されたスタックセットオペレーションの説明を返します。
- DescribeStackSet アクションで、指定されたスタックセットの説明を返します。

AWS CodeBuild ビルドおよびテストアクションリファレンス

パイプラインの一部としてビルドとテストを実行できます。CodeBuild のビルドまたはテストアクションを実行すると、buildspec で指定されたコマンドが CodeBuild コンテナ内で実行されます。CodeBuild アクションへの入力アーティファクトとして指定されたすべてのアーティファクトは、コマンドを実行するコンテナ内で使用できます。CodeBuild は、ビルドまたはテストアクションのいずれかを提供することができます。詳細については、「AWS CodeBuild ユーザーガイド」を参照してください。

コンソールの CodePipeline ウィザードを使用してビルドプロジェクトを作成すると、CodeBuild のビルドプロジェクトにはソースプロバイダが CodePipeline であることが表示されます。CodeBuild コンソールでビルドプロジェクトを作成する場合、ソースプロバイダとして CodePipeline を指定することはできませんが、パイプラインにビルドアクションを追加すると CodeBuild コンソールでソースが調整されます。詳細については、[AWS CodeBuild API リファレンス] の [ProductInformation] を参照してください。

トピック

- アクションタイプ
- 設定パラメータ
- 入力アーティファクト
- 出力アーティファクト
- 出力変数
- サービスロールのアクセス許可: CodeBuild アクション
- アクション宣言 (CodeBuild の例)
- 関連情報

アクションタイプ

- カテゴリ: `Build` または `Test`
- 所有者: `AWS`
- プロバイダー: `CodeBuild`
- バージョン: `1`

設定パラメータ

ProjectName

必須: はい

`ProjectName` は、CodeBuild のビルドプロジェクト名です。

PrimarySource

必須: 条件による

`PrimarySource` パラメータの値は、アクションへの入力アーティファクトの名前の 1 つでなければなりません。CodeBuild は、このアーティファクトを解凍したものを含むディレクトリで buildspec のファイルを探し、buildspec コマンドを実行します。

このパラメータは、CodeBuild アクションに複数の入力アーティファクトが指定されている場合に必要となります。アクションのソースアーティファクトが 1 つだけの場合、`PrimarySource` アーティファクトはデフォルトでそのアーティファクトになります。

BatchEnabled

必須: いいえ

この BatchEnabled パラメータのブール値は、アクションが同じビルド実行で複数のビルドを実行することを可能にします。

このオプションを有効にすると、CombineArtifacts オプションが使用できます。

バッチビルドを有効にしたパイプラインの例については、「CodePipeline と CodeBuild およびバッチビルドの統合」を参照してください。

BuildspecOverride

必須: いいえ

ビルドプロジェクトのこのビルドに定義された最新バージョンを上書きするインライン buildspec 定義または buildspec ファイル宣言。プロジェクトで定義された buildspec は変更されません。

この値は、次のいずれかに設定できます。

- インライン buildspec 定義。詳細については、「Buildspec の構文」の構文リファレンスを参照してください。
- 組み込みの環境変数 CODEBUILD_SRC_DIR の値に相対的な代替 buildspec ファイルへのパスか、S3 バケットへのパス。バケットは、ビルドプロジェクトと同じ AWS リージョン に存在する必要があります。ARN を使用して buildspec ファイルを指定します (例: arn:aws:s3:::my-codebuild-sample2/buildspec.yml) 。この値が指定されていない場合や、空の文字列に設定されている場合、ソースコードのルートディレクトリに buildspec ファイルが含まれている必要があります。パスの追加の詳細については、「buildspec ファイル名とストレージの場所」を参照してください。

> **Note**
>
> このプロパティを使用すると、コンテナで実行されるビルドコマンドを変更できるため、この API を呼び出してこのパラメータを設定する権限を持つ IAM プリンシパルがデフォルト設定を上書きできることに注意してください。さらに、ソースリポジトリのファイルや Amazon S3 バケットなどの信頼できる buildspec の場所を使用することをお勧めします。

CombineArtifacts

必須: いいえ

この CombineArtifacts パラメータのブール値は、バッチ ビルドからのすべてのビルド アーティファクトをビルド アクションのための単一の アーティファクト ファイルにまとめます。

このオプションを使用するには、BatchEnabled パラメータを有効にする必要があります。

EnvironmentVariables

必須: いいえ

このパラメータの値は、パイプラインの CodeBuild アクションの環境変数を設定するために使用されます。EnvironmentVariables パラメータの値は、環境変数オブジェクトの JSON 配列の形式をとります。「アクション宣言 (CodeBuild の例)」のパラメータ例を参照してください。

各オブジェクトには 3 つの部分があり、それらはすべて文字列です。

- name: 環境変数の名前またはキー。
- value: 環境変数の値。PARAMETER_STORE または SECRETS_MANAGERタイプを使用する場合、この値は AWS Systems Manager パラメータストアに既に保存されているパラメータの名前、または Secrets Manager に AWS 既に保存されているシークレットの名前である必要があります。

> **Note**
>
> 機密情報、特に AWS 認証情報を保存するために環境変数を使用することは強くお勧めしません。CodeBuild コンソールまたは CLI AWS を使用すると、環境変数がプレーンテキストで表示されます。機密の値の場合は、代わりに SECRETS_MANAGER タイプを使用することをお勧めします。

- type: (任意) 環境変数の型。有効な値は PARAMETER_STORE、SECRETS_MANAGER、または PLAINTEXT です。指定しない場合、この値はデフォルトで PLAINTEXT になります。

> **Note**
>
> 環境変数の設定に name、value、および type を入力する場合 (特に環境変数に CodePipeline の出力変数の構文が含まれている場合) は、設定の値フィールドの 1000 文字制限を超えないようにしてください。この制限を超えると、検証エラーが返されます。

詳細については、 AWS CodeBuild API リファレンスのEnvironmentVariable」を参照してください。GitHub ブランチ名に解決される環境変数を持つ CodeBuild アクションの例については、例:CodeBuild 環境変数で BranchName 変数を使用する を参照してください。

ServiceRoleArnOverride

必須: いいえ

CodeBuild が、このアクションによってトリガーされるビルドのサービスロールとして使用し、CodeBuild プロジェクトで設定されたデフォルトのサービスロールを上書きする IAM ロール ARN。存在しない場合、CodeBuild プロジェクトはデフォルトのサービスロールを使用します。

指定されたロールには、`codebuild.amazonaws.com`サービスが引き受けることを許可する信頼ポリシーが必要です。パイプラインの作成または更新に使用される ID とパイプラインサービスロールの両方に、指定されたロールに対する`iam:PassRole`アクセス許可が必要です。ポリシーの例については「サービスロールのアクセス許可: CodeBuild アクション」を参照してください。

セキュリティのベストプラクティスとして、CodeBuild プロジェクトを複数のパイプライン間で共有する場合は、最小特権の原則に従って、パイプラインごとにスコープ付きロールを指定することをお勧めします。CodeConnections ソースアクションを使用するパイプラインについては、特定のリポジトリへのロールの接続アクセス許可の範囲設定の例Bitbucket、GitHub、GitHub Enterprise Server、または GitLab.com に接続するための CodeBuild GitClone アクセス許可を追加します。については、「」を参照してください。

> **Note**
>
> ロールは CodeBuild プロジェクトと同じ AWS アカウントに存在する必要があります。

入力アーティファクト

- アーティファクトの数: `1 to 5`
- 説明: CodeBuild は buildspec ファイルを探し、プライマリソースアーティファクトのディレクトリから buildspec コマンドを実行します。単一の入力ソースを指定する場合、または CodeBuild アクションに複数の入力ソースを指定する場合、単一のアーティファクト、または複数の入力ソースの場合はプライマリアーティファクトを CodePipeline の `PrimarySource` アクション設定パラメータを使用して設定する必要があります。

各入力アーティファクトは独自のディレクトリに抽出され、その場所は環境変数に保存されます。プライマリソースアーティファクトのディレクトリは `$CODEBUILD_SRC_DIR` で使用できるようになります。他のすべての入力アーティファクトのディレクトリは、`$CODEBUILD_SRC_DIR_yourInputArtifactName` で使用できるようになります。

> **Note**
>
> CodeBuild プロジェクトで設定されたアーティファクトは、パイプラインの CodeBuild アクションで使用される入力アーティファクトになります。

出力アーティファクト

- アーティファクトの数: `0 to 5`
- 説明: これらは、CodeBuild の buildspec ファイルで定義されたアーティファクトを、パイプラインの後続のアクションで利用できるようにするために使用します。出力アーティファクトが 1 つだけ定義されている場合、このアーティファクトは buildspec の `artifacts` セクションの直下で定義できます。複数の出力アーティファクトを指定する場合、参照されるすべてのアーティファクトは、buildspec ファイルでセカンダリアーティファクトとして定義する必要があります。CodePipeline の出力アーティファクトの名前は、buildspec ファイルのアーティファクト識別子と一致しなければなりません。

> **Note**
>
> CodeBuild プロジェクトで設定されたアーティファクトは、パイプラインアクションの CodePipeline 入力アーティファクトになります。

バッチビルド用に `CombineArtifacts` パラメータが選択されている場合、出力アーティファクトの場所には、同じ実行で実行された複数のビルドから結合されたアーティファクトが含まれます。

出力変数

このアクションは、ビルドの一部としてエクスポートされたすべての環境変数を変数として生成します。環境変数をエクスポートする方法の詳細については、AWS CodeBuild API ガイドの「EnvironmentVariable」を参照してください。

CodePipeline で CodeBuild 環境変数を使用する方法については、CodeBuild アクションの出力変数の例を参照してください。CodeBuild で使用できる環境変数のリストについては、AWS CodeBuild ユーザーガイドの「ビルド環境の環境変数」を参照してください。

サービスロールのアクセス許可: CodeBuild アクション

CodeBuild を対応すべく、以下をポリシーステートメントに追加します。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "codebuild:BatchGetBuilds",
                "codebuild:StartBuild",
                "codebuild:BatchGetBuildBatches",
                "codebuild:StartBuildBatch"
            ],
            "Resource": [
                "arn:aws:codebuild:*:111122223333:project/[[ProjectName]]"
            ],
            "Effect": "Allow"
        }
    ]
}
```

`ServiceRoleArnOverride` パラメータを使用する場合は、次の`iam:PassRole`アクセス許可を追加して、CodePipeline が CodeBuild にロールを渡すことを許可します。

```
{
    "Effect": "Allow",
```

```
    "Action": "iam:PassRole",
    "Resource": "arn:aws:iam::111122223333:role/your-codebuild-override-role",
    "Condition": {
        "StringEquals": {
            "iam:PassedToService": "codebuild.amazonaws.com"
        }
    }
}
```

アクション宣言（CodeBuild の例）

YAML

```
Name: Build
Actions:
  - Name: PackageExport
    ActionTypeId:
      Category: Build
      Owner: AWS
      Provider: CodeBuild
      Version: '1'
    RunOrder: 1
    Configuration:
      BatchEnabled: 'true'
      CombineArtifacts: 'true'
      ProjectName: my-build-project
      PrimarySource: MyApplicationSource1
      EnvironmentVariables:
 '[{"name":"TEST_VARIABLE","value":"TEST_VALUE","type":"PLAINTEXT"},
{"name":"ParamStoreTest","value":"PARAMETER_NAME","type":"PARAMETER_STORE"}]'
      ServiceRoleArnOverride: 'arn:aws:iam::111122223333:role/my-scoped-codebuild-
role'
    OutputArtifacts:
      - Name: MyPipeline-BuildArtifact
    InputArtifacts:
      - Name: MyApplicationSource1
      - Name: MyApplicationSource2
```

JSON

```
{
```

```
    "Name": "Build",
    "Actions": [
        {
            "Name": "PackageExport",
            "ActionTypeId": {
                "Category": "Build",
                "Owner": "AWS",
                "Provider": "CodeBuild",
                "Version": "1"
            },
            "RunOrder": 1,
            "Configuration": {
                "BatchEnabled": "true",
                "CombineArtifacts": "true",
                "ProjectName": "my-build-project",
                "PrimarySource": "MyApplicationSource1",
                "EnvironmentVariables": "[{\"name\":\"TEST_VARIABLE\",\"value\":
\"TEST_VALUE\",\"type\":\"PLAINTEXT\"},{\"name\":\"ParamStoreTest\",\"value\":
\"PARAMETER_NAME\",\"type\":\"PARAMETER_STORE\"}]",
                "ServiceRoleArnOverride": "arn:aws:iam::111122223333:role/my-scoped-
codebuild-role"
            },
            "OutputArtifacts": [
                {
                    "Name": "MyPipeline-BuildArtifact"
                }
            ],
            "InputArtifacts": [
                {
                    "Name": "MyApplicationSource1"
                },
                {
                    "Name": "MyApplicationSource2"
                }
            ]
        }
    ]
}
```

関連情報

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- AWS CodeBuild ユーザーガイド – CodeBuild アクションを使用したパイプラインの例については、「CodeBuild で CodeBuild CodePipeline を使用してコードをテストし、ビルドを実行する」を参照してください。CodeBuild のアーティファクトを複数入力· 出力するプロジェクトの例については、[CodeBuild および複数の入力ソースと出力アーティファクトのサンプルと CodePipeline の統合] および [複数の入力ソースと出力アーティファクトのサンプル] を参照してください。
- チュートリアル: を使用して Android アプリを構築およびテストするパイプラインを作成する AWS Device Farm - このチュートリアルでは、CodeBuild と AWS Device Farmを使用して Android アプリケーションをビルドおよびテストするパイプラインを GitHub ソースで作成するためのサンプル buildspec ファイルとサンプルアプリケーションを提供します。
- CodeBuild のビルド仕様リファレンス - このリファレンストピックでは、CodeBuild の buildspec ファイルを理解するための定義と例を提供します。CodeBuild で使用できる環境変数のリストについては、AWS CodeBuild ユーザーガイドの「ビルド環境の環境変数」を参照してください。

AWS CodePipeline アクションリファレンスを呼び出す

CodePipeline 呼び出しアクションを使用して、ダウンストリームパイプライン実行のトリガーと、パイプライン変数とソースリビジョンをパイプライン間で渡す処理を簡素化します。

> **Note**
>
> このアクションは V2 タイプのパイプラインでのみサポートされています。

トピック

- アクションタイプ
- 設定パラメータ
- 入力アーティファクト
- 出力アーティファクト
- CodePipeline 呼び出しアクションのサービスロールポリシーのアクセス許可
- アクションの宣言
- 関連情報

アクションタイプ

- カテゴリ:`Invoke`

- 所有者: AWS
- プロバイダー: CodePipeline
- バージョン: 1

設定パラメータ

PipelineName

必須: はい

実行中に現在のターゲットパイプラインを開始するパイプラインの名前。呼び出しパイプラインを既に作成している必要があります。my-s3-pipeline という名前の (呼び出し) パイプラインが実行を開始すると、アクションは s3-pipeline-test (ターゲット) パイプラインを開始します。

SourceRevisions

必須: いいえ

呼び出しパイプラインによって開始されたときにターゲットパイプラインで使用するソースリビジョン。例えば、S3 ソースアクションは、S3 バージョン ID やオブジェクトキーなどの出力変数を提供します。パイプラインが呼び出されるときに使用するリビジョン値を指定できます。

CLI では、ソースリビジョンをシリアル化された JSON 文字列として指定します。ソースリビジョンオーバーライドの使用の詳細については、「CodePipeline API ガイド」の「SourceRevisionOverride」を参照してください。

マッピングでは、次の例に示すように文字列形式を使用します。

```
[{"actionName":"Source","revisionType":"S3_OBJECT_VERSION_ID","revision
Value":"zq8mjNEXAMPLE"}]
```

[変数]

必須: いいえ

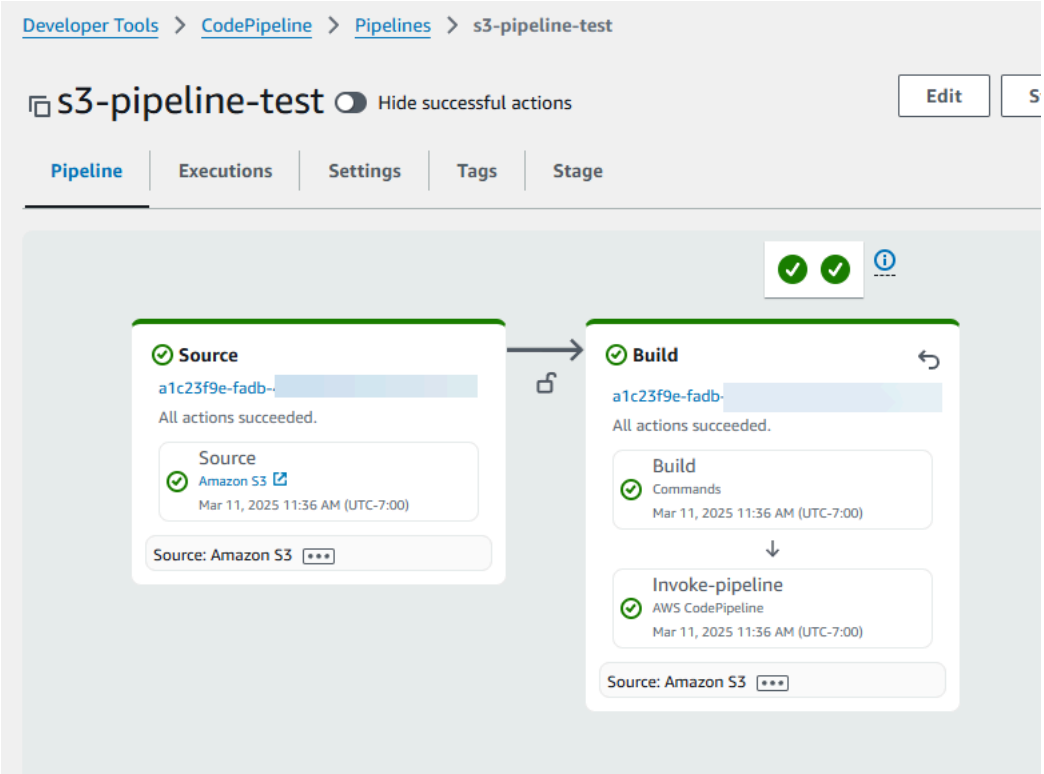
アクションでサポートする変数の名前と値。

CLI では、シリアル化された JSON 文字列として変数を指定します。パイプライン変数の使用の詳細については、「CodePipeline API ガイド」の「PipelineVariable」を参照してください。

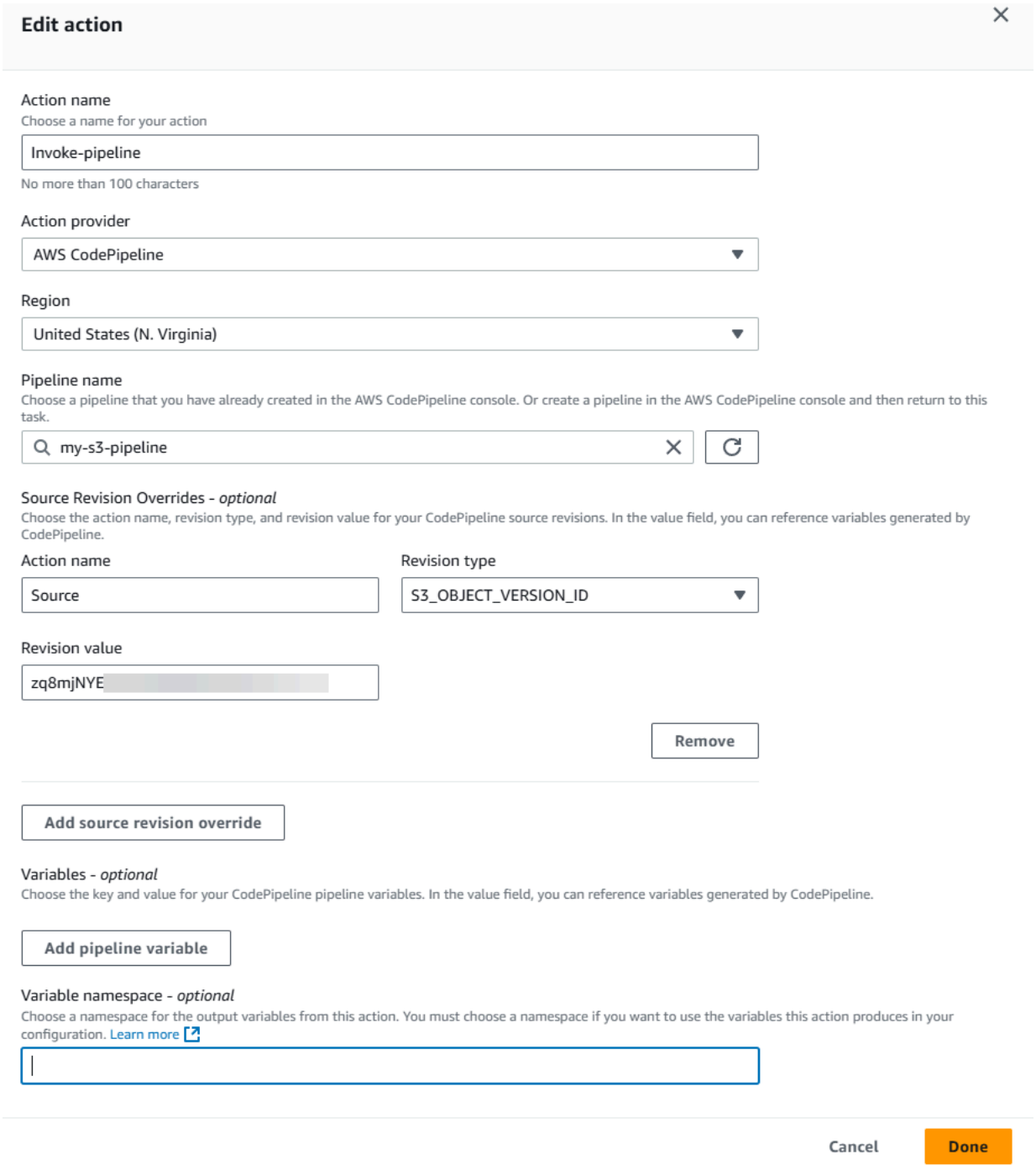
マッピングでは、次の例に示すように文字列形式を使用します。

```
[{"name":"VAR1","value":"VALUE1"}]
```

次の図は、コンソールでパイプラインに追加されたアクションの例を示しています。



次の画像は、アクションの [編集] ページの例を示しています。次の例では、s3-pipeline-test という名前のパイプラインに、コンソールに示されているように設定されたパイプライン呼び出しアクションがあります。my-s3-pipeline という名前のパイプラインの実行が完了すると、アクションによって s3-pipeline-test パイプラインが開始されます。この例では、指定されたリビジョン値が zq8mjNYEexample の S3_OBJECT_VERSION_ID ソースオーバーライドのソースリビジョンオーバーライドを示しています。



入力アーティファクト

- アーティファクトの数: 0
- 説明: 入力アーティファクトは、このアクションタイプには適用されません。

出力アーティファクト

- アーティファクトの数: `0`
- 説明: 出力アーティファクトは、このアクションタイプには適用されません。

CodePipeline 呼び出しアクションのサービスロールポリシーのアクセス許可

CodePipeline がアクションを実行する際、CodePipeline サービスロールポリシーには、最小特権でアクセスを維持するために、パイプラインリソース ARN に適切にスコープダウンされた `codepipeline:StartPipelineExecution` アクセス許可が必要です。

```
{
           "Sid": "StatementForPipelineInvokeAction",
           "Effect": "Allow",
           "Action": "codepipeline:StartPipelineExecution",
           "Resource": [
               "arn:aws:codepipeline:{{region}}:{{AccountId}}:{{pipelineName}}"
           ]
       }
```

アクションの宣言

YAML

```
name: Invoke-pipeline
actionTypeId:
  category: Invoke
  owner: AWS
  provider: CodePipeline
  version: '1'
runOrder: 2
configuration:
  PipelineName: my-s3-pipeline
  SourceRevisions:
 '[{"actionName":"Source","revisionType":"S3_OBJECT_VERSION_ID","revision
Value":"zq8mjNEXAMPLE"}]'
  Variables: '[{"name":"VAR1","value":"VALUE1"}]'
```

JSON

```
{
    "name": "Invoke-pipeline",
    "actionTypeId": {
        "category": "Invoke",
        "owner": "AWS",
        "provider": "CodePipeline",
        "version": "1"
    },
    "runOrder": 2,
    "configuration": {
        "PipelineName": "my-s3-pipeline",
        "SourceRevisions": "[{\"actionName\":\"Source\",\"revisionType\":
\"S3_OBJECT_VERSION_ID\",\"revisionValue\":\"zq8mjNEXAMPLE\"}]",
        "Variables": "[{\"name\":\"VAR1\",\"value\":\"VALUE1\"}]"
    }
},
```

関連情報

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- ソースリビジョンオーバーライドでパイプラインを開始する – このセクションでは、手動で、または EventBridge イベント入力トランスフォーマーを使用して、ソースリビジョンでパイプラインを開始する方法について説明します。

CodeCommit ソースアクションリファレンス

定された CodeCommit リポジトリとブランチで新しいコミットが行われると、パイプラインがスタートします。

コンソールを使用してパイプラインを作成または編集する場合、CodePipeline はリポジトリで変更が生じた場合にパイプラインを開始する EventBridge ルールを作成します。

> **Note**
>
> Amazon ECR、Amazon S3、または CodeCommit ソースの場合、入力変換エントリを使用してソースオーバーライドを作成し、パイプラインイベントの EventBridge で `revisionValue` を使用することもできます。ここで、`revisionValue` はオブジェクト

キー、コミット、またはイメージ ID のソースイベント変数から派生します。詳細については、Amazon ECR ソースアクションと EventBridge リソース、イベントに対して有効にしたソースを使用して Amazon S3 ソースアクションに接続する、または CodeCommit ソースアクションと EventBridge の手順に含まれる入力変換エントリのオプション手順を参照してください。

CodeCommit アクションを使用してパイプラインを接続する前に、CodeCommit リポジトリを作成しておく必要があります。

コードの変更が検出された後は、後続のアクションにコードを渡すための次のオプションがあります。

- [デフォルト] — CodeCommit ソースアクションが、コミットの浅いコピーを含む ZIP ファイルを出力するように設定します。
- [フルクローン] — ソースアクションが、後続のアクションのためにリポジトリへの Git URL リファレンスを出力するように設定します。

  現在、Git URL リファレンスは、リポジトリと関連する Git メタデータをクローンするためにダウンストリーム CodeBuild アクションでのみ使用できます。Git URL リファレンスを CodeBuild 以外のアクションに渡そうとすると、エラーが発生します。

トピック

- アクションタイプ
- 設定パラメータ
- 入力アーティファクト
- 出力アーティファクト
- 出力変数
- サービスロールのアクセス許可: CodeCommit アクション
- アクション設定の例
- 関連情報

アクションタイプ

- カテゴリ:`Source`

- 所有者: `AWS`
- プロバイダー: `CodeCommit`
- バージョン: `1`

設定パラメータ

RepositoryName

必須: はい

ソースの変更が検出されるリポジトリの名前。

BranchName

必須: はい

ソースの変更が検出されるブランチの名前。

PollForSourceChanges

必須: いいえ

`PollForSourceChanges` は、CodePipeline がソースの変更について CodeCommit リポジトリをポーリングするかどうかを制御します。代わりに CloudWatch Events を使用してソースの変更を検出することをお勧めします。CloudWatch Events の構成については、ポーリングパイプラインを移行する (CodeCommit ソース) (CLI) または ポーリングパイプラインの移行 (CodeCommit ソース) (CloudFormation テンプレート) を参照してください。

> **Important**
>
> CloudWatch Events ルールを設定する場合、パイプラインの重複実行を避けるために `PollForSourceChanges` を `false` に設定する必要があります。

このパラメータの有効な値:

- `true`: 設定されている場合、CodePipeline はソースの変更についてポーリングします。

> **Note**
>
> PollForSourceChanges を省略した場合、CodePipeline はデフォルトでソースの変更についてリポジトリをポーリングします。この動作は、PollForSourceChanges が含まれており、true に設定されている場合と同じです。

- false: 設定されている場合、CodePipeline は、ソースの変更についてリポジトリをポーリングしません。CloudWatch Events ルールを設定してソース変更を検出する場合は、この設定を使用します。

OutputArtifactFormat

必須: いいえ

出力 アーティファクト フォーマット。値は CODEBUILD_CLONE_REF または CODE_ZIP のいずれかです。指定しない場合、デフォルトの CODE_ZIP が使用されます。

> **Important**
>
> CODEBUILD_CLONE_REF のオプションは、CodeBuild のダウンストリームアクションでのみ使用可能です。
> このオプションを選択する場合、codecommit:GitPull に示すように、CodeBuild サービスロールに CodeBuild GitClone のアクセス権限を CodeCommit ソースアクションに追加します。 許可を追加する必要があります。また、codecommit:GetRepository に示すように、CodePipeline のサービス· ロールに CodePipeline サービスロールにアクセス許可を追加する 許可を追加する必要もあります。[フルクローン] オプションを使用する方法を示すチュートリアルについては、チュートリアル: CodeCommit パイプラインソースでフルクローンを使用する を参照してください。

入力アーティファクト

- アーティファクトの数: 0
- 説明: 入力アーティファクトは、このアクションタイプには適用されません。

出力アーティファクト

- アーティファクトの数: 1

- 説明: このアクションの出力アーティファクトは、パイプライン実行のソースリビジョンとして指定されたコミットで設定されたリポジトリとブランチの内容を含む ZIP ファイルです。リポジトリから生成されるアーティファクトは、CodeCommit アクションの出力アーティファクトです。ソースコードのコミット ID は、パイプライン実行のトリガーとなるソースリビジョンとして、CodePipeline に表示されます。

出力変数

このアクションを設定すると、パイプライン内のダウンストリームアクションのアクション設定によって参照できる変数が生成されます。このアクションは、アクションに名前空間がない場合でも、出力変数として表示できる変数を生成します。名前空間を使用してアクションを設定し、これらの変数をダウンストリームアクションの設定で使用できるようにします。

詳細については、「変数リファレンス」を参照してください。

CommitId

パイプライン実行のトリガーとなった CodeCommit のコミット ID。コミット ID は、コミットの完全な SHA です。

CommitMessage

パイプライン実行をトリガーしたコミットに関連付けられた説明メッセージ (存在する場合)。

RepositoryName

パイプラインのトリガーとなるコミットが行われた CodeCommit リポジトリの名前。

BranchName

ソース変更が行われた CodeCommit リポジトリのブランチ名。

AuthorDate

コミットが認証された日付 (タイムスタンプ形式)。

CommitterDate

コミットがコミットされた日付 (タイムスタンプ形式)。

サービスロールのアクセス許可: CodeCommit アクション

CodePipeline がアクションを実行する際、CodePipeline サービスロールポリシーには、最小特権でアクセスを維持するために、パイプラインリソース ARN に適切にスコープダウンされた次のアクセス許可が必要です。例えば、以下をポリシーステートメントに追加します。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codecommit:CancelUploadArchive",
                "codecommit:GetBranch",
                "codecommit:GetCommit",
                "codecommit:GetRepository",
                "codecommit:GetUploadArchiveStatus",
                "codecommit:UploadArchive"
            ],
            "Resource": [
                "arn:aws:codecommit:*:111122223333:[[codecommitRepostories]]"
            ]
        }
    ]
}
```

アクション設定の例

デフォルトの出力アーティファクト フォーマットの例

YAML

```
name: Source
actionTypeId:
  category: Source
  owner: AWS
  provider: CodeCommit
  version: '1'
```

```
runOrder: 1
configuration:
  BranchName: main
  PollForSourceChanges: 'false'
  RepositoryName: MyWebsite
outputArtifacts:
  - name: Artifact_MyWebsiteStack
inputArtifacts: []
region: us-west-2
namespace: SourceVariables
```

JSON

```
{
    "name": "Source",
    "actionTypeId": {
        "category": "Source",
        "owner": "AWS",
        "provider": "CodeCommit",
        "version": "1"
    },
    "runOrder": 1,
    "configuration": {
        "BranchName": "main",
        "PollForSourceChanges": "false",
        "RepositoryName": "MyWebsite"
    },
    "outputArtifacts": [
        {
            "name": "Artifact_MyWebsiteStack"
        }
    ],
    "inputArtifacts": [],
    "region": "us-west-2",
    "namespace": "SourceVariables"
}
```

フル クローン出力アーティファクト フォーマットの例

YAML

```
name: Source
```

```
actionTypeId:
  category: Source
  owner: AWS
  provider: CodeCommit
  version: '1'
runOrder: 1
configuration:
  BranchName: main
  OutputArtifactFormat: CODEBUILD_CLONE_REF
  PollForSourceChanges: 'false'
  RepositoryName: MyWebsite
outputArtifacts:
  - name: SourceArtifact
inputArtifacts: []
region: us-west-2
namespace: SourceVariables
```

JSON

```
{
    "name": "Source",
    "actionTypeId": {
        "category": "Source",
        "owner": "AWS",
        "provider": "CodeCommit",
        "version": "1"
    },
    "runOrder": 1,
    "configuration": {
        "BranchName": "main",
        "OutputArtifactFormat": "CODEBUILD_CLONE_REF",
        "PollForSourceChanges": "false",
        "RepositoryName": "MyWebsite"
    },
    "outputArtifacts": [
        {
            "name": "SourceArtifact"
        }
    ],
    "inputArtifacts": [],
    "region": "us-west-2",
    "namespace": "SourceVariables"
}
```

関連情報

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- チュートリアル: シンプルなパイプラインを作成する (CodeCommit リポジトリ) – このチュートリアルでは、サンプル AppSpec ファイルとサンプル CodeDeploy アプリケーションおよびデプロイグループを提供します。このチュートリアルを参照して、Amazon EC 2 インスタンスにデプロイする CodeCommit ソースを持つパイプラインを作成します。

AWS CodeDeploy デプロイアクションリファレンス

AWS CodeDeploy アクションを使用して、アプリケーションコードをデプロイフリートにデプロイします。デプロイフリートは、Amazon EC2 インスタンス、オンプレミスインスタンス、またはその両方で構成することができます。

> **Note**
>
> このリファレンストピックでは、Amazon EC2 をデプロイプラットフォームとする CodePipeline の CodeDeploy デプロイアクションを説明します。CodePipeline における Amazon Elastic Container Service から CodeDeploy の blue/green デプロイアクションのリファレンス情報については、Amazon ECS および CodeDeploy ブルー/グリーンデプロイアクションリファレンス を参照してください。

トピック

- アクションタイプ
- 設定パラメータ
- 入力アーティファクト
- 出力アーティファクト
- サービスロールのアクセス許可: AWS CodeDeploy アクション
- アクションの宣言
- 関連情報

アクションタイプ

- カテゴリ:`Deploy`
- 所有者: `AWS`
- プロバイダー: `CodeDeploy`
- バージョン: `1`

設定パラメータ

ApplicationName

必須: はい

CodeDeploy で作成したアプリケーションの名前。

DeploymentGroupName

必須: はい

CodeDeploy で作成したデプロイメントグループ。

入力アーティファクト

- アーティファクトの数: `1`
- 説明: CodeDeploy が判断に使用する AppSpec ファイル
  - Amazon S3 や GitHub にある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 ファイルの詳細については、[CodeDeploy AppSpec ファイルリファレンス] を参照してください。

出力アーティファクト

- アーティファクトの数: `0`
- 説明: 出力アーティファクトは、このアクションタイプには適用されません。

サービスロールのアクセス許可: AWS CodeDeploy アクション

AWS CodeDeploy サポートを受けるには、ポリシーステートメントに以下を追加します。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codedeploy:CreateDeployment",
                "codedeploy:GetApplication",
                "codedeploy:GetDeployment",
                "codedeploy:RegisterApplicationRevision",
                "codedeploy:ListDeployments",
                "codedeploy:ListDeploymentGroups",
                "codedeploy:GetDeploymentGroup"
            ],
            "Resource": [
                "arn:aws:codedeploy:*:111122223333:application:
[[codedeployApplications]]",
                "arn:aws:codedeploy:*:111122223333:deploymentgroup:
[[codedeployApplications]]/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "codedeploy:GetDeploymentConfig"
            ],
            "Resource": [
                "arn:aws:codedeploy:*:111122223333:deploymentconfig:
[[deploymentConfigs]]"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "codedeploy:ListDeploymentConfigs"
            ],
            "Resource": [
```

```
                "*"
            ]
        }
    ]
}
```

アクションの宣言

YAML

```
Name: Deploy
Actions:
  - Name: Deploy
    ActionTypeId:
      Category: Deploy
      Owner: AWS
      Provider: CodeDeploy
      Version: '1'
    RunOrder: 1
    Configuration:
      ApplicationName: my-application
      DeploymentGroupName: my-deployment-group
    OutputArtifacts: []
    InputArtifacts:
      - Name: SourceArtifact
    Region: us-west-2
    Namespace: DeployVariables
```

JSON

```
{
    "Name": "Deploy",
    "Actions": [
        {
            "Name": "Deploy",
            "ActionTypeId": {
                "Category": "Deploy",
                "Owner": "AWS",
                "Provider": "CodeDeploy",
                "Version": "1"
            },
            "RunOrder": 1,
```

```
            "Configuration": {
                "ApplicationName": "my-application",
                "DeploymentGroupName": "my-deployment-group"
            },
            "OutputArtifacts": [],
            "InputArtifacts": [
                {
                    "Name": "SourceArtifact"
                }
            ],
            "Region": "us-west-2",
            "Namespace": "DeployVariables"
        }
    ]
},
```

関連情報

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- チュートリアル: シンプルなパイプラインを作成する (S3 バケット) — このチュートリアルでは、ソースバケット、EC2 インスタンス、および CodeDeploy リソースを作成して、サンプルアプリケーションをデプロイする手順について説明します。次に、S3 バケットに保持されているコードを Amazon EC2 インスタンスにデプロイする CodeDeploy デプロイアクションを使用してパイプラインを構築します。
- チュートリアル: シンプルなパイプラインを作成する (CodeCommit リポジトリ) — このチュートリアルでは、CodeCommit ソースリポジトリ、EC2 インスタンス、および CodeDeploy リソースを作成して、サンプルアプリケーションをデプロイする手順を説明します。次に、CodeCommit リポジトリから Amazon EC2 インスタンスにコードをデプロイする CodeDeploy デプロイアクションを使用してパイプラインを構築します。
- [CodeDeploy AppSpec ファイルリファレンス] — [AWS CodeDeploy ユーザーガイド] のこのリファレンスの章では CodeDeploy AppSpec ファイルのリファレンス情報と例について説明します。

CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアクションの場合)

接続のソースアクションは でサポートされています AWS CodeConnections。 AWS CodeConnections を使用すると、 AWS リソースと GitHub などのサードパーティーリポジトリ間の接続を作成および管理できます。サードパーティーのソースコードリポジトリで新しいコミットが行われたときに、パイプラインを開始します。ソースアクションは、パイプラインが手動で実行されたとき、またはソースプロバイダから webhook イベントが送信されたときに、コードの変更を取得します。

パイプラインのアクションを設定して、トリガーでパイプラインを開始できる Git 設定を使用できます。トリガーを使用してフィルタリングするようにパイプラインのトリガー設定を構成する方法の詳細については、「コードプッシュまたはプルリクエストイベントタイプを使用してトリガーを追加する」を参照してください。

> **Note**
>
> この機能は、アジアパシフィック (香港) 、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 (ジャカルタ) 、アジアパシフィック (メルボルン) 、アジアパシフィック (大阪) 、アフリカ (ケープタウン) 、中東 (バーレーン) 、中東 (アラブ首長国連邦) 、欧州 (スペイン) 、欧州 (チューリッヒ) 、イスラエル (テルアビブ) 、または AWS GovCloud (米国西部) の各リージョンでは使用できません。利用可能なその他のアクションについては、「CodePipeline との製品とサービスの統合」を参照してください。欧州 (ミラノ) リージョンでのこのアクションに関する考慮事項については、「CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアクションの場合)」の注意を参照してください。

接続では、 AWS リソースを次のサードパーティーリポジトリに関連付けることができます。

- Bitbucket Cloud (CodePipeline コンソールの Bitbucket プロバイダーオプションまたは CLI の `Bitbucket` プロバイダーを使用)

> **Note**
>
> Bitbucket Cloud リポジトリへの接続を作成できます。Bitbucket サーバーなど、インストールされている Bitbucket プロバイダーのタイプはサポートされていません。

-

> **Note**
>
> Bitbucket ワークスペースを使用している場合、接続を作成するには管理者アクセス権が必要です。

- GitHub および GitHub Enterprise Cloud (CodePipeline コンソールの GitHub (GitHub アプリ経由) プロバイダーオプションまたは CLI の `GitHub` プロバイダーを使用)

> **Note**
>
> リポジトリが GitHub 組織に属している場合、接続を作成するには組織の所有者であることが必要です。組織に属していないリポジトリを使用する場合は、リポジトリの所有者であることが必要です。

- GitHub Enterprise Server (CodePipeline コンソールの GitHub Enterprise Server プロバイダーオプションまたは CLI の `GitHub Enterprise Server` プロバイダーを使用)
- GitLab.com (CodePipeline コンソールの GitLab プロバイダーオプションまたは CLI の `GitLab` プロバイダーを使用)

> **Note**
>
> GitLab で、自分が所有者ロールを持っているリポジトリへの接続を作成すると、その接続を CodePipeline などのリソースを含むリポジトリで使用できます。グループ内のリポジトリでは、グループの所有者である必要はありません。

- GitLab (エンタープライズエディションまたはコミュニティエディション) のセルフマネージドインストール (CodePipeline コンソールの GitLab セルフマネージドプロバイダーオプションまたは CLI の `GitLabSelfManaged` プロバイダーを使用)

> **Note**
>
> 各接続は、そのプロバイダーのすべてのリポジトリをサポートします。プロバイダーの種類ごとに新しい接続を作成する必要があります。

Connections により、パイプラインはサードパーティープロバイダーのインストールアプリを通じてソースの変更を検出することができます。例えば、webhooks は GitHub イベントタイプのサブスクライブに使用され、組織、リポジトリ、または GitHub アプリにインストールできます。接続によって、GitHub プッシュタイプのイベントをサブスクライブするリポジトリ webhook が GitHub アプリにインストールされます。

コードの変更が検出された後は、後続のアクションにコードを渡すための次のオプションがあります。

- デフォルト: 他の既存の CodePipeline ソースアクションと同様に、`CodeStarSourceConnection` はコミットの浅いコピーを含む ZIP ファイルを出力できます。
- フルクローン: `CodeStarSourceConnection` は、後続のアクションのためにリポジトリへの URL リファレンスを出力するように設定することも可能です。

  現在、Git URL リファレンスは、リポジトリと関連する Git メタデータをクローンするためにダウンストリーム CodeBuild アクションでのみ使用できます。Git URL リファレンスを CodeBuild 以外のアクションに渡そうとすると、エラーが発生します。

CodePipeline では、接続の作成時に AWS Connector インストールアプリをサードパーティーアカウントに追加するように求められます。`CodeStarSourceConnection` アクションを介して接続する前に、サードパーティープロバイダのアカウントとリポジトリを作成しておく必要があります。

> **Note**
>
> AWS CodeStar 接続を使用するために必要なアクセス許可を持つロールにポリシーを作成またはアタッチするには、Connections アクセス許可リファレンスを参照してください。CodePipeline サービスロールが作成された日時によっては、 AWS CodeStar 接続をサポートするためにアクセス許可を更新する必要がある場合があります。手順については、「CodePipeline サービスロールにアクセス許可を追加する」を参照してください。

> **Note**
>
> 欧州 (ミラノ) で接続を使用するには AWS リージョン、以下を行う必要があります。
>
> 1. リージョン固有のアプリをインストールする
> 2. リージョンを有効にする
>
> このリージョン固有のアプリで、欧州 (ミラノ) リージョンの接続をサポートします。サードパーティープロバイダーのサイトで公開されているアプリであり、他のリージョンの接続をサポートする既存のアプリとは別のものです。このアプリをインストールすることで、このリージョンでのみサービスとデータを共有することをサードパーティープロバイダーに許可します。アプリをアンインストールすることでいつでもアクセス許可を取り消すことができます。
> リージョンを有効にしない限り、サービスはデータを処理または保存しません。このリージョンを有効にすることで、データを処理および保存するアクセス許可をサービスに付与したことになります。
> リージョンが有効になっていなくても、リージョン固有のアプリがインストールされたままであれば、サードパーティープロバイダーはお客様のデータをサービスと共有できます。したがって、リージョンを無効にしたら、必ずアプリをアンインストールしてください。詳細については、「リージョンの有効化」を参照してください。

トピック

- アクションタイプ
- 設定パラメータ
- 入力アーティファクト
- 出力アーティファクト
- 出力変数
- サービスロールのアクセス許可: AWS CodeConnections アクション
- アクションの宣言
- インストールアプリケーションのインストールと接続の作成
- 関連情報

アクションタイプ

- カテゴリ:`Source`
- 所有者: `AWS`
- プロバイダー: `CodeStarSourceConnection`
- バージョン: `1`

設定パラメータ

ConnectionArn

必須: はい

ソースプロバイダに対して設定および認証された接続 ARN。

FullRepositoryId

必須: はい

ソースの変更が検出される所有者とリポジトリの名前。

例: `some-user/my-repo`

> **Important**
>
> FullRepositoryId 値の大文字と小文字は正しく保持する必要があります。例えば、ユーザー名が `some-user` で、リポジトリ名が `My-Repo` の場合、FullRepositoryId の推奨値は `some-user/My-Repo` です。

BranchName

必須: はい

ソースの変更が検出されるブランチの名前。

OutputArtifactFormat

必須: いいえ

出力アーティファクト形式を指定します。`CODEBUILD_CLONE_REF` または `CODE_ZIP` のいずれかになります。指定しない場合、デフォルトの `CODE_ZIP` が使用されます。

> **Important**
>
> CODEBUILD_CLONE_REF のオプションは、CodeBuild のダウンストリームアクションでのみ使用可能です。
> このオプションを選択した場合は、Bitbucket、GitHub、GitHub Enterprise Server、または GitLab.com に接続するための CodeBuild GitClone アクセス許可を追加します。 で示されるように CodeBuild プロジェクトサービスロールの許可を更新する必要があります。[フルクローン] オプションを使用する方法を示すチュートリアルについては、チュートリアル: CodeCommit パイプラインソースで完全なクローンを使用する を参照してください。

DetectChanges

必須: いいえ

設定したリポジトリとブランチで新しいコミットが行われたときに、パイプラインを自動的にスタートするように制御します。未指定の場合、デフォルト値は true となり、フィールドはデフォルトで表示されません。このパラメータの有効な値:

- true: CodePipeline は、新しいコミットでパイプラインを自動的に開始します。
- false: CodePipeline は新しいコミットでパイプラインを開始しません。

入力アーティファクト

- アーティファクトの数: 0
- 説明: 入力アーティファクトは、このアクションタイプには適用されません。

出力アーティファクト

- アーティファクトの数: 1
- 説明: レポジトリから生成されたアーティファクトは、CodeStarSourceConnection アクションに対する出力アーティファクトです。ソースコードのコミット ID は、パイプライン実行のトリガーとなるソースリビジョンとして、CodePipeline に表示されます。このアクションの出力アーティファクトは、次で構成できます。
  - パイプライン実行のソースレビジョンとして指定されたコミットで設定されたレポジトリおよびブランチのコンテンツを含む ZIP ファイル。

- リポジトリへの URL リファレンスを含む JSON ファイル。これで下流のアクションが Git コマンドを直接実行できるようになります。

> **Important**
>
> このオプションは、CodeBuild ダウンストリームアクションでのみ使用できます。このオプションを選択した場合は、CodePipeline のトラブルシューティング で示されるように CodeBuild プロジェクトサービスロールの権限を更新する必要があります。[フルクローン] オプションを使用する方法を示すチュートリアルについては、チュートリアル: CodeCommit パイプラインソースで完全なクローンを使用する を参照してください。

出力変数

このアクションを設定すると、パイプライン内のダウンストリームアクションのアクション設定によって参照できる変数が生成されます。このアクションは、アクションに名前空間がない場合でも、出力変数として表示できる変数を生成します。名前空間を使用してアクションを設定し、これらの変数をダウンストリームアクションの設定で使用できるようにします。

詳細については、「変数リファレンス」を参照してください。

AuthorDate

コミットが認証された日付 (タイムスタンプ形式)。

BranchName

ソースが変更された リポジトリのブランチの名前。

CommitId

パイプライン実行をトリガーした コミット ID。

CommitMessage

パイプライン実行をトリガーしたコミットに関連付けられた説明メッセージ (存在する場合)。

ConnectionArn

ソースプロバイダに対して設定および認証された接続 ARN。

FullRepositoryName

パイプラインをトリガーしたコミットが行われた リポジトリの名前。

サービスロールのアクセス許可: AWS CodeConnections アクション

では AWS CodeConnections、Bitbucket Cloud などの接続を使用するソースでパイプラインを作成するには、次のアクセス許可が必要です。

```
{
    "Effect": "Allow",
    "Action": [
        "codeconnections:UseConnection",
        "codestar-connections:UseConnection"
    ],
    "Resource": "resource_ARN"
},
```

> **Note**
>
> 両方のアクションが必要です。呼び出し によって呼び出されるオペレーション とcodestar-connections:UseConnection、呼び出し の外部で呼び出されるオペレーションcodeconnections:UseConnection。両方のアクションを付与すると、接続のリソース ARN が codeconnectionsまたは codestar-connectionsサービスプレフィックスを使用しているかどうかに関係なく、ロールは両方のコードパスで動作します。

接続に関する IAM アクセス許可の詳細については、「Connections アクセス許可リファレンス」を参照してください。

アクションの宣言

次の例では、CODE_ZIP ARN との接続で出力アーティファクトが arn:aws:codestar-connections:region:*account-id*:connection/*connection-id* のデフォルト ZIP 形式に設定されています。

YAML

```
Name: Source
```

```
Actions:
  - InputArtifacts: []
    ActionTypeId:
      Version: '1'
      Owner: AWS
      Category: Source
      Provider: CodeStarSourceConnection
    OutputArtifacts:
      - Name: SourceArtifact
    RunOrder: 1
    Configuration:
      ConnectionArn: "arn:aws:codestar-connections:region:account-
id:connection/connection-id"
      FullRepositoryId: "some-user/my-repo"
      BranchName: "main"
      OutputArtifactFormat: "CODE_ZIP"
    Name: ApplicationSource
```

JSON

```
{
    "Name": "Source",
    "Actions": [
        {
            "InputArtifacts": [],
            "ActionTypeId": {
                "Version": "1",
                "Owner": "AWS",
                "Category": "Source",
                "Provider": "CodeStarSourceConnection"
            },
            "OutputArtifacts": [
                {
                    "Name": "SourceArtifact"
                }
            ],
            "RunOrder": 1,
            "Configuration": {
                "ConnectionArn": "arn:aws:codestar-connections:region:account-
id:connection/connection-id",
                "FullRepositoryId": "some-user/my-repo",
                "BranchName": "main",
                "OutputArtifactFormat": "CODE_ZIP"
```

```
            },
            "Name": "ApplicationSource"
        }
    ]
},
```

インストールアプリケーションのインストールと接続の作成

コンソールを使用してサードパーティーリポジトリに新しい接続を初めて追加するときは、リポジトリへの CodePipeline アクセスを認可する必要があります。サードパーティーコードリポジトリを作成したアカウントに接続するためのインストールアプリを選択または作成します。

AWS CLI または CloudFormation テンプレートを使用する場合は、インストールハンドシェイクをすでに通過している接続の接続 ARN を指定する必要があります。提供しないと、パイプラインはトリガーされません。

> **Note**
>
> `CodeStarSourceConnection` ソースアクションの場合、webhook を設定したり、ポーリングをデフォルトにする必要はありません。接続アクションは、ソースの変更検出を管理します。

関連情報

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- AWS::CodeStarConnections::Connection – AWS CodeStar Connections リソースの CloudFormation テンプレートリファレンスは、 CloudFormation テンプレート内の接続のパラメータと例を提供します。
- AWS CodeStar Connections API リファレンス – AWS CodeStar Connections API リファレンスは、使用可能な接続アクションのリファレンス情報を提供します。
- 接続でサポートされているソースアクションでパイプラインを作成するステップについては、以下を参照してください。
  - Bitbucket Cloud の場合は、コンソールの [Bitbucket] オプションまたは CLI の `CodestarSourceConnection` アクションを使用します。「Bitbucket Cloud への接続」を参照してください。

- GitHub および GitHub Enterprise Cloud の場合、コンソールの [GitHub] プロバイダオプションまたは CodestarSourceConnection CLI のアクションを使用します。「GitHub コネクション」を参照してください。
- GitHub Enterprise Server の場合は、コンソールの [GitHub Enterprise Server] プロバイダオプション、または CodestarSourceConnection CLI のアクションを使用します。「GitHub Enterprise Server 接続」を参照してください。
- GitLab.com の場合は、コンソールの GitLab プロバイダーオプション、または CLI の CodestarSourceConnection アクションとGitLab プロバイダーを使用します。「GitLab.com への接続」を参照してください。

- Bitbucket ソースと CodeBuild アクションを使用してパイプラインを作成する スタートアップチュートリアルを表示するには、[接続をはじめよう] を参照してください。
- GitHub リポジトリに接続し、ダウンストリーム CodeBuild アクションで [フルクローン] オプションを使用する方法を紹介したチュートリアルは、チュートリアル: CodeCommit パイプラインソースで完全なクローンを使用する を参照してください。

コマンドアクションリファレンス

コマンドアクションを使用すると、仮想コンピューティングインスタンスでシェルコマンドを実行できます。アクションを実行すると、アクション設定で指定したコマンドが別のコンテナで実行されます。CodeBuild アクションへの入力アーティファクトとして指定されたすべてのアーティファクトは、コマンドを実行するコンテナ内で使用できます。このアクションでは、CodeBuild プロジェクトを最初に作成せずにコマンドを指定できます。詳細については、「AWS CodePipeline API リファレンス」の「ActionDeclaration」と「OutputArtifact」を参照してください。

> **Important**
>
> このアクションでは、CodePipeline マネージド CodeBuild コンピューティングを使用して、ビルド環境でコマンドを実行します。コマンドアクションを実行すると、 AWS CodeBuildで別途料金が発生します。

> **Note**
>
> コマンドアクションは V2 タイプのパイプラインでのみ使用できます。

トピック

- コマンドアクションに関する考慮事項
- サービスロールのポリシーのアクセス許可
- アクションタイプ
- 設定パラメータ
- 入力アーティファクト
- 出力アーティファクト
- 環境変数
- サービスロールのアクセス許可: コマンドアクション
- アクションの宣言 (例)
- 関連情報

コマンドアクションに関する考慮事項

コマンドアクションには以下の考慮事項が適用されます。

- コマンドアクションは CodeBuild アクションと同様の CodeBuild リソースを使用しますが、ビルドプロジェクトを関連付けたり作成したりする必要はなく、仮想コンピューティングインスタンス内でシェル環境コマンドが許可されます。

> **Note**
>
> コマンドアクションを実行すると、 AWS CodeBuildで別途料金が発生します。

- CodePipeline のコマンドアクションは CodeBuild リソースを使用するため、アクションで実行するビルドには、アカウントに対する CodeBuild のビルド制限が適用されます。コマンドアクションで実行したビルドは、アカウントに設定されている同時ビルド制限にカウントされます。
- コマンドアクションを使用したビルドのタイムアウトは、CodeBuild ビルドに基づいて 55 分です。
- コンピューティングインスタンスは、CodeBuild の分離されたビルド環境を使用します。

> **Note**
>
> 分離されたビルド環境はアカウントレベルで使用されるため、インスタンスは別のパイプライン実行に再利用される場合があります。

- 複数行形式を除くすべての形式がサポートされています。コマンドを入力するときは、単一行形式を使用する必要があります。
- コマンドアクションは、クロスアカウントアクションでサポートされています。クロスアカウントコマンドアクションを追加するには、アクション宣言でターゲットアカウントから `actionRoleArn` を追加します。
- このアクションでは、CodePipeline がパイプラインサービスロールを引き受け、このロールを使用してランタイムにリソースへのアクセスを許可します。アクセス許可の範囲をアクションレベルまで絞り込むように、サービスロールを設定することをお勧めします。
- CodePipeline サービスロールに追加されるアクセス許可の詳細については、「CodePipeline サービスロールにアクセス許可を追加する」を参照してください。
- コンソールでログを表示するために必要なアクセス許可の詳細については、「コンソールでコンピューティングログを表示するために必要なアクセス許可」を参照してください。
- CodePipeline の他のアクションとは異なり、アクション設定ではフィールドを設定せず、アクション設定の外部でアクション設定フィールドを設定します。

サービスロールのポリシーのアクセス許可

CodePipeline は、アクションを実行するときに、次のようにパイプライン名を使用してロググループを作成します。これにより、パイプライン名を使用してアクセス許可の範囲をリソースのログ記録に絞り込むことができます。

```
/aws/codepipeline/MyPipelineName
```

既存のサービスロールを使用している場合、コマンドアクションを使用するには、サービスロールに以下のアクセス許可を追加する必要があります。

- logs:CreateLogGroup
- logs:CreateLogStream
- logs:PutLogEvents

サービスロールポリシーステートメントで、次の例に示すように、アクセス許可の範囲をパイプラインレベルに絞り込みます。

```
{
    "Effect": "Allow",
    "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
    ],
    "Resource": [
        "arn:aws:logs:*:YOUR_AWS_ACCOUNT_ID:log-group:/aws/
codepipeline/YOUR_PIPELINE_NAME",
        "arn:aws:logs:*:YOUR_AWS_ACCOUNT_ID:log-group:/aws/
codepipeline/YOUR_PIPELINE_NAME:*"
   ]
}
```

アクションの詳細ダイアログページを使用してコンソールでログを表示するには、ログを表示するアクセス許可をコンソールロールに追加する必要があります。詳細については、「コンソールでコンピューティングログを表示するために必要なアクセス許可」でコンソールのアクセス許可ポリシーの例を参照してください。

アクションタイプ

- カテゴリ:Compute
- 所有者: AWS
- プロバイダー: Commands
- バージョン: 1

設定パラメータ

コマンド

必須: はい

Commands アクションで実行するシェルコマンドを指定できます。コンソールでは、各コマンドを個別の行に入力します。CLI では、コマンドを個別の文字列として入力します。

> **Note**
>
> 複数行形式はサポートされていないため、エラーメッセージが表示されます。[コマンド] フィールドにコマンドを入力するには、単一行形式を使用する必要があります。

> **Important**
>
> EnvironmentType と ComputeType の値は CodeBuild の値と一致します。使用可能なタイプのサブセットがサポートされています。詳細については、「ビルド環境コンピューティングタイプ」を参照してください。

EnvironmentType

必須: いいえ

コマンドアクションをサポートするビルド環境の OS イメージ。ビルド環境の有効な値は次のとおりです。

- LINUX_CONTAINER
- WINDOWS_SERVER_2022_CONTAINER

[EnvironmentType] を選択すると、[ComputeType] フィールドでその OS のコンピューティングタイプが許可されます。このアクションで利用可能な CodeBuild コンピューティングタイプの詳細については、「CodeBuild ユーザーガイド」の「ビルド環境のコンピューティングモードおよびタイプ」リファレンスを参照してください。

> **Note**
>
> 指定しない場合、コンピューティングはビルド環境に対してデフォルトで次のようになります。
>
> - コンピューティングタイプ: BUILD_GENERAL1_SMALL
> - 環境タイプ: LINUX_CONTAINER

ComputeType

必須: いいえ

EnvironmentType の選択に基づいて、コンピューティングタイプを指定できます。以下はコンピューティングで使用可能な値ですが、使用可能なオプションは OS によって異なる場合があります。

- BUILD_GENERAL1_SMALL
- BUILD_GENERAL1_MEDIUM
- BUILD_GENERAL1_LARGE

> **Important**
>
> 一部のコンピューティングタイプは、特定の環境タイプと互換性がありません。例えば、WINDOWS_SERVER_2022_CONTAINER は BUILD_GENERAL1_SMALL と互換性がありません。互換性のない組み合わせを使用すると、アクションは失敗し、ランタイムエラーが生成されます。

outputVariables

必須: いいえ

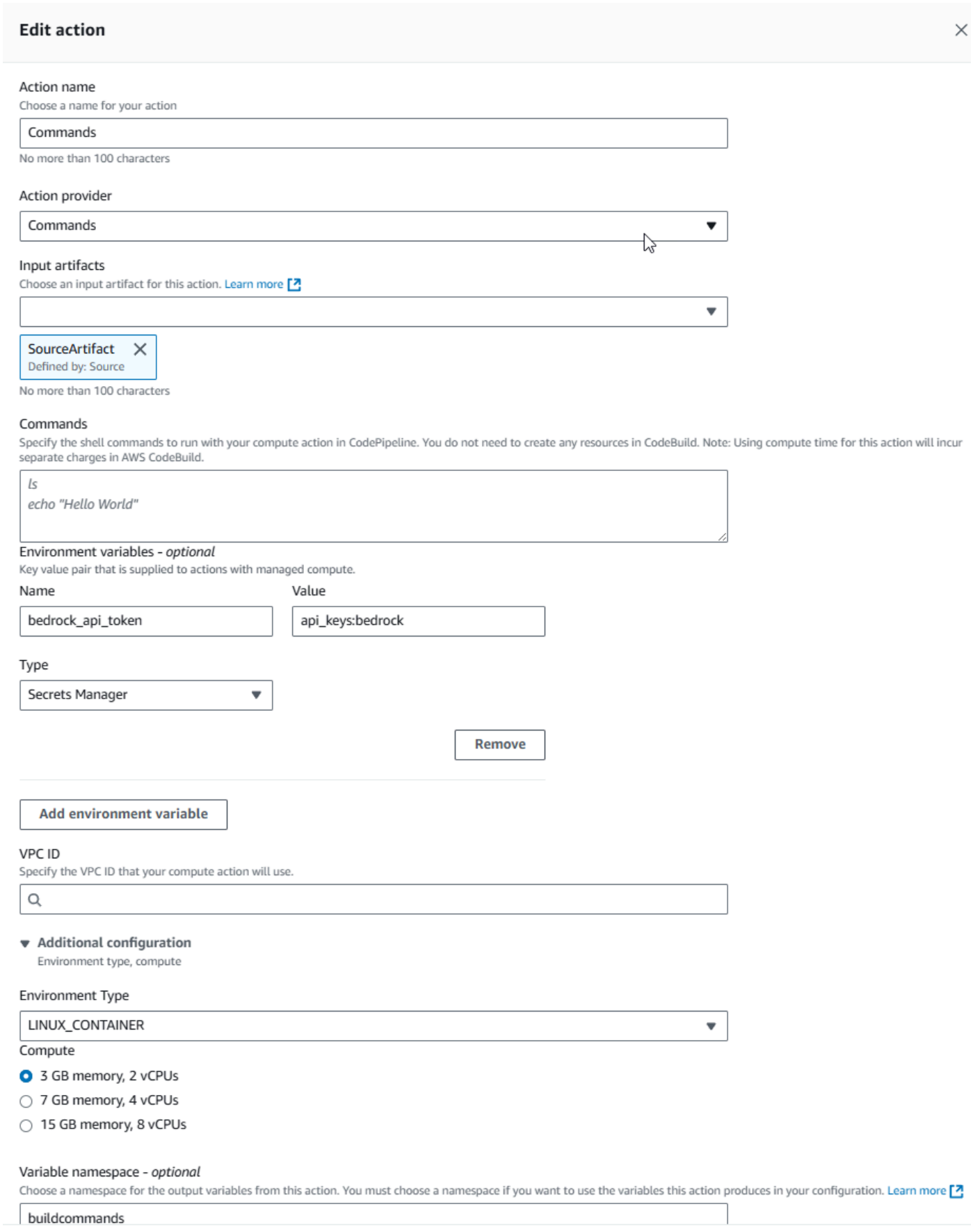
エクスポートする環境内の変数の名前を指定します。CodeBuild 環境変数のリファレンスについては、「CodeBuild ユーザーガイド」の「ビルド環境の環境変数」を参照してください。

ファイル

必須: いいえ

エクスポートするファイルは、アクションの出力アーティファクトとして指定できます。

サポートされているファイル形式は、CodeBuild ファイルパターンと同じです。例えば、すべてのファイルの場合は「**/」と入力します。詳細については、「CodeBuild ユーザーガイド」の「CodeBuild のビルド仕様リファレンス」を参照してください。


Edit action
Action name
Choose a name for your action
Commands
No more than 100 characters
Action provider
Commands
Input artifacts
Choose an input artifact for this action. Learn more
SourceArtifact
Defined by: Source
No more than 100 characters
Commands
Specify the shell commands to run with your compute action in CodePipeline. You do not need to create any resources in CodeBuild. Note: Using compute time for this action will incur separate charges in AWS CodeBuild.
ls
echo "Hello World"
Environment variables - optional
Key value pair that is supplied to actions with managed compute.
Name
Value
bedrock_api_token
api_keys:bedrock
Type
Secrets Manager
Remove
Add environment variable
VPC ID
Specify the VPC ID that your compute action will use.
Additional configuration
Environment type, compute
Environment Type
LINUX_CONTAINER
Compute
3 GB memory, 2 vCPUs
7 GB memory, 4 vCPUs
15 GB memory, 8 vCPUs
Variable namespace - optional
Choose a namespace for the output variables from this action. You must choose a namespace if you want to use the variables this action produces in your configuration. Learn more
buildcommands
Cancel
Done

VpcId

必須: いいえ

リソースの VPC ID。

サブネット

必須: いいえ

VPC のサブネット。このフィールドは、コマンドが VPC 内のリソースに接続する必要がある場合に必要です。

SecurityGroupIds

必須: いいえ

VPC のセキュリティグループ。このフィールドは、コマンドが VPC 内のリソースに接続する必要がある場合に必要です。

以下は、環境とコンピューティングタイプの設定フィールドと環境変数の例を示す アクションの JSON 例です。

```
{
           "name": "Commands1",
           "actionTypeId": {
             "category": "Compute",
             "owner": "AWS",
             "provider": "Commands",
             "version": "1"
           },
           "inputArtifacts": [
             {
               "name": "SourceArtifact"
             }
           ],
           "commands": [
             "ls",
             "echo hello",
             "echo $BEDROCK_TOKEN",
           ],
           "configuration": {
             "EnvironmentType": "LINUX_CONTAINER",
```

```
            "ComputeType": "BUILD_GENERAL1_MEDIUM"
          },
          "environmentVariables": [
            {
              "name": "BEDROCK_TOKEN",
              "value": "apiTokens:bedrockToken",
              "type": "SECRETS_MANAGER"
            }
          ],
          "runOrder": 1
        }
```

入力アーティファクト

- アーティファクトの数: `1 to 10`

出力アーティファクト

- アーティファクトの数: `0 to 1`

環境変数

キー

キーと値の環境変数ペアのキー (BEDROCK_TOKEN など)。

値

キーと値のペアの値 (apiTokens:bedrockToken など)。値は、パイプラインアクションまたはパイプライン変数からの出力変数でパラメータ化できます。

SECRETS_MANAGER タイプを使用する場合、この値は Secrets Manager に既に保存しているシー AWS クレットの名前である必要があります。

タイプ

環境変数値の使用タイプを指定します。この値は PLAINTEXT または SECRETS_MANAGER となります。値が SECRETS_MANAGER の場合、EnvironmentVariable 値に Secrets リファレンスを指定します。指定しない場合、この値はデフォルトで PLAINTEXT になります。

> **Note**
>
> 機密情報、特に AWS 認証情報を保存するためにプレーンテキストの環境変数を使用することは強くお勧めしません。CodeBuild コンソールまたは AWS CLIを使用すると、plaintext 環境変数はプレーンテキストで表示されます。機密の値の場合は、代わりに SECRETS_MANAGER タイプを使用することをお勧めします。

> **Note**
>
> 環境変数の設定に name、value、および type を入力する場合 (特に環境変数に CodePipeline の出力変数の構文が含まれている場合) は、設定の値フィールドの 1000 文字制限を超えないようにしてください。この制限を超えると、検証エラーが返されます。

環境変数を示すアクション宣言の例については、「設定パラメータ」を参照してください。

> **Note**
>
> - SECRETS_MANAGER タイプは、コマンドアクションでのみサポートされます。
> - コマンドアクションで参照されるシークレットは、CodeBuild のようなビルドログで編集されます。ただし、パイプラインへの編集アクセスを持つパイプラインユーザーは、コマンドを変更することで、これらのシークレット値に引き続きアクセスできる可能性があります。
> - SecretsManager を使用するには、パイプラインサービスロールに次のアクセス許可を追加する必要があります。
>
> ```
> {
>             "Effect": "Allow",
>             "Action": [
>                 "secretsmanager:GetSecretValue"
>             ],
>             "Resource": [
>                 "SECRET_ARN"
>             ]
>         }
> ```

サービスロールのアクセス許可: コマンドアクション

コマンドがサポートされるように、以下をポリシーステートメントに追加します。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:iam::*:role/Service*",
                "arn:aws:iam::*:role/Service*"
            ]
        }
    ]
}
```

アクションの宣言 (例)

YAML

```
name: Commands_action
actionTypeId:
  category: Compute
  owner: AWS
  provider: Commands
  version: '1'
runOrder: 1
configuration: {}
commands:
- ls
- echo hello
- 'echo pipeline Execution Id is #{codepipeline.PipelineExecutionId}'
outputArtifacts:
```

```
- name: BuildArtifact
  files:
  - **/
inputArtifacts:
- name: SourceArtifact
outputVariables:
- AWS_DEFAULT_REGION
region: us-east-1
namespace: compute
```

JSON

```
{
    "name": "Commands_action",
    "actionTypeId": {
        "category": "Compute",
        "owner": "AWS",
        "provider": "Commands",
        "version": "1"
    },
    "runOrder": 1,
    "configuration": {},
    "commands": [
        "ls",
        "echo hello",
        "echo pipeline Execution Id is #{codepipeline.PipelineExecutionId}"
    ],
    "outputArtifacts": [
        {
            "name": "BuildArtifact",
            "files": [
                "**/"
            ]
        }
    ],
    "inputArtifacts": [
        {
            "name": "SourceArtifact"
        }
    ],
    "outputVariables": [
        "AWS_DEFAULT_REGION"
    ],
```

```
    "region": "us-east-1",
    "namespace": "compute"
}
```

関連情報

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- チュートリアル: コンピューティングを使用してコマンドを実行するパイプラインを作成する (V2 タイプ) – このチュートリアルでは、コマンドアクションを使用したサンプルパイプラインを提供します。

AWS Device Farm テストアクションリファレンス

パイプラインでは、 がデバイス上でアプリケーションを実行およびテスト AWS Device Farm するために使用するテストアクションを設定できます。Device Farm は、デバイスのテストプールとテストフレームワークを使用して、特定のデバイス上でアプリケーションをテストします。Device Farm アクションでサポートされているテストフレームワークのタイプについては、AWS Device Farm でのテストタイプの操作」を参照してください。

トピック

- アクションタイプ
- 設定パラメータ
- 入力アーティファクト
- 出力アーティファクト
- サービスロールのアクセス許可: AWS Device Farm アクション
- アクションの宣言
- 関連情報

アクションタイプ

- カテゴリ:`Test`
- 所有者: `AWS`
- プロバイダー: `DeviceFarm`

- バージョン: 1

設定パラメータ

AppType

必須: はい

テストする OS とアプリケーションのタイプ。有効な値のリストを次に示します。

- `iOS`
- `Android`
- `Web`

ProjectId

必須: はい

Device Farm プロジェクト ID。

プロジェクト ID を見つけるには、Device Farm コンソールでプロジェクトを選択します。ブラウザで、新しいプロジェクトの URL をコピーします。URL には、プロジェクト ID が含まれます。プロジェクト ID は、`projects/` 以降の URL の値です。次の例で、プロジェクト ID は `eec4905f-98f8-40aa-9afc-4c1cfexample` です。

```
https://<region-URL>/devicefarm/home?region=us-west-2#/projects/
eec4905f-98f8-40aa-9afc-4c1cfexample/runs
```

アプリケーション

必須: はい

入力アーティファクト内のアプリケーションファイルの名前と場所。例: `s3-ios-test-1.ipa`

TestSpec

条件付き: はい

入力アーティファクト内のテストスペック定義ファイルの場所。これはカスタムモードのテストに必要です。

DevicePoolArn

必須: はい

Device Farm デバイスプールの ARN。

上位デバイスの ARNs など、プロジェクトの使用可能なデバイスプール ARN を取得するには、CLI AWS を使用して次のコマンドを入力します。

```
aws devicefarm list-device-pools --arn arn:aws:devicefarm:us-
west-2:account_ID:project:project_ID
```

TestType

必須: はい

テストでサポートされるテストフレームワークを指定します。TestType の有効な値のリストを次に示します。

- APPIUM_JAVA_JUNIT
- APPIUM_JAVA_TESTNG
- APPIUM_NODE
- APPIUM_RUBY
- APPIUM_PYTHON
- APPIUM_WEB_JAVA_JUNIT
- APPIUM_WEB_JAVA_TESTNG
- APPIUM_WEB_NODE
- APPIUM_WEB_RUBY
- APPIUM_WEB_PYTHON
- BUILTIN_FUZZZ
- INSTRUMENTATION
- XCTEST
- XCTEST_UI

> **Note**
>
> 以下のテストタイプは、CodePipeline のアクション WEB_PERFORMANCE_PROFILE、REMOTE_ACCESS_RECORD、および REMOTE_ACCESS_REPLAY ではサポートされていません。

Device Farm のテストタイプついては、[AWS Device Farm でのテストタイプの操作] を参照してください。

RadioBluetoothEnabled

必須: いいえ

テストの開始時に Bluetooth を有効にするかどうかを示すブール値。

RecordAppPerformanceData

必須: いいえ

テスト中に CPU、FPS、メモリパフォーマンスなどのデバイスパフォーマンスデータを記録するかどうかを示すブール値。

RecordVideo

必須: いいえ

テスト中にビデオを記録するかどうかを示すブール値。

RadioWifiEnabled

必須: いいえ

テストの開始時に Wi-Fi を有効にするかどうかを示すブール値。

RadionFCenabled

必須: いいえ

テストの開始時に NFC を有効にするかどうかを示すブール値。

RadioGpsEnabled

必須: いいえ

テストの開始時に GPS を有効にするかどうかを示すブール値。

テスト

必須: いいえ

ソースの場所にあるテスト定義ファイルの名前とパス。パスは、テストの入力アーティファクトのルートに関連します。

FuzzEventCount

必須: いいえ

ファズテストが実行するユーザーインターフェイスイベントの数で、1 から 10000 の間で指定します。

FuzzEventThrottle

必須: いいえ

ファズテストが次のユーザーインターフェイスイベントを実行する前に待機するミリ秒数で、1 から 1000 の間で指定します。

FuzzRandomizerSeed

必須: いいえ

ユーザインタフェースイベントをランダム化するために使用するファズテストのシード。後続のファズテストに同じ番号を使用すると、同じイベントシーケンスになります。

CustomHostMachineArtifacts

必須: いいえ

ホストマシン上でカスタムアーティファクトが格納される場所。

CustomDeviceArtifacts

必須: いいえ

カスタムアーティファクトが保存されるデバイス上の場所。

UnmeteredDevicesOnly

必須: いいえ

この手順でテストを実行するときに、測定されていないデバイスのみを使用するかどうかを示すブール値。

JobTimeoutMinutes

必須: いいえ

テスト実行がタイムアウトになるまでにデバイスごとに実行される分数。

Latitude (緯度)

必須: いいえ

デバイスの緯度は、地理座標系の度数で表されます。

Longitude (経度)

必須: いいえ

地理座標系の度数で表されたデバイスの経度。

入力アーティファクト

- アーティファクトの数: 1
- 説明: テストアクションで使用可能にするアーティファクトのセット。Device Farm は、ビルドされたアプリケーションとテスト定義を使用するために検索します。

出力アーティファクト

- アーティファクトの数: 0
- 説明: 出力アーティファクトは、このアクションタイプには適用されません。

サービスロールのアクセス許可: AWS Device Farm アクション

CodePipeline がアクションを実行する際、CodePipeline サービスロールポリシーには、最小特権でアクセスを維持するために、パイプラインリソース ARN に適切にスコープダウンされた次のアクセス許可が必要です。例えば、以下をポリシーステートメントに追加します。

```
{
    "Effect": "Allow",
    "Action": [
        "devicefarm:ListProjects",
        "devicefarm:ListDevicePools",
        "devicefarm:GetRun",
        "devicefarm:GetUpload",
        "devicefarm:CreateUpload",
```

```
        "devicefarm:ScheduleRun"
    ],
    "Resource": "resource_ARN"
},
```

アクションの宣言

YAML

```
Name: Test
Actions:
  - Name: TestDeviceFarm
    ActionTypeId: null
    category: Test
    owner: AWS
    provider: DeviceFarm
    version: '1'
RunOrder: 1
Configuration:
  App: s3-ios-test-1.ipa
  AppType: iOS
  DevicePoolArn: >-
    arn:aws:devicefarm:us-west-2::devicepool:0EXAMPLE-d7d7-48a5-ba5c-b33d66efa1f5
  ProjectId: eec4905f-98f8-40aa-9afc-4c1cfEXAMPLE
  TestType: APPIUM_PYTHON
  TestSpec: example-spec.yml
OutputArtifacts: []
InputArtifacts:
  - Name: SourceArtifact
Region: us-west-2
```

JSON

```
{
    "Name": "Test",
    "Actions": [
        {
            "Name": "TestDeviceFarm",
            "ActionTypeId": null,
            "category": "Test",
            "owner": "AWS",
            "provider": "DeviceFarm",
```

```
            "version": "1"
        }
    ],
    "RunOrder": 1,
    "Configuration": {
        "App": "s3-ios-test-1.ipa",
        "AppType": "iOS",
        "DevicePoolArn": "arn:aws:devicefarm:us-west-2::devicepool:0EXAMPLE-
d7d7-48a5-ba5c-b33d66efa1f5",
        "ProjectId": "eec4905f-98f8-40aa-9afc-4c1cfEXAMPLE",
        "TestType": "APPIUM_PYTHON",
        "TestSpec": "example-spec.yml"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "SourceArtifact"
        }
    ],
    "Region": "us-west-2"
},
```

関連情報

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- [Device Farm でのテストタイプの実行] - [Device Farm 開発者ガイド] のこのリファレンス章では、Device Farm でサポートされる Android、iOS、および Web アプリケーションのテストフレームワークについて詳しく説明します。
- [Device Farm のアクション] - [Device Farm API リファレンス] の API 呼び出しとパラメータは、Device Farm プロジェクトの操作に役立ちます。
- チュートリアル: を使用して Android アプリを構築およびテストするパイプラインを作成する AWS Device Farm – このチュートリアルでは、CodeBuild と Device Farm を使用して Android アプリを構築およびテストする GitHub ソースを使用してパイプラインを作成するための buildspec ファイルとサンプルアプリケーションのサンプルを提供します。
- チュートリアル: で iOS アプリをテストするパイプラインを作成する AWS Device Farm — このチュートリアルでは、Device Farm でビルドされた iOS アプリケーションをテストする Amazon S3 ソースでパイプラインを作成するためのサンプルアプリケーションを提供します。

Elastic Beanstalk のデプロイアクションリファレンス

Elastic Beanstalk は、ウェブアプリケーションのデプロイとスケーリングに使用される AWS 内のプラットフォームです。Elastic Beanstalk のアクションを使用して、デプロイ環境にアプリケーションコードをデプロイします。

トピック

- アクションタイプ
- 設定パラメータ
- 入力アーティファクト
- 出力アーティファクト
- サービスロールのアクセス許可: ElasticBeanstalk のデプロイアクション
- アクションの宣言
- 関連情報

アクションタイプ

- カテゴリ:`Deploy`
- 所有者: `AWS`
- プロバイダー: `ElasticBeanstalk`
- バージョン: `1`

設定パラメータ

ApplicationName

必須: はい

Elastic Beanstalk に作成したアプリケーションの名前。

EnvironmentName

必須: はい

Elastic Beanstalk に作成した環境の名前。環境は、アプリケーションバージョンを実行する AWS リソースのコレクションです。各環境は一度に 1 つのアプリケーションバージョンのみを実行し

ますが、多くの環境で同じアプリケーションバージョンまたは異なるアプリケーションバージョンを同時に実行できます。

入力アーティファクト

- アーティファクトの数: 1
- 説明: アクションの入力アーティファクト。

出力アーティファクト

- アーティファクトの数: 0
- 説明: 出力アーティファクトは、このアクションタイプには適用されません。

サービスロールのアクセス許可: ElasticBeanstalk のデプロイアクション

Elastic Beanstalk では、ElasticBeanstalk デプロイアクションを使用してパイプラインを作成するために必要な最小限のアクセス許可は次のとおりです。

```
{
    "Effect": "Allow",
    "Action": [
        "elasticbeanstalk:*",
        "ec2:*",
        "elasticloadbalancing:*",
        "autoscaling:*",
        "cloudwatch:*",
        "s3:*",
        "sns:*",
        "cloudformation:*",
        "rds:*",
        "sqs:*",
        "ecs:*"
    ],
    "Resource": "resource_ARN"
},
```

> **Note**
>
> リソースポリシーのワイルドカードは、アクセスを制限するアカウントのリソースに置き換える必要があります。最小権限アクセスを付与するポリシーの作成の詳細については、https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#grant-least-privilege を参照してください。

アクションの宣言

YAML

```
Name: Deploy
Actions:
  - Name: Deploy
    ActionTypeId:
      Category: Deploy
      Owner: AWS
      Provider: ElasticBeanstalk
      Version: '1'
    RunOrder: 1
    Configuration:
      ApplicationName: my-application
      EnvironmentName: my-environment
    OutputArtifacts: []
    InputArtifacts:
      - Name: SourceArtifact
    Region: us-west-2
    Namespace: DeployVariables
```

JSON

```
{
    "Name": "Deploy",
    "Actions": [
        {
            "Name": "Deploy",
            "ActionTypeId": {
                "Category": "Deploy",
                "Owner": "AWS",
                "Provider": "ElasticBeanstalk",
```

```
                "Version": "1"
            },
            "RunOrder": 1,
            "Configuration": {
                "ApplicationName": "my-application",
                "EnvironmentName": "my-environment"
            },
            "OutputArtifacts": [],
            "InputArtifacts": [
                {
                    "Name": "SourceArtifact"
                }
            ],
            "Region": "us-west-2",
            "Namespace": "DeployVariables"
        }
    ]
},
```

関連情報

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- Flask アプリケーションを Elastic Beanstalk にデプロイする – このチュートリアルでは、サンプル Flask アプリケーションを使用して Elastic Beanstalk でアプリケーションと環境リソースを作成する方法について説明します。その後、ソースリポジトリから Elastic Beanstalk 環境にアプリケーションをデプロイする Elastic Beanstalk デプロイアクションを使用してパイプラインを構築できます。

Amazon Inspector **InspectorScan** 呼び出しアクションリファレンス

Amazon Inspector は、ソフトウェアの脆弱性や意図しないネットワークの露出についてワークロードを自動的に検出し、継続的にスキャンする脆弱性管理サービスです。CodePipeline の `InspectorScan` アクションは、オープンソースコードのセキュリティの脆弱性の検出と修正を自動化します。このアクションは、セキュリティスキャン機能を備えたマネージド型のコンピューティングアクションです。InspectorScan は、GitHub や Bitbucket Cloud などのサードパーティーのリポ

ジトリ内のアプリケーションソースコード、またはコンテナアプリケーションのイメージで使用できます。アクションは、設定した脆弱性レベルとアラートをスキャンしてレポートします。

> **Important**
>
> このアクションでは、CodePipeline マネージド CodeBuild コンピューティングを使用して、ビルド環境でコマンドを実行します。このアクションを実行すると、 AWS CodeBuildで別途料金が発生します。

トピック

- アクションタイプ ID
- 設定パラメータ
- 入力アーティファクト
- 出力アーティファクト
- 出力変数
- サービスロールのアクセス許可: InspectorScan アクション
- アクションの宣言
- 関連情報

アクションタイプ ID

- カテゴリ: `Invoke`
- 所有者: `AWS`
- プロバイダー: `InspectorScan`
- バージョン: `1`

例:

```
        {
            "Category": "Invoke",
            "Owner": "AWS",
            "Provider": "InspectorScan",
            "Version": "1"
```

```
            },
```

設定パラメータ

InspectorRunMode

(必須) スキャンモードを示す文字列。有効な値は `SourceCodeScan | ECRImageScan` です。

ECRRepositoryName

イメージがプッシュされた Amazon ECR リポジトリの名前。

ImageTag

イメージに使用するタグ。

このアクションのパラメータは、指定した脆弱性のレベルをスキャンします。脆弱性のしきい値には、次のレベルがあります。

CriticalThreshold

CodePipeline がアクションを失敗させる必要のある、ソースで検出された重大な脆弱性の数。

HighThreshold

CodePipeline がアクションを失敗させる必要のある、ソースで検出された重要度が高い脆弱性の数。

MediumThreshold

CodePipeline がアクションを失敗させる必要のある、ソースで検出された重要度が中程度の脆弱性の数。

LowThreshold

CodePipeline がアクションを失敗させる必要のある、ソースで検出された重要度が低い脆弱性の数。

Action name
Choose a name for your action
Scan
No more than 100 characters

Action provider
AWS InspectorScan

Region
US East (N. Virginia)

Input artifacts
Choose an input artifact for this action. Learn more

SourceArtifact
Defined by: Source
No more than 100 characters

Inspector Run mode
InspectorRunMode: SourceCodeScan or ECRImageScan.

Source Code Scan
SourceCodeScan: Scan the source code. Choose an input artifact for this run mode.

ECR Image Scan
ECRImageScan: Scan the ECR image.

Critical Threshold - *optional*
Threshold value for critical severity vulnerabilities to allow the action to succeed. Default: Integer.MAX_VALUE

High Threshold - *optional*
Threshold value for high severity vulnerabilities to allow the action to succeed. Default: Integer.MAX_VALUE

Medium Threshold - *optional*
Threshold value for medium severity vulnerabilities to allow the action to succeed. Default: Integer.MAX_VALUE

Low Threshold - *optional*
Threshold value for low severity vulnerabilities to allow the action to succeed. Default: Integer.MAX_VALUE

入力アーティファクト

- アーティファクトの数: 1
- 説明: 脆弱性をスキャンするソースコード。スキャンが ECR リポジトリ向けの場合、この入力アーティファクトは必要ありません。

出力アーティファクト

- アーティファクトの数: 1
- 説明: ソフトウェア部品表 (SBOM) ファイル形式のソースの脆弱性の詳細。

出力変数

このアクションを設定すると、パイプライン内のダウンストリームアクションのアクション設定によって参照できる変数が生成されます。このアクションは、アクションに名前空間がない場合でも、出力変数として表示できる変数を生成します。名前空間を使用してアクションを設定し、これらの変数をダウンストリームアクションの設定で使用できるようにします。

詳細については、「変数リファレンス」を参照してください。

HighestScannedSeverity

スキャンからの最も高い重要度の出力。有効な値は `medium | high | critical` です。

サービスロールのアクセス許可: **InspectorScan** アクション

`InspectorScan` アクションをサポートするには、以下をポリシーステートメントに追加します。

```
{
        "Effect": "Allow",
        "Action": "inspector-scan:ScanSbom",
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "ecr:GetDownloadUrlForLayer",
            "ecr:BatchGetImage",
```

```
            "ecr:BatchCheckLayerAvailability"
        ],
        "Resource": "resource_ARN"
    },
```

さらに、Commands アクションにまだ追加されていない場合は、CloudWatch ログを表示するために、サービスロールに次のアクセス許可を追加します。

```
{
    "Effect": "Allow",
    "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
    ],
    "Resource": "resource_ARN"
},
```

> **Note**
>
> サービスロールポリシーステートメントでリソースベースのアクセス許可を使用して、アクセス許可の範囲をパイプラインリソースレベルに絞り込みます。

アクションの宣言

YAML

```
name: Scan
actionTypeId:
  category: Invoke
  owner: AWS
  provider: InspectorScan
  version: '1'
runOrder: 1
configuration:
  InspectorRunMode: SourceCodeScan
outputArtifacts:
- name: output
inputArtifacts:
- name: SourceArtifact
```

```
region: us-east-1
```

JSON

```
{
                        "name": "Scan",
                        "actionTypeId": {
                            "category": "Invoke",
                            "owner": "AWS",
                            "provider": "InspectorScan",
                            "version": "1"
                        },
                        "runOrder": 1,
                        "configuration": {
                            "InspectorRunMode": "SourceCodeScan"
                        },
                        "outputArtifacts": [
                            {
                                "name": "output"
                            }
                        ],
                        "inputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "region": "us-east-1"
                    },
```

関連情報

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- Amazon Inspector の詳細については、「Amazon Inspector ユーザーガイド」を参照してください。

AWS Lambda アクションリファレンスを呼び出す

パイプラインのアクションとして Lambda 関数を実行できます。この関数への入力であるイベントオブジェクトを使用して、関数はアクション設定、入力アーティファクトの場所、出力アー

ティファクトの場所、およびアーティファクトへのアクセスに必要なその他の情報にアクセスできます。Lambda 呼び出し関数に渡されるイベントの例については、JSON イベントの例 を参照してください。Lambda 関数の実装の一部として、PutJobSuccessResult API または PutJobFailureResult API への呼び出しが必要です。それ以外の場合、このアクションの実行は、アクションがタイムアウトするまでハングします。アクションの出力アーティファクトを指定する場合、関数の実装の一部として S3 バケットにアップロードする必要があります。

> **Important**
>
> CodePipeline が Lambda に送信する JSON イベントをログに記録しないでください。これにより、CloudWatch Logs にユーザー認証情報が記録される可能性があるためです。CodePipeline ロールは JSON イベントを使用して、一時的な認証情報を `artifactCredentials` フィールドの Lambda に渡します。イベント例については、「JSON イベントの例」を参照してください。

アクションタイプ

- カテゴリ:`Invoke`
- 所有者: `AWS`
- プロバイダー: `Lambda`
- バージョン: `1`

設定パラメータ

FunctionName

必須: はい

`FunctionName` は、Lambda で作成された関数の名前です。

UserParameters

必須: いいえ

Lambda 関数による入力として処理できる文字列。

入力アーティファクト

- アーティファクトの数: `0 to 5`
- 説明: Lambda 関数で使用できるようにするアーティファクトのセット。

出力アーティファクト

- アーティファクトの数: `0 to 5`
- 説明: Lambda 関数によって出力として生成されるアーティファクトのセット。

出力変数

このアクションは、変数として PutJobSuccessResult API リクエストのセクション `outputVariables` に含まれるすべてのキー値のペアを生成します。

CodePipeline における変数の詳細については、変数リファレンス を参照してください。

アクション設定の例

YAML

```
Name: Lambda
Actions:
  - Name: Lambda
    ActionTypeId:
      Category: Invoke
      Owner: AWS
      Provider: Lambda
      Version: '1'
    RunOrder: 1
    Configuration:
      FunctionName: myLambdaFunction
      UserParameters: 'http://192.0.2.4'
    OutputArtifacts: []
    InputArtifacts: []
    Region: us-west-2
```

JSON

```
{
    "Name": "Lambda",
    "Actions": [
        {
            "Name": "Lambda",
            "ActionTypeId": {
                "Category": "Invoke",
                "Owner": "AWS",
                "Provider": "Lambda",
                "Version": "1"
            },
            "RunOrder": 1,
            "Configuration": {
                "FunctionName": "myLambdaFunction",
                "UserParameters": "http://192.0.2.4"
            },
            "OutputArtifacts": [],
            "InputArtifacts": [],
            "Region": "us-west-2"
        }
    ]
},
```

JSON イベントの例

Lambda アクションは、ジョブ ID、パイプラインアクション設定、入力および出力アーティファクトの場所、およびアーティファクトの暗号化情報を含む JSON イベントを送信します。ジョブワーカーは、これらの詳細にアクセスして Lambda アクションを完了します。詳細については、ジョブの詳細を参照してください。以下に示しているのは、イベントの例です。

```
{
    "CodePipeline.job": {
        "id": "11111111-abcd-1111-abcd-111111abcdef",
        "accountId": "111111111111",
        "data": {
            "actionConfiguration": {
                "configuration": {
                    "FunctionName": "MyLambdaFunction",
                    "UserParameters": "input_parameter"
```

```
                }
            },
            "inputArtifacts": [
                {
                    "location": {
                        "s3Location": {
                            "bucketName": "bucket_name",
                            "objectKey": "filename"
                        },
                        "type": "S3"
                    },
                    "revision": null,
                    "name": "ArtifactName"
                }
            ],
            "outputArtifacts": [],
            "artifactCredentials": {
                "secretAccessKey": "secret_key",
                "sessionToken": "session_token",
                "accessKeyId": "access_key_ID"
            },
            "continuationToken": "token_ID",
            "encryptionKey": {
              "id": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
              "type": "KMS"
            }
        }
    }
}
```

JSON イベントは、CodePipeline の Lambda アクションの次のジョブ詳細を提供します。

- id: システムによって生成されたジョブの固有の ID。
- accountId: ジョブに関連付けられた AWS アカウント ID。
- data: ジョブワーカーがジョブを完了するために必要なその他の情報。
  - actionConfiguration: Lambda アクションのアクションパラメータ。定義については、設定パラメータ を参照してください。
  - inputArtifacts: アクションに指定されたアーティファクト。
    - location: アーティファクトストアの場所。
      - s3Location: アクションの入力アーティファクトの場所情報。

        - bucketName: アクションのパイプラインアーティファクトストアの名前（例: codepipeline-us-east-2-1234567890という名前の Amazon S3 バケット）。
        - objectKey: アプリケーションの名前（例: CodePipelineDemoApplication.zip）。
      - type: ロケーション内のアーティファクトのタイプ。現在、S3 は唯一の有効なアーティファクトタイプです。
    - revision: アーティファクトのリビジョン ID。オブジェクトのタイプに応じて、コミット ID (GitHub) またはリビジョン ID（Amazon Simple Storage Service）になります。詳細については、ArtifactRevision を参照してください。
    - name: MyApp などの作業するアーティファクトの名前。
  - outputArtifacts: アクションの出力。
    - location: アーティファクトストアの場所。
      - s3Location: アクションの出力アーティファクトの場所情報。
        - bucketName: アクションのパイプラインアーティファクトストアの名前（例: codepipeline-us-east-2-1234567890という名前の Amazon S3 バケット）。
        - objectKey: アプリケーションの名前（例: CodePipelineDemoApplication.zip）。
      - type: ロケーション内のアーティファクトのタイプ。現在、S3 は唯一の有効なアーティファクトタイプです。
    - revision: アーティファクトのリビジョン ID。オブジェクトのタイプに応じて、コミット ID (GitHub) またはリビジョン ID（Amazon Simple Storage Service）になります。詳細については、ArtifactRevision を参照してください。
    - name: MyApp などのアーティファクトの出力の名前。
  - artifactCredentials: Amazon S3 バケットの入力アーティファクトと出力アーティファクトへのアクセスに使用される AWS セッション認証情報。これらの認証情報は、 AWS Security Token Service （AWS STS）によって発行される一時的な認証情報です。
    - secretAccessKey: セッションのシークレットアクセスキー。
    - sessionToken: セッションのトークン。
    - accessKeyId: セッションのアクセスキー ID。
  - continuationToken: アクションによって生成されたトークン。今後のアクションでは、このトークンを使用して、アクションの実行中のインスタンスを識別します。アクションが完了すると、継続トークンは指定されません。

- `encryptionKey`: キーなど、アーティファクトストア内のデータの暗号化に使用される暗号化 AWS KMS キー。これが未定義の場合は、Amazon Simple Storage Service のデフォルトキーが使用されます。
  - `id`: キーを識別するために使用された ID。 AWS KMS キーの場合、キー ID、キー ARN、またはエイリアス ARN を使用できます。
  - `type`: AWS KMS などの暗号化キーのタイプ。

関連情報

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- AWS CloudFormation ユーザーガイド – パイプラインの Lambda アクションと CloudFormation アーティファクトの詳細については、CodePipeline Pipelines でのパラメータオーバーライド関数の使用」、「Lambda ベースのアプリケーションのデプロイの自動化」、および AWS CloudFormation 「アーティファクト」を参照してください。
- CodePipeline のパイプラインで AWS Lambda 関数を呼び出す - この手順では、サンプルの Lambda 関数を示し、コンソールを使用して Lambda 呼び出しアクションでパイプラインを作成する方法を示します。

AWS OpsWorks デプロイアクションリファレンス

AWS OpsWorks アクションを使用して、パイプラインを使用して OpsWorks でデプロイします。

アクションタイプ

- カテゴリ:`Deploy`
- 所有者: `AWS`
- プロバイダー: `OpsWorks`
- バージョン: `1`

設定パラメータ

アプリケーション

必須: はい

OpsWorks アプリ。アプリは、デプロイして実行するコードを表します。

スタック

必須: はい

OpsWorks スタック。スタックは、アプリケーションインフラストラクチャのコンテナです。

レイヤー

必須: いいえ

OpsWorks レイヤー。レイヤーは、一連のインスタンスの設定とリソースを指定します。

入力アーティファクト

- アーティファクトの数: 1
- 説明: これはアクションの入力アーティファクトです。

出力アーティファクト

- アーティファクトの数: 0 to 1
- 説明: 出力アーティファクトは、このアクションタイプには適用されません。

サービスロールのアクセス許可: AWS OpsWorks アクション

AWS OpsWorks サポートを受けるには、ポリシーステートメントに以下を追加します。

```
{
    "Effect": "Allow",
    "Action": [
        "opsworks:CreateDeployment",
        "opsworks:DescribeApps",
        "opsworks:DescribeCommands",
        "opsworks:DescribeDeployments",
        "opsworks:DescribeInstances",
        "opsworks:DescribeStacks",
        "opsworks:UpdateApp",
        "opsworks:UpdateStack"
```

```
    ],
    "Resource": "resource_ARN"
},
```

アクション設定の例

YAML

```
Name: ActionName
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Version: 1
  Provider: OpsWorks
InputArtifacts:
  - Name: myInputArtifact
Configuration:
  Stack: my-stack
  App: my-app
```

JSON

```
{
    "Name": "ActionName",
    "ActionTypeId": {
        "Category": "Deploy",
        "Owner": "AWS",
        "Version": 1,
        "Provider": "OpsWorks"
    },
    "InputArtifacts": [
        {
            "Name": "myInputArtifact"
        }
    ],
    "Configuration": {
        "Stack": "my-stack",
        "App": "my-app"
    }
}
```

関連情報

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- AWS OpsWorks ユーザーガイド – でのデプロイの詳細については AWS OpsWorks、AWS OpsWorks 「 ユーザーガイド」を参照してください。

AWS Service Catalog デプロイアクションリファレンス

AWS Service Catalog アクションを使用して、パイプラインを使用してテンプレートをデプロイします。これらは、Service Catalog で作成したリソーステンプレートです。

アクションタイプ

- カテゴリ:`Deploy`
- 所有者: `AWS`
- プロバイダー: `ServiceCatalog`
- バージョン: `1`

設定パラメータ

TemplateFilePath

必須: はい

ソースの場所にあるリソーステンプレートのファイルパス。

ProductVersionName

必須: はい

Service Catalog の製品バージョン。

ProductType

必須: はい

Service Catalog の製品タイプ。

ProductId

必須: はい

Service Catalog の製品 ID。

ProductVersionDescription

必須: いいえ

Service Catalog の製品バージョンの説明。

入力アーティファクト

- アーティファクトの数: 1
- 説明: これはアクションの入力アーティファクトです。

出力アーティファクト

- アーティファクトの数: 0
- 説明: 出力アーティファクトは、このアクションタイプには適用されません。

サービスロールのアクセス許可: Service Catalog アクション

Service Catalog がサポートされるように、以下をポリシーステートメントに追加します。

```
{
    "Effect": "Allow",
    "Action": [
        "servicecatalog:ListProvisioningArtifacts",
        "servicecatalog:CreateProvisioningArtifact",
        "servicecatalog:DescribeProvisioningArtifact",
        "servicecatalog:DeleteProvisioningArtifact",
        "servicecatalog:UpdateProduct"
    ],
    "Resource": "resource_ARN"
},
{
    "Effect": "Allow",
    "Action": [
```

```
        "cloudformation:ValidateTemplate"
    ],
    "Resource": "resource_ARN"
}
```

設定ファイルのタイプ別のアクション設定の例

次の例は、個別の設定ファイルを使用しないでパイプラインをコンソールで作成する場合に、Service Catalog を使用するデプロイアクションの有効な設定を示しています。

```
"configuration": {
  "TemplateFilePath": "S3_template.json",
  "ProductVersionName": "devops S3 v2",
  "ProductType": "CLOUD_FORMATION_TEMPLATE",
  "ProductVersionDescription": "Product version description",
  "ProductId": "prod-example123456"
}
```

次の例は、個別の `sample_config.json` 設定ファイルを使用してパイプラインをコンソールで作成する場合に、Service Catalog を使用するデプロイアクションの有効な設定を示しています。

```
"configuration": {
  "ConfigurationFilePath": "sample_config.json",
  "ProductId": "prod-example123456"
}
```

アクション設定の例

YAML

```
Name: ActionName
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Version: 1
  Provider: ServiceCatalog
OutputArtifacts:
- Name: myOutputArtifact
Configuration:
  TemplateFilePath: S3_template.json
  ProductVersionName: devops S3 v2
  ProductType: CLOUD_FORMATION_TEMPLATE
```

```
  ProductVersionDescription: Product version description
  ProductId: prod-example123456
```

JSON

```
{
    "Name": "ActionName",
    "ActionTypeId": {
        "Category": "Deploy",
        "Owner": "AWS",
        "Version": 1,
        "Provider": "ServiceCatalog"
    },
    "OutputArtifacts": [
        {
            "Name": "myOutputArtifact"
        }
    ],
    "Configuration": {
        "TemplateFilePath": "S3_template.json",
        "ProductVersionName": "devops S3 v2",
        "ProductType": "CLOUD_FORMATION_TEMPLATE",
        "ProductVersionDescription": "Product version description",
        "ProductId": "prod-example123456"
    }
}
```

関連情報

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- Service Catalog ユーザーガイド – Service Catalog のリソースとテンプレートの詳細については、「Service Catalog ユーザーガイド」を参照してください。
- チュートリアル: Service Catalog にデプロイするパイプラインを作成する – このチュートリアルでは、製品テンプレートを Service Catalog にデプロイし、ソースリポジトリで行った変更を配信するためのパイプラインを作成して設定する方法を示します。

AWS Step Functions アクションリファレンスを呼び出す

以下を実行する AWS CodePipeline アクション。

- パイプラインから AWS Step Functions ステートマシンの実行を開始します。
- アクション設定のプロパティ、または入力として渡されるパイプラインアーティファクトにあるファイルのいずれかを使用して、ステートマシンに初期状態を提供します。
- オプションで、アクションから発生した実行を識別するための実行 ID プレフィックスを設定します。
- 標準および Express ステートマシンをサポートします。

> **Note**
>
> Step Functions アクションは Lambda で実行するため、Lambda 関数のアーティファクトサイズクォータと同じアーティファクトサイズクォータが適用されます。詳細については、「Lambda デベロッパーガイド」の「Lambda クォータ」を参照してください。

アクションタイプ

- カテゴリ:`Invoke`
- 所有者: `AWS`
- プロバイダー: `StepFunctions`
- バージョン: `1`

設定パラメータ

StateMachineArn

必須: はい

呼び出されるステートマシンの Amazon リソースネーム (ARN)。

ExecutionNamePrefix

必須: いいえ

デフォルトでは、アクション実行 ID がステートマシンの実行名として使用されます。プレフィックスが指定されている場合は、アクション実行 ID の先頭にハイフンが付加され、ステートマシンの実行名として使用されます。

```
myPrefix-1624a1d1-3699-43f0-8e1e-6bafd7fde791
```

Express ステートマシンの場合、名前には 0 ~ 9、A ~ Z、a ~ z、- および _ のみを含める必要があります。

InputType

必須: いいえ

- [Literal (リテラル)] (デフォルト): 指定すると、[Input (入力)] フィールドの値がステートマシンの入力に直接渡されます。

  リテラルを選択した場合の、Input フィールドの入力例。

  ```
  {"action": "test"}
  ```

- [FilePath] : [Input (入力)] フィールドで指定された入力アーティファクトのファイルの内容が、ステートマシン実行の入力として使用されます。[InputType] タイプが [FilePath] に設定されているときは、入力アーティファクトが必要です。

  リテラル を選択した場合の、Input フィールドの入力例。

  ```
  assets/input.json
  ```

Input

必須: 条件による

- [Literal (リテラル)] : [InputType] が [Literal (リテラル)] (デフォルト) に設定されている場合、このフィールドはオプションです。

  指定されている場合、[入力] フィールドはステートマシン実行の入力として直接使用されます。それ以外の場合は、空の JSON オブジェクト {} を使用してステートマシンが呼び出されます。

- [FilePath] : [InputType] が [FilePath] に設定されている場合、このフィールドは必須です。

  [InputType] が [FilePath] に設定されている場合は、入力アーティファクトも必須です。

  指定された入力アーティファクトのファイルの内容は、ステートマシン実行の入力として使用されます。

入力アーティファクト

- アーティファクトの数: `0 to 1`
- 説明: [InputType] が [FilePath] に設定されている場合、このアーティファクトは必須であり、ステートマシンの実行のための入力のソースに使用されます。

出力アーティファクト

- アーティファクトの数: `0 to 1`
- 説明:
  - 標準ステートマシン: 指定すると、出力アーティファクトには、ステートマシンの出力が入力されます。これは、ステートマシンの実行が正常に終了した後、ステップ関数 DescribeExecution API 応答の`output` プロパティから取得できます。
  - Express ステートマシン: サポートされていません。

出力変数

このアクションにより、パイプライン内のダウンストリームアクションのアクション設定によって参照できる出力変数が生成されます。

詳細については、「変数リファレンス」を参照してください。

StateMachineArn

ステートマシンの ARN。

ExecutionArn

ステートマシンの実行の ARN。標準ステートマシンのみ。

サービスロールのアクセス許可: `StepFunctions` アクション

`StepFunctions` アクションでは、Step Functions 呼び出しアクションを使用してパイプラインを作成するために必要な最小限のアクセス許可は次のとおりです。

```
{
    "Effect": "Allow",
    "Action": [
```

```
        "states:DescribeStateMachine",
        "states:DescribeExecution",
        "states:StartExecution"
    ],
    "Resource": "resource_ARN"
},
```

アクション設定の例

デフォルト入力の例

YAML

```
Name: ActionName
ActionTypeId:
  Category: Invoke
  Owner: AWS
  Version: 1
  Provider: StepFunctions
OutputArtifacts:
  - Name: myOutputArtifact
Configuration:
  StateMachineArn: arn:aws:states:us-east-1:111122223333:stateMachine:HelloWorld-
StateMachine
  ExecutionNamePrefix: my-prefix
```

JSON

```
{
    "Name": "ActionName",
    "ActionTypeId": {
        "Category": "Invoke",
        "Owner": "AWS",
        "Version": 1,
        "Provider": "StepFunctions"
    },
    "OutputArtifacts": [
        {
            "Name": "myOutputArtifact"
        }
    ],
    "Configuration": {
```

```
        "StateMachineArn": "arn:aws:states:us-
east-1:111122223333:stateMachine:HelloWorld-StateMachine",
        "ExecutionNamePrefix": "my-prefix"
    }
}
```

リテラル入力の例

YAML

```
Name: ActionName
ActionTypeId:
  Category: Invoke
  Owner: AWS
  Version: 1
  Provider: StepFunctions
OutputArtifacts:
  - Name: myOutputArtifact
Configuration:
  StateMachineArn: arn:aws:states:us-east-1:111122223333:stateMachine:HelloWorld-
StateMachine
  ExecutionNamePrefix: my-prefix
  Input: '{"action": "test"}'
```

JSON

```
{
    "Name": "ActionName",
    "ActionTypeId": {
        "Category": "Invoke",
        "Owner": "AWS",
        "Version": 1,
        "Provider": "StepFunctions"
    },
    "OutputArtifacts": [
        {
            "Name": "myOutputArtifact"
        }
    ],
    "Configuration": {
        "StateMachineArn": "arn:aws:states:us-
east-1:111122223333:stateMachine:HelloWorld-StateMachine",
```

```
        "ExecutionNamePrefix": "my-prefix",
        "Input": "{\"action\": \"test\"}"
    }
}
```

入力ファイルの例

YAML

```
Name: ActionName
InputArtifacts:
  - Name: myInputArtifact
ActionTypeId:
  Category: Invoke
  Owner: AWS
  Version: 1
  Provider: StepFunctions
OutputArtifacts:
  - Name: myOutputArtifact
Configuration:
  StateMachineArn: 'arn:aws:states:us-east-1:111122223333:stateMachine:HelloWorld-
StateMachine'
  ExecutionNamePrefix: my-prefix
  InputType: FilePath
  Input: assets/input.json
```

JSON

```
{
    "Name": "ActionName",
    "InputArtifacts": [
        {
            "Name": "myInputArtifact"
        }
    ],
    "ActionTypeId": {
        "Category": "Invoke",
        "Owner": "AWS",
        "Version": 1,
        "Provider": "StepFunctions"
    },
    "OutputArtifacts": [
```

```
        {
            "Name": "myOutputArtifact"
        }
    ],
    "Configuration": {
        "StateMachineArn": "arn:aws:states:us-
east-1:111122223333:stateMachine:HelloWorld-StateMachine",
        "ExecutionNamePrefix": "my-prefix",
        "InputType": "FilePath",
        "Input": "assets/input.json"
    }
}
```

行動

リリース中、CodePipeline はアクション設定で指定された入力を使用して設定されたステートマシンを実行します。

[InputType] が [Literal (リテラル)] に設定されている場合、[Input (入力)] アクション設定フィールドの内容がステートマシンの入力として使用されます。リテラル入力が指定されていない場合、ステートマシンの実行では空の JSON オブジェクト {} が使用されます。入力なしでステートマシンを実行する方法の詳細については、「ステップ関数 StartExecution API」を参照してください。

[InputType] が [FilePath] に設定されている場合、アクションは入力アーティファクトを解凍し、[Input (入力)] アクション設定フィールドで指定されたファイルの内容をステートマシンの入力として使用します。[FilePath] が指定されている場合、[Input (入力)] フィールドは必須で、入力アーティファクトが存在する必要があります。そうでない場合、アクションは失敗します。

起動が正常に実行されると、標準と Express の 2 つのステートマシンタイプに対して動作が分岐します。

標準ステートマシン

標準ステートマシンの実行が正常に開始された場合、CodePipeline は実行がターミナルステータスに達するまで `DescribeExecution` API をポーリングします。実行が正常に完了するとアクションは成功し、それ以外の場合は失敗します。

出力アーティファクトが設定されている場合、アーティファクトにはステートマシンの戻り値が含まれます。これは、ステートマシンの実行が正常に終了した後、ステップ関数 DescribeExecution API

応答のoutput プロパティから取得できます。この API には出力長の制約が適用されることに注意してください。

エラー処理

- アクションがステートマシンの実行を開始できない場合、アクションの実行は失敗します。
- CodePipeline ステップ関数アクションがタイムアウト (デフォルトは 7 日間) に達する前にステートマシンの実行がターミナルステータスに到達しなかった場合、アクションの実行は失敗します。この障害にもかかわらず、ステートマシンは続行される可能性があります。ステップ関数でのステートマシンの実行タイムアウトの詳細については、「標準ワークフローと Express ワークフロー」を参照してください。

  > **Note**
  >
  > アクションを持つアカウントの呼び出しアクションタイムアウトのクォータの引き上げをリクエストできます。ただし、クォータの引き上げは、そのアカウントのすべてのリージョンで、このタイプのすべてのアクションに適用されます。

- ステートマシンの実行が FAILED、TIMED_OUT、または ABORTED のターミナルステータスに達すると、アクションの実行は失敗します。

Express ステートマシン

Express ステートマシンの実行が正常に開始されると、呼び出しアクションの実行は正常に完了します。

Express ステートマシン用に設定されたアクションに関する考慮事項。

- 出力アーティファクトは指定できません。
- アクションは、ステートマシンの実行が完了するまで待機しません。
- CodePipeline でアクションの実行が開始されると、ステートマシンの実行が失敗した場合でも、アクションの実行は成功します。

エラー処理

- CodePipeline がステートマシンの実行のスタートに失敗すると、アクションの実行は失敗します。それ以外の場合、アクションはすぐに成功します。CodePipeline のアクションは、ステートマシンの実行が完了するまでの時間、またはその結果に関係なく、成功します。

関連情報

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- AWS Step Functions デベロッパーガイド – ステートマシン、実行、およびステートマシンの入力については、「 AWS Step Functions デベロッパーガイド」を参照してください。
- チュートリアル: パイプラインで AWS Step Functions 呼び出しアクションを使用する - このチュートリアルでは、サンプルの標準ステートマシンから開始し、コンソールを使用してステップ関数の呼び出しアクションを追加してパイプラインを更新する方法について説明します。

ルール構造リファレンス

このセクションは、ルール設定のみのリファレンスです。パイプライン構造の概念的な概要については、「CodePipeline パイプライン構造リファレンス」を参照してください。

CodePipeline の各ルールプロバイダーは、パイプライン構造の必須およびオプションの設定フィールドのセットを使用します。このセクションでは、以下のリファレンス情報をルールプロバイダー別に提供します。

- パイプライン構造のルールブロックに含まれている `RuleType` フィールドの有効な値 (`Owner` や `Provider` など)。
- パイプライン構造のルールセクションに含まれている `Configuration` パラメータ (必須およびオプション) の説明およびその他のリファレンス情報。
- 有効な JSON および YAML ルール設定フィールドの例。

以下のルールプロバイダーのリファレンス情報が利用可能です。

トピック

- CloudwatchAlarm
- CodeBuild ルール
- コマンド
- DeploymentWindow
- LambdaInvoke
- VariableCheck

CloudwatchAlarm

条件を作成するときに、`CloudWatchAlarm` ルールを追加できます。このセクションでは、ルールパラメータのリファレンスを提供します。ルールと条件の詳細については、「ステージ条件はどのように機能しますか?」を参照してください。

Amazon CloudWatch で別のリソースとしてアラームを既に作成している必要があります。

トピック

- ルールタイプ

- 設定パラメータ
- ルール設定の例
- 関連情報

ルールタイプ

- カテゴリ: `Rule`
- 所有者: `AWS`
- プロバイダー: `CloudWatchAlarm`
- バージョン: `1`

設定パラメータ

AlarmName

必須: はい

CloudWatch アラームの名前。これは、CloudWatch で作成した別のリソースです。

AlarmStates

必須: いいえ

ルールでモニタリングする CloudWatch アラームの状態。有効値は、ALARM、OK、INSUFFICIENT_DATA です。

WaitTime

必須: いいえ

ルール結果を実行するまでに状態変更を許可する待機時間 (分単位)。例えば、ルール結果を適用するまでにアラーム状態が [OK] に変わるのを 20 分待つように設定します。

ルール設定の例

YAML

```
rules:
```

```
    - name: MyMonitorRule
      ruleTypeId:
        category: Rule
        owner: AWS
        provider: CloudWatchAlarm
        version: '1'
      configuration:
        AlarmName: CWAlarm
        WaitTime: '1'
      inputArtifacts: []
      region: us-east-1
```

JSON

```
                    "rules": [
                        {
                            "name": "MyMonitorRule",
                            "ruleTypeId": {
                                "category": "Rule",
                                "owner": "AWS",
                                "provider": "CloudWatchAlarm",
                                "version": "1"
                            },
                            "configuration": {
                                "AlarmName": "CWAlarm",
                                "WaitTime": "1"
                            },
                            "inputArtifacts": [],
                            "region": "us-east-1"
                        }
                    ]
                }
```

関連情報

このルールを利用する際に以下の関連リソースが役立ちます。

- 失敗時の条件の作成 – このセクションでは、アラームルールを使用して失敗時の条件を作成する手順を示しています。

CodeBuild ルール

条件を作成する際、CodeBuild ルールを追加できます。このセクションでは、ルールパラメータのリファレンスを提供します。ルールと条件の詳細については、「ステージ条件はどのように機能しますか?」を参照してください。

CodeBuild ルールを使用して、ビルド実行が beforeEntry 条件で正常に完了するなど、ビルドプロジェクトがルールの基準を満たす条件を作成できます。

> **Note**
>
> スキップ結果で設定された beforeEntry 条件では、`LambdaInvoke` および `VariableCheck` のルールのみを使用できます。

トピック

- サービスロールのポリシーのアクセス許可
- ルールタイプ
- 設定パラメータ
- ルール設定の例
- 関連情報

サービスロールのポリシーのアクセス許可

このルールのアクセス許可については、CodePipeline サービスロールのポリシーステートメントに以下を追加します。アクセス許可をリソースレベルに絞り込みます。

```
{
    "Effect": "Allow",
    "Action": [
        "codebuild:BatchGetBuilds",
        "codebuild:StartBuild"
    ],
    "Resource": "resource_ARN"
},
```

ルールタイプ

- カテゴリ: `Rule`
- 所有者: `AWS`
- プロバイダー: `CodeBuild`
- バージョン: `1`

設定パラメータ

ProjectName

必須: はい

`ProjectName` は、CodeBuild のビルドプロジェクト名です。

PrimarySource

必須: 条件による

`PrimarySource` パラメータの値は、アクションへの入力アーティファクトの名前の 1 つでなければなりません。CodeBuild は、このアーティファクトを解凍したものを含むディレクトリで buildspec のファイルを探し、buildspec コマンドを実行します。

このパラメータは、CodeBuild アクションに複数の入力アーティファクトが指定されている場合に必要となります。アクションのソースアーティファクトが 1 つだけの場合、`PrimarySource` アーティファクトはデフォルトでそのアーティファクトになります。

BatchEnabled

必須: いいえ

この `BatchEnabled` パラメータのブール値は、アクションが同じビルド実行で複数のビルドを実行することを可能にします。

このオプションを有効にすると、`CombineArtifacts` オプションが使用できます。

バッチビルドを有効にしたパイプラインの例については、「CodePipeline と CodeBuild およびバッチビルドの統合」を参照してください。

CombineArtifacts

必須: いいえ

この CombineArtifacts パラメータのブール値は、バッチ ビルドからのすべてのビルド アーティファクトをビルド アクションのための単一の アーティファクト ファイルにまとめます。

このオプションを使用するには、BatchEnabled パラメータを有効にする必要があります。

EnvironmentVariables

必須: いいえ

このパラメータの値は、パイプラインの CodeBuild アクションの環境変数を設定するために使用されます。EnvironmentVariables パラメータの値は、環境変数オブジェクトの JSON 配列の形式をとります。「アクション宣言 (CodeBuild の例)」のパラメータ例を参照してください。

各オブジェクトには 3 つの部分があり、それらはすべて文字列です。

- name: 環境変数の名前またはキー。
- value: 環境変数の値。PARAMETER_STORE または SECRETS_MANAGER タイプを使用する場合、この値はそれぞれ AWS Systems Manager Parameter Store に保存済みのパラメータ名か、AWS Secrets Manager に保存済みのシークレット名である必要があります。

  > **Note**
  >
  > 機密情報 (特に AWS 認証情報) を保存する場合は、環境変数を使用しないことを強くお勧めします。CodeBuild コンソールまたは AWS CLI を使用すると、環境変数がプレーンテキストで表示されます。機密の値の場合は、代わりに SECRETS_MANAGER タイプを使用することをお勧めします。

- type: (任意) 環境変数の型。有効な値は PARAMETER_STORE、SECRETS_MANAGER、または PLAINTEXT です。指定しない場合、この値はデフォルトで PLAINTEXT になります。

> **Note**
>
> 環境変数の設定に name、value、および type を入力する場合 (特に環境変数に CodePipeline の出力変数の構文が含まれている場合) は、設定の値フィールドの 1000 文字制限を超えないようにしてください。この制限を超えると、検証エラーが返されます。

詳細については、AWS CodeBuild API リファレンスの「EnvironmentVariable」を参照してください。GitHub ブランチ名に解決される環境変数を持つ CodeBuild アクションの例については、例:CodeBuild 環境変数で BranchName 変数を使用する を参照してください。

ルール設定の例

YAML

```
name: codebuild-rule
ruleTypeId:
  category: Rule
  owner: AWS
  provider: CodeBuild
  version: '1'
configuration:
  ProjectName: my-buildproject
  EnvironmentVariables: '[{"name":"VAR1","value":"variable","type":"PLAINTEXT"}]'
inputArtifacts:
- name: SourceArtifact
region: us-east-1
```

JSON

```
{
    "name": "codebuild-rule",
    "ruleTypeId": {
        "category": "Rule",
        "owner": "AWS",
        "provider": "CodeBuild",
        "version": "1"
    },
    "configuration": {
        "ProjectName": "my-buildproject"
    },
    "inputArtifacts": [
        {
            "name": "SourceArtifact",
            "EnvironmentVariables": "[{\"name\":\"VAR1\",\"value\":\"variable\",
\"type\":\"PLAINTEXT\"}]"
        }
    ],
    "region": "us-east-1"
}
```

関連情報

このルールを利用する際に以下の関連リソースが役立ちます。

- ルールと条件の詳細については、「CodePipeline API ガイド」の「条件」、「RuleTypeId」、「RuleExecution」を参照してください。

コマンド

条件を作成するときに、Commands ルールを追加できます。このセクションでは、ルールパラメータのリファレンスを提供します。ルールと条件の詳細については、「ステージ条件はどのように機能しますか?」を参照してください。

Commands ルールを使用して、コマンドの出力やファイルパスが beforeEntry 条件を満たすなど、コマンドの正常な完了のルール基準で条件を作成することができます。

> **Note**
>
> スキップ結果で設定された beforeEntry 条件では、LambdaInvoke および VariableCheck のルールのみを使用できます。

トピック

- コマンドルールに関する考慮事項
- サービスロールのポリシーのアクセス許可
- ルールタイプ
- 設定パラメータ
- ルール設定の例
- 関連情報

コマンドルールに関する考慮事項

コマンドルールには以下の考慮事項が適用されます。

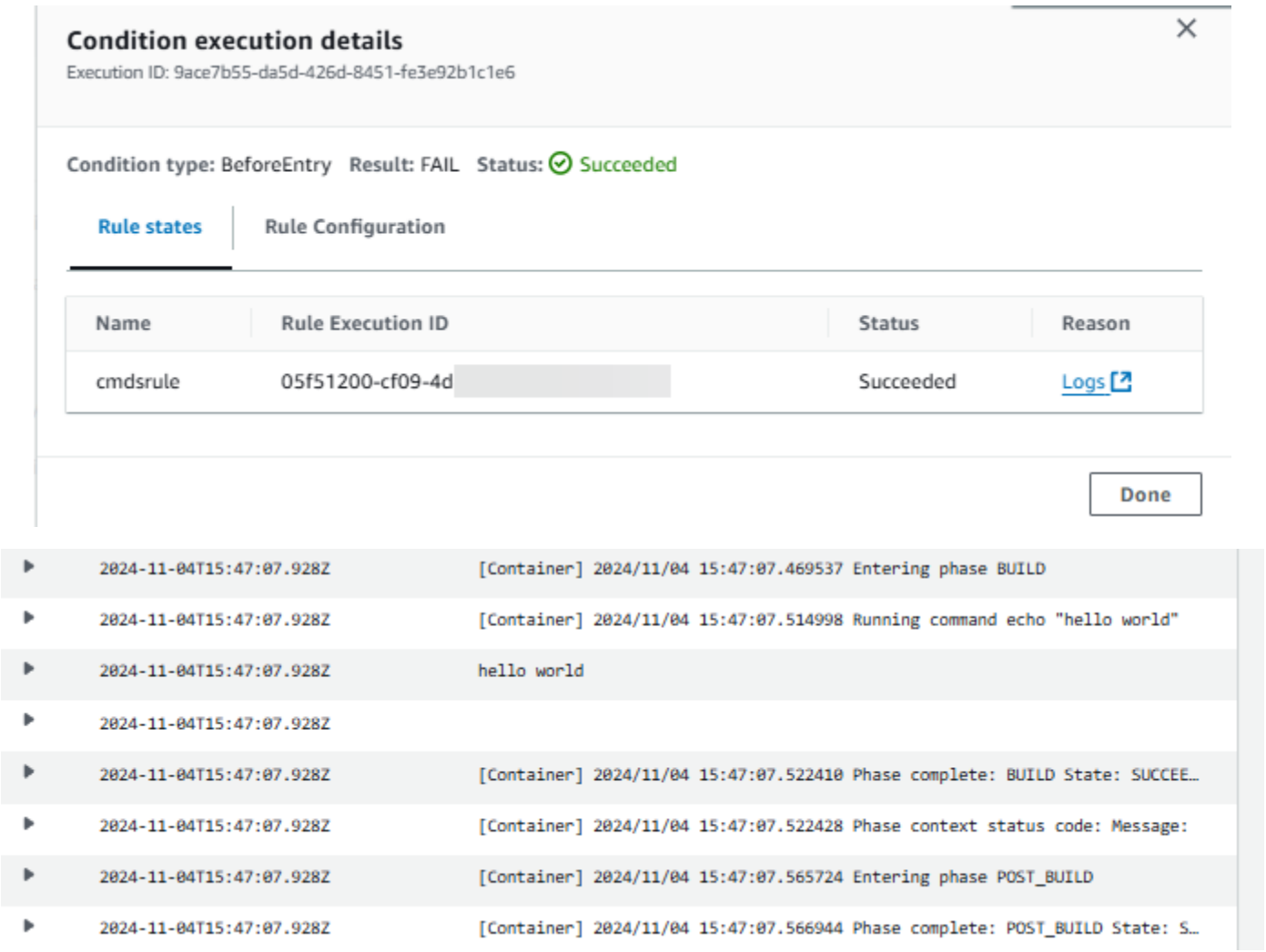
- コマンドルールは CodeBuild アクションと同様の CodeBuild リソースを使用しますが、ビルドプロジェクトを関連付けたり作成したりする必要はなく、仮想コンピューティングインスタンス内でシェル環境コマンドが許可されます。

> **Note**
>
> コマンドルールを実行すると、個別の料金が発生します AWS CodeBuild。

- CodePipeline のコマンドルールは CodeBuild リソースを使用するため、アクションで実行するビルドには、アカウントに対する CodeBuild のビルド制限が適用されます。コマンドルールで実行したビルドは、アカウントに設定されている同時ビルド制限にカウントされます。
- コマンドルールを使用したビルドのタイムアウトは、CodeBuild ビルドに基づいて 55 分です。
- コンピューティングインスタンスは、CodeBuild の分離されたビルド環境を使用します。

> **Note**
>
> 分離されたビルド環境はアカウントレベルで使用されるため、インスタンスは別のパイプライン実行に再利用される場合があります。

- 複数行形式を除くすべての形式がサポートされています。コマンドを入力するときは、単一行形式を使用する必要があります。
- このルールでは、CodePipeline がパイプラインサービスロールを引き受け、このロールを使用してランタイムにリソースへのアクセスを許可します。アクセス許可の範囲をアクションレベルまで絞り込むように、サービスロールを設定することをお勧めします。
- CodePipeline サービスロールに追加されるアクセス許可の詳細については、「CodePipeline サービスロールにアクセス許可を追加する」を参照してください。
- コンソールでログを表示するために必要なアクセス許可の詳細については、「コンソールでコンピューティングログを表示するために必要なアクセス許可」を参照してください。次の画面の例では、ログリンクを使用して、CloudWatch Logs で正常に完了したコマンドルールのログを表示します。



- CodePipeline の他のアクションとは異なり、アクション設定ではフィールドを設定せず、アクション設定の外部でアクション設定フィールドを設定します。

サービスロールのポリシーのアクセス許可

CodePipeline は、ルールを実行するときに、次のようにパイプライン名を使用してロググループを作成します。これにより、パイプライン名を使用してアクセス許可の範囲をリソースのログ記録に絞り込むことができます。

```
/aws/codepipeline/MyPipelineName
```

既存のサービスロールを使用している場合、コマンドアクションを使用するには、サービスロールに以下のアクセス許可を追加する必要があります。

- logs:CreateLogGroup

- logs:CreateLogStream
- logs:PutLogEvents

サービスロールポリシーステートメントで、次の例に示すように、アクセス許可の範囲をパイプラインレベルに絞り込みます。

```
{
    "Effect": "Allow",
    "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
    ],
    "Resource": "arn:aws:logs:*:YOUR_AWS_ACCOUNT_ID:log-group:/aws/
codepipeline/YOUR_PIPELINE_NAME:*"
}
```

アクションの詳細ダイアログページを使用してコンソールでログを表示するには、ログを表示するアクセス許可をコンソールロールに追加する必要があります。詳細については、「コンソールでコンピューティングログを表示するために必要なアクセス許可」でコンソールのアクセス許可ポリシーの例を参照してください。

ルールタイプ

- カテゴリ: `Rule`
- 所有者: `AWS`
- プロバイダー: `Commands`
- バージョン: `1`

設定パラメータ

コマンド

必須: はい

Commands ルールで実行するシェルコマンドを指定できます。コンソールでは、各コマンドを個別の行に入力します。CLI では、コマンドを個別の文字列として入力します。

> **Note**
>
> 複数行形式はサポートされていないため、エラーメッセージが表示されます。[コマンド] フィールドにコマンドを入力するには、単一行形式を使用する必要があります。

以下の詳細により、コマンドルールに使用するデフォルトのコンピューティングを指定します。詳細については、「CodeBuild ユーザーガイド」の「ビルド環境のコンピューティングモードおよびタイプ」リファレンスを参照してください。

- CodeBuild イメージ: aws/codebuild/amazonlinux2-x86_64-standard:5.0
- コンピューティングタイプ: Linux Small
- 環境の computeType 値: BUILD_GENERAL1_SMALL
- 環境タイプの値: LINUX_CONTAINER

ルール設定の例

YAML

```
result: FAIL
rules:
- name: CommandsRule
  ruleTypeId:
    category: Rule
    owner: AWS
    provider: Commands
    version: '1'
  configuration: {}
  commands:
  - ls
  - printenv
  inputArtifacts:
  - name: SourceArtifact
  region: us-east-1
```

JSON

```
{
    "result": "FAIL",
    "rules": [
```

```
        {
            "name": "CommandsRule",
            "ruleTypeId": {
                "category": "Rule",
                "owner": "AWS",
                "provider": "Commands",
                "version": "1"
            },
            "configuration": {},
            "commands": [
                "ls",
                "printenv"
            ],
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-east-1"
        }
    ]
}
```

関連情報

このルールを利用する際に以下の関連リソースが役立ちます。

- ルールと条件の詳細については、「CodePipeline API ガイド」の「条件」、「RuleTypeId」、「RuleExecution」を参照してください。

DeploymentWindow

条件を作成するときに、DeploymentWindow ルールを追加できます。このセクションでは、ルールパラメータのリファレンスを提供します。ルールと条件の詳細については、「ステージ条件はどのように機能しますか?」を参照してください。

トピック

- ルールタイプ
- 設定パラメータ

- ルール設定の例
- 関連情報

ルールタイプ

- カテゴリ: `Rule`
- 所有者: `AWS`
- プロバイダー: `DeploymentWindow`
- バージョン: `1`

設定パラメータ

Cron

必須: はい

デプロイを許可する日時を定義する式。cron 式は、6 つの必須フィールドと、空白で区切られた 1 つのオプションフィールドで構成されます。cron 式フィールドを使用すると、次のように cron 式でスケジュールパターンを指定できます。

フィールド名	許可される値	使用できる特殊文字
秒	該当なし	*
分	0-59	, - * /
時間	0-23	, - * /
日	1-31	, - * ? / L W
月	1-12 または JAN-DEC	, - * /
曜日	1-7 または SUN-SAT	, - * ? / L #
年 (オプション)	空、1970-2199	, - * /

- すべての値を指定するには、「*」文字を使用します。例えば、分フィールドの「*」は「毎分」を意味します。
- 「?」文字は、日フィールドと曜日フィールドで使用できます。「特定の値なし」を指定するために使用します。2 つのフィールドの一方だけに指定し、他方には指定する必要がない場合に便利です。
- 範囲を指定するには、「-」文字を使用します。例えば、時間フィールドの「10-12」は「10、11、12 時間」を意味します。
- 「,」文字は追加の値を指定するために使用します。例えば、曜日フィールドの「MON,WED,FRI」は、「月曜日、水曜日、金曜日」を意味します。
- 「/」文字は増分を指定するために使用します。例えば、秒フィールドの「0/15」は「0 秒、15 秒、30 秒、45 秒」を意味します。また、秒フィールドの「5/15」は「5 秒、20 秒、35 秒、50 秒」を意味します。「/」の前に「*」を指定することは、開始値として 0 を指定することと同じです。
- 「L」文字は、日フィールドと曜日フィールドで使用できます。この文字は「last」の省略文字ですが、2 つのフィールド間で意味が異なります。例えば、日フィールドの「L」値は「月末」を意味します。例えば、1 月 31 日、うるう年の 2 月 28 日などです。曜日フィールドで単独で使用した場合は、「7」または「土曜日」を意味します。ただし、別の値に続けて曜日フィールドで使用した場合は、「月の最後の <指定した曜日>」を意味します。例えば「6L」は「月の最後の金曜日」を意味します。また、月末からのオフセットを指定することもできます。例えば、「L-3」は月末の 3 日前を意味します。
- 「W」文字は日フィールドで使用できます。この文字は、特定の日に最も近い平日 (月 ~ 金) を指定するために使用します。例えば、日フィールドの値として「15W」と指定すると、「月の 15 日に最も近い平日」を意味します。したがって、15 日が土曜日の場合、トリガーは 14 日の金曜日に発生します。15 日が日曜日の場合、トリガーは 16 日の月曜日に発生します。15 日が火曜日の場合、トリガーは 15 日の火曜日に発生します。
- 「L」文字と「W」文字を日フィールドで組み合わせて「LW」と指定することもできます。これは、「月の最後の平日」を意味します。
- 「#」文字は曜日フィールドで使用できます。この文字は、月の「n 番目」の <指定した曜日> を指定するために使用します。例えば、曜日フィールドの「6#3」値は、月の第 3 金曜日を指定します。「6」= 金曜日、「#3」= 月の 3 番目を意味します。
- 有効な文字や、月と曜日の名前では大文字と小文字が区別されません。

TimeZone

必須: いいえ

デプロイウィンドウのタイムゾーン。正規表現は、以下の形式のパターンと一致します。

- リージョン/都市の形式。値は、リージョン/都市またはリージョン/都市_都市の形式のタイムゾーンと一致します。例えば、America/New_York、Europe/Berlin です。
- UTC 形式。値は文字列 UTC に一致し、オプションで +HH:MM または -HH:MM の形式でオフセットが続きます。例えば、UTC、UTC+05:30、または UTC-03:00 です。パラメータが特に設定されていない場合、これがデフォルト形式です。
- 略語形式。値は、タイムゾーンの 3~5 文字の略語と一致します。例えば、EST、IST です。

  有効な TimeZoneID 値の表については、https://docs.oracle.com/middleware/1221/wcs/tag-ref/MISC/TimeZones.html を参照してください。特定の略語は重複していることに注意してください。例えば、CST は標準時、中国標準時、キューバ標準時を意味します。

ルール設定の例

YAML

```
- name: MyDeploymentRule
  ruleTypeId:
    category: Rule
    owner: AWS
    provider: DeploymentWindow
    version: '1'
  configuration:
    Cron: 0 0 9-17 ? * MON-FRI *
    TimeZone: PST
  inputArtifacts: []
  region: us-east-1
```

JSON

```
[
                    {
                        "name": "MyDeploymentRule",
                        "ruleTypeId": {
                            "category": "Rule",
                            "owner": "AWS",
                            "provider": "DeploymentWindow",
                            "version": "1"
                        },
```

```
                "configuration": {
                    "Cron": "0 0 9-17 ? * MON-FRI *",
                    "TimeZone": "PST"
                },
                "inputArtifacts": [],
                "region": "us-east-1"
            }
        ]
```

関連情報

このルールを利用する際に以下の関連リソースが役立ちます。

- 成功時の条件の作成 – このセクションでは、デプロイウィンドウルールを使用して成功時の条件を作成する手順を示しています。
- ルールと条件の詳細については、「CodePipeline API ガイド」の「条件」、「RuleTypeId」、「RuleExecution」を参照してください。

LambdaInvoke

条件を作成するときに、LambdaInvoke ルールを追加できます。このセクションでは、ルールパラメータのリファレンスを提供します。ルールと条件の詳細については、「ステージ条件はどのように機能しますか?」を参照してください。

Lambda で別のリソースとして関数を既に作成している必要があります。

トピック

- ルールタイプ
- 設定パラメータ
- ルール設定の例
- 関連情報

ルールタイプ

- カテゴリ:Rule
- 所有者: AWS

- プロバイダー: LambdaInvoke
- バージョン: 1

設定パラメータ

FunctionName

必須: はい

Lambda 関数の名前

UserParameters

必須: いいえ

これらは、キーと値のペア形式で関数の入力として提供されるパラメータです。

ルール設定の例

YAML

```
- name: MyLambdaRule
  ruleTypeId:
    category: Rule
    owner: AWS
    provider: LambdaInvoke
    version: '1'
  configuration:
    FunctionName: my-function
  inputArtifacts:
  - name: SourceArtifact
  region: us-east-1
```

JSON

```
[
    {
        "name": "MyLambdaRule",
        "ruleTypeId": {
            "category": "Rule",
```

```
            "owner": "AWS",
            "provider": "LambdaInvoke",
            "version": "1"
        },
        "configuration": {
            "FunctionName": "my-function"
        },
        "inputArtifacts": [
            {
                "name": "SourceArtifact"
            }
        ],
        "region": "us-east-1"
    }
]
```

関連情報

このルールを利用する際に以下の関連リソースが役立ちます。

- 失敗時の条件の作成 – このセクションでは、アラームルールを使用して失敗時の条件を作成する手順を示しています。

VariableCheck

条件を作成するときに、VariableCheck ルールを追加できます。このセクションでは、ルールパラメータのリファレンスを提供します。ルールと条件の詳細については、「ステージ条件はどのように機能しますか?」を参照してください。

VariableCheck ルールを使用して、出力変数を、指定した式と照合する条件を作成できます。変数値がルール基準を満たすと (変数値が指定した出力変数以上であるなど)、ルールはチェックに合格します。

トピック

- ルールタイプ
- 設定パラメータ
- ルール設定の例
- 関連情報

ルールタイプ

- カテゴリ: Rule
- 所有者: AWS
- プロバイダー: VariableCheck
- バージョン: 1

設定パラメータ

オペレーター

必須: はい

変数チェックでどのオペレーションを実行するかを示す演算子。

次の例では、リポジトリ名の出力変数が MyDemoRepo と等しいかどうかをチェックします。

```
        "configuration": {
            "Variable": "#{SourceVariables.RepositoryName}",
            "Value": "MyDemoRepo",
            "Operator": "EQ"
        },
```

次の演算子は、以下に示すように式を作成するために使用できます。

- 等しい - 変数が文字列値と等しいかどうかをチェックするには、この演算子を選択します。

  CLI パラメータ: EQ
- 含む - 変数に文字列値が部分文字列として含まれているかどうかを確認するには、この演算子を選択します。

  CLI パラメータ: CONTAINS
- 一致 - 変数が特定の正規表現式と (文字列値として) 一致するかどうかをチェックするには、この演算子を選択します。

  CloudFormation のすべての正規表現は、Java の正規表現構文に準拠しています。Java の正規表現構文とそのコンストラクトの包括的な説明については、「java.util.regex.Pattern」を参照してください。

CLI パラメータ: MATCHES

- 等しくない - 変数が文字列値と等しくないかどうかを確認するには、この演算子を選択します。

  CLI パラメータ: NE

変数

必須: はい

チェックするパイプライン変数。

値

必須: はい

チェックする式の値。

次の例では、リポジトリ名の出力変数が MyDemoRepo と等しいかどうかをチェックします。

```
        "configuration": {
            "Variable": "#{SourceVariables.RepositoryName}",
            "Value": "MyDemoRepo",
            "Operator": "EQ"
        },
```

次の JSON の例では、2 つの個別のルールを定義しています。1 つは #{SourceVariables.RepositoryName} としてフォーマットされたリポジトリ名とブランチ名をチェックする EQ (等しい) ステートメント用で、もう 1 つは #{SourceVariables.CommitMessage} としてフォーマットされたコミットメッセージ出力変数を、指定した値「update」と照合する CONTAINS 用です。

```
  "beforeEntry": {
                    "conditions": [
                        {
                            "result": "FAIL",
                            "rules": [
                                {
                                    "name": "MyVarCheckRule",
                                    "ruleTypeId": {
                                        "category": "Rule",
                                        "owner": "AWS",
```

```
                                    "provider": "VariableCheck",
                                    "version": "1"
                                },
                                "configuration": {
                                    "Operator": "EQ",
                                    "Value": "MyDemoRepo",
                                    "Variable": "#{SourceVariables.RepositoryName}"
                                },
                                "inputArtifacts": [],
                                "region": "us-east-1"
                            },
                            {
                                "name": "MyVarCheckRuleContains",
                                "ruleTypeId": {
                                    "category": "Rule",
                                    "owner": "AWS",
                                    "provider": "VariableCheck",
                                    "version": "1"
                                },
                                "configuration": {
                                    "Operator": "CONTAINS",
                                    "Value": "update",
                                    "Variable": "#{SourceVariables.CommitMessage}"
                                },
                                "inputArtifacts": [],
                                "region": "us-east-1"
                            }
                        ]
                    }
                ]
            }
        }
    ],
```

ルール設定の例

YAML

```
- name: MyVariableCheck
  ruleTypeId:
    category: Rule
    owner: AWS
    provider: VariableCheck
```

```
    version: '1'
  configuration:
    Variable: "#{SourceVariables.RepositoryName}"
    Value: MyDemoRepo
    Operator: EQ
  inputArtifacts: []
  region: us-west-2
```

JSON

```
"rules": [
    {
        "name": "MyVariableCheck",
        "ruleTypeId": {
            "category": "Rule",
            "owner": "AWS",
            "provider": "VariableCheck",
            "version": "1"
        },
        "configuration": {
            "Variable": "#{SourceVariables.RepositoryName}",
            "Value": "MyDemoRepo",
            "Operator": "EQ"
        },
        "inputArtifacts": [],
        "region": "us-west-2"
    }
]
```

関連情報

このルールを利用する際に以下の関連リソースが役立ちます。

- チュートリアル: パイプラインの変数チェックルールを入力条件として作成する – このセクションでは、変数チェックルールを使用してエントリ時の条件を作成する手順を示すチュートリアルを提供します。
- 変数リファレンス – このセクションでは、パイプライン変数のリファレンス情報と例を示します。
- ルールと条件の詳細については、「CodePipeline API ガイド」の「条件」、「RuleTypeId」、「RuleExecution」を参照してください。

統合モデルのリファレンス

パイプラインのリリースプロセスに、既存の顧客ツールを構築するのに役立つサードパーティーサービスのための事前構築済みの統合がいくつかあります。パートナーまたはサードパーティーのサービスプロバイダーは、統合モデルを使用して CodePipeline で使用するアクションタイプを実装します。

このリファレンスは、CodePipeline でサポートされている統合モデルで管理されるアクションタイプを計画または操作するときに使用します。

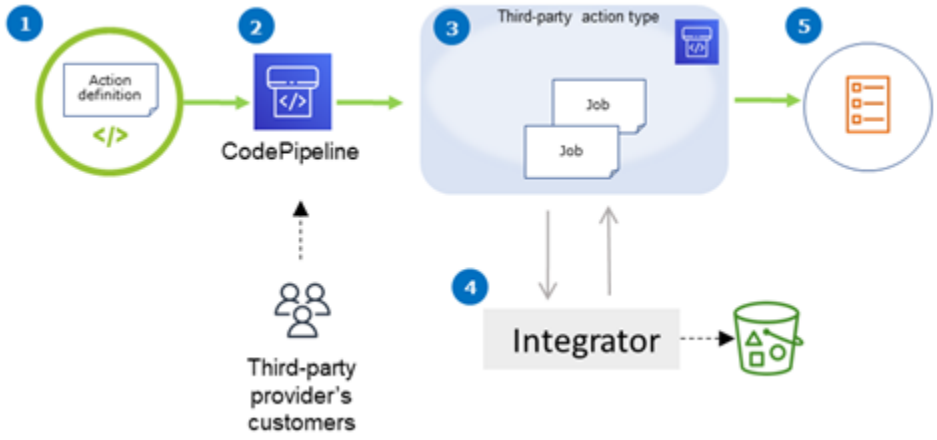
CodePipeline とのパートナー統合としてサードパーティーアクションタイプを認証するには、 AWS パートナーネットワーク (APN) を参照してください。この情報は、AWS CLI リファレンス を補足するものです。

トピック

- インテグレータとサードパーティーのアクションタイプがどのように機能するか
- 概念
- サポートされている統合モデル
- Lambda 統合モデル
- ジョブワーカー統合モデル

インテグレータとサードパーティーのアクションタイプがどのように機能するか

カスタマーパイプラインにサードパーティーのアクションタイプを追加して、顧客リソースのタスクを完了できます。インテグレータはジョブリクエストを管理し、CodePipeline でアクションを実行します。次の図表は、顧客がパイプラインで使用するために作成されたサードパーティーのアクションタイプを示しています。顧客がアクションを設定すると、アクションが実行され、インテグレータのアクションエンジンによって処理されるジョブリクエストが作成されます。



図表に示す内容は以下のステップです。

1. アクション定義が登録され、CodePipeline で使用できるようになります。サードパーティーのアクションは、サードパーティープロバイダのカスタマーに対して実行できます。
2. プロバイダーの顧客は CodePipeline でアクションを選択し、設定します。
3. アクションが実行され、ジョブが CodePipeline にキューに入れられます。CodePipeline でジョブの準備が整ったら、ジョブリクエストを送信します。
4. インテグレーター (サードパーティーポーリング API または Lambda 関数のジョブワーカー) は、ジョブリクエストを取得し、確認を返し、アクションのアーティファクトを処理します。
5. インテグレーターは、ジョブ結果と継続トークンを使用して、成功/失敗の出力 (ジョブワーカーは成功/失敗 API を使用するか、Lambda 関数が成功/失敗の出力を送信します) を返します。

アクションタイプのリクエスト、表示、および更新に使用できる手順については、 アクションタイプの使用 を参照してください。

概念

このセクションでは、サードパーティーのアクションタイプについて以下の用語を使用します。

アクションタイプ

同じ継続的デリバリーのワークロードを実行するパイプラインで再利用できる反復可能なプロセス。アクションタイプは、`Owner`、`Category`、`Provider`、および `Version` 。例:

```
{
```

```
            "Category": "Deploy",
            "Owner": "AWS",
            "Provider": "CodeDeploy",
            "Version": "1"
        },
```

同じタイプのアクションはすべて、同じ実装を共有します。

アクション

アクションタイプの単一のインスタンス。パイプラインのステージ内で発生する個別のプロセスの1つです。これには、通常、このアクションが実行されるパイプラインに固有のユーザー値が含まれます。

アクションの定義

アクションおよび入出力アーティファクトの構成に必要なプロパティを定義するアクションタイプのスキーマ。

アクションの実行

顧客のパイプラインでのアクションが成功したかどうかを判断するために実行されたジョブの集まり。

アクション実行エンジン

アクションタイプで使用される統合タイプを定義するアクション実行設定のプロパティ。有効な値は、`JobWorker` および `Lambda` です。

統合

アクションタイプを実装するためにインテグレータによって実行されるソフトウェアについて説明します。CodePipeline は、サポートされている 2 つのアクションエンジン `JobWorker` そして `Lambda` に対応する 2 つの統合タイプをサポートしています。

インテグレータ

アクションタイプの実装を所有している人。

ジョブ

パイプラインと顧客コンテキストを使用して統合を実行するための作業です。アクションの実行は、1 つ以上のジョブで構成されます。

ジョブワーカー

顧客入力を処理してジョブを実行するサービス。

サポートされている統合モデル

CodePipeline には 2 つの統合モデルがあります

- Lambda 統合モデル:CodePipeline でアクションタイプを操作するには、この統合モデルが推奨されます。Lambda 統合モデルは、アクションの実行時に Lambda 関数を使用してジョブリクエストを処理します。
- ジョブワーカー統合モデル: ジョブワーカー統合モデルは、サードパーティー統合で以前に使用されたモデルです。ジョブワーカー統合モデルは、 CodePipeline API にアクセスするように設定されたジョブワーカーを使用して、アクションの実行時にジョブリクエストを処理します。

比較のために、次の表に 2 つのモデルの特徴を示します。

	Lambda 統合モデル	ジョブワーカー統合モデル
説明	インテグレータは、インテグレーションを Lambda 関数として書き込み、アクションで使用可能なジョブがあるたびに CodePipeline によって呼び出されます。Lambda 関数は、使用可能なジョブをポーリングするのではなく、次のジョブ リクエストが受信されるまで待機します。	インテグレータは、インテグレーションをジョブワーカーとして書き込み、顧客のパイプラインで利用可能なジョブを常にポーリングします。その後ジョブワーカーはジョブを実行し、CodePipeline API を使用してジョブ結果を CodePipeline に送り返します。
インフラストラクチャ	AWS Lambda	Amazon EC2 インスタンスなどのインテグレータのインフラストラクチャにジョブワーカーコードをデプロイします。
開発作業	統合にはビジネスロジックのみが含まれます。	統合は、ビジネスロジックを含めるだけでなく、CodePipeline API と対話する必要があります。

	Lambda 統合モデル	ジョブワーカー統合モデル
Opsの努力	インフラストラクチャは AWS リソースにすぎないため、運用の労力が少なくなります。	ジョブワーカーはスタンドアロンのハードウェアを必要とするため、オペレーションの労力が高くなります。
最大ジョブのランタイム	統合を 15 分以上アクティブに実行する必要がある場合は、このモデルを使用できません。このアクションは、プロセスを開始する (顧客のコードアーティファクトの構築を開始するなど)、終了時に結果を返す必要があるインテグレータを対象としています。インテグレータがビルドの終了を待ち続けることはお勧めしません。代わりに、継続 を返す。CodePipeline は、インテグレータのコードから継続を受け取り、終了するまでジョブをチェックすると、さらに 30 秒以内に新しいジョブを作成します。	このモデルを使用すると、非常に長時間実行されるジョブ (時間/日) を維持できます。

Lambda 統合モデル

サポートされる Lambda 統合モデルには、Lambda 関数の作成と、サードパーティーアクションタイプの出力の定義が含まれます。

Lambda 関数を更新して CodePipeline からの入力を処理します。

新しい Lambda 関数の作成 アクションタイプのパイプラインで利用可能なジョブがあるときに実行される Lambda 関数にビジネスロジックを追加できます。たとえば、顧客とパイプラインのコンテキストを考えると、顧客のためのサービスでビルドを開始したい場合があります。

以下のパラメータを使用して Lambda 関数を更新し、 CodePipeline からの入力を処理します。

形式:

- jobId:
  - システムによって生成されたジョブの固有の ID 。
  - タイプ: 文字列
  - パターン:[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}
- accountId:
  - ジョブの実行時に使用する顧客の AWS アカウントの ID。
  - タイプ: 文字列
  - Pattern: [0-9]\{12\}
- data:
  - 統合がジョブの完了に使用するジョブに関するその他の情報。
  - 次のいずれかのマップが含まれます。
    - actionConfiguration:
      - アクションの設定データ。アクション設定フィールドは、顧客が値を入力するためのキー値のペアのマッピングです。キーは、アクションを設定するときに、アクションタイプ定義ファイル内のキーパラメータによって決定されます。この例では、Username そして Password フィールドに情報を指定するアクションのユーザーよって値が決定される。
      - タイプ : 文字列から文字列へのマッピング、オプションで提供

        例:

```
"configuration": {
    "Username": "MyUser",
    "Password": "MyPassword"
},
```

    - encryptionKey:
      - キーなど、アーティファクトストア内のデータの暗号化に使用される AWS KMS キーに関する情報を表します。
      - 内容 : データ型タイプ encryptionKey 、オプションで提供
    - inputArtifacts:
      - 作業対象のアーティファクト (テスト、構築アーティファクトなど) に関する情報のリスト。
      - 次: データ型のリスト Artifact、オプションで提供

- outputArtifacts:
  - アクションの出力に関する情報のリスト。
  - 目次:データ型のリスト Artifact、オプションで提供
- actionCredentials:
  - AWS セッション認証情報オブジェクトを表します。これらの認証情報は、 AWS STSによって発行される一時的な認証情報です。CodePipeline にパイプラインのアーティファクトを格納するために使用される S3 バケットの入出力アーティファクトにアクセスするために使用できます。

    これらの認証情報には、アクションタイプ定義ファイル内の指定されたポリシーステートメントテンプレートと同じ権限もあります。
  - 内容 : データ型タイプ AWSSessionCredentials 、オプションで提供
- actionExecutionId:
  - アクションの実行の外部 ID。
  - タイプ: 文字列
- continuationToken:
  - ジョブを非同期的に続行するためにジョブに必要な、デプロイメント ID などのシステム生成トークン。
  - タイプ : 文字列、オプションで提供

データ型:

- encryptionKey:
  - id:
    - キーを識別する際に使用された ID。 AWS KMS キーには、キー ID、キー ARN、またはエイリアス ARN を使用できます。
    - タイプ: 文字列
  - type:
    - キーなどの暗号化 AWS KMS キーのタイプ。
    - 型: 文字列
    - 有効な値: KMS
- Artifact:
  - name:

    - アーティファクトの名前。
    - タイプ : 文字列、オプションで提供
  - revision:
    - アーティファクトのリビジョン ID 。オブジェクトのタイプに応じて、コミット ID (GitHub) またはリビジョン ID （ Amazon S3 ）になります。
    - タイプ : 文字列、オプションで提供
  - location:
    - アーティファクトの位置。
    - 内容 : データ型タイプ ArtifactLocation 、オプションで提供
- ArtifactLocation:
  - type:
    - ロケーション内のアーティファクトのタイプ。
    - タイプ : 文字列、オプションで提供
    - 有効な値: S3
  - s3Location:
    - リビジョンを含む S3 バケットの場所。
    - 内容 : データ型タイプ S3Location 、オプションで提供
- S3Location:
  - bucketName:
    - S3 バケットの名前。
    - タイプ: 文字列
  - objectKey:
    - S3 バケット内のオブジェクトのキー。バケット内のオブジェクトを一意に識別します。
    - タイプ: 文字列
- AWSSessionCredentials:
  - accessKeyId:
    - セッションのアクセスキー。
    - タイプ: 文字列
  - secretAccessKey:
    - セッションのシークレットアクセスキー。

- タイプ: 文字列

- sessionToken:
  - セッションのトークン。
  - タイプ: 文字列

例:

```
{
    "jobId": "01234567-abcd-abcd-abcd-012345678910",
    "accountId": "012345678910",
    "data": {
        "actionConfiguration": {
            "key1": "value1",
            "key2": "value2"
        },
        "encryptionKey": {
            "id": "123-abc",
            "type": "KMS"
        },
        "inputArtifacts": [
            {
                "name": "input-art-name",
                "location": {
                    "type": "S3",
                    "s3Location": {
                        "bucketName": "inputBucket",
                        "objectKey": "inputKey"
                    }
                }
            }
        ],
        "outputArtifacts": [
            {
                "name": "output-art-name",
                "location": {
                    "type": "S3",
                    "s3Location": {
                        "bucketName": "outputBucket",
                        "objectKey": "outputKey"
                    }
                }
```

```
            }
        ],
        "actionExecutionId": "actionExecutionId",
        "actionCredentials": {
            "accessKeyId": "access-id",
            "secretAccessKey": "secret-id",
            "sessionToken": "session-id"
        },
        "continuationToken": "continueId-xxyyzz"
    }
}
```

Lambda 関数の結果を CodePipeline に返す

インテグレータのジョブワーカー リソースは、成功、失敗、または継続の場合に有効なペイロードを返す必要があります。

形式:

- `result` : ジョブの結果。
  - 必須
  - 有効な値 (大文字と小文字は区別されません)
    - `Success` : ジョブが正常に終了したことを示します。
    - `Continue` ：ジョブが正常に実行され、ジョブワーカーが同じアクションの実行に対して再呼び出された場合など、続行する必要があることを示します。
    - `Fail` : ジョブが失敗し、ターミナルであることを示します。
- `failureType` : 失敗したジョブに関連付ける障害タイプ。

  パートナーアクションの `failureType` カテゴリは、ジョブの実行中に発生した障害のタイプを示します。インテグレータは、ジョブの障害結果を CodePipeline に戻すときに、障害メッセージとともにタイプを設定します。
  - オプション。結果が `Fail` の場合に必須。
  - `result` が `Success` または `Continue` の場合は、null にする必要があります。
  - 有効な値:
    - 構成エラー
    - ジョブ失敗
    - パーミッションエラー

    - リビジョンOutOfSync
    - リビジョン利用不可
    - システム利用不可
- continuation : 現在のアクション実行内で次のジョブに渡される継続状態。
    - オプション。結果が Continue の場合に必須。
    - result が Success または Fail の場合は、 null にする必要があります。
    - プロパティ:
        - State : 渡すステートのハッシュ。
- status : アクション実行のステータス。
    - オプション。
    - プロパティ:
        - ExternalExecutionId : ジョブに関連付けるオプションの外部実行 ID またはコミット ID。
        - Summary : 発生した事象の要約。障害シナリオでは、これがユーザーに表示される障害メッセージになります。
- outputVariables : 次のアクションの実行に渡されるキーと値のペアのセット。
    - オプション。
    - result が Continue または Fail の場合は、 null にする必要があります。

例:

```
{
    "result": "success",
    "failureType": null,
    "continuation": null,
    "status": {
        "externalExecutionId": "my-commit-id-123",
        "summary": "everything is dandy"
    },
    "outputVariables": {
        "FirstOne": "Nice",
        "SecondOne": "Nicest",
        ...
    }
}
```

継続トークンを使用して、非同期プロセスの結果を待つ

continuation トークンは Lambda 関数のペイロードと結果の一部です。これは、ジョブの状態を CodePipeline に渡し、ジョブを継続する必要があることを示す方法です。たとえば、インテグレーターがリソースでお客様のビルドを開始した後、ビルドが完了するまで待機しませんが、 を result として返し、ビルドの一意の ID をcontinuationトークンとして CodePipeline にcontinue返すことで、ターミナル結果がないことを CodePipeline に示します。

> **Note**
>
> Lambda 関数は最大 15 分まで実行できます。ジョブを長く実行する必要がある場合は、継続トークンを使用できます。

CodePipeline チームは 30 秒後にインテグレータを呼び出します。ペイロードに continuation トークンを入れて、完了をチェックできるようにします。構築が完了すると、インテグレータはターミナルの成功/失敗結果を返し、そうでない場合は続行します。

ランタイム時にインテグレーターの Lambda 関数を呼び出す権限を CodePipeline に提供します。

インテグレータの Lambda 関数に、CodePipeline サービスプリンシパルを使用して呼び出す権限を追加して、 CodePipeline サービスに提供します。: codepipeline.amazonaws.com アクセス許可を追加するには、 CloudFormation または コマンドラインを使用します。例については、アクションタイプの使用を参照してください。

ジョブワーカー統合モデル

高レベル ワークフローを綿密に設計した後、ジョブワーカーを作成できます。最終的にはサードパーティーアクションの仕様がジョブワーカーに必要なものを決定しますが、サードパーティーアクションのジョブワーカーの多くは以下の機能を含みます:

- PollForThirdPartyJobs を使用して CodePipeline からジョブをポーリングする。
- ジョブを確認し、 CodePipeline に AcknowledgeThirdPartyJob、PutThirdPartyJobSuccessResult および PutThirdPartyJobFailureResultを使用して結果を返す。
- パイプラインの Amazon S3 バケットからアーティファクトを取得する、またはアーティファクトを配置する。Amazon S3 バケットからアーティファクトをダウンロードするには、署名バージョ

ン 4 の署名 (Sig V4) を使用する Amazon S3 クライアントを作成する必要があります。Sig V4 は必須です AWS KMS。

Amazon S3 バケットにアーティファクトをアップロードするには、 AWS Key Management Service (AWS KMS). AWS KMS uses で暗号化を使用するように Amazon S3 `PutObject`リクエストも設定する必要があります AWS KMS keys。 AWS マネージドキー またはカスタマーマネージドキーを使用してアーティファクトをアップロードするかどうかを知るには、ジョブワーカーがジョブデータを調べ、暗号化キープロパティを確認する必要があります。プロパティが設定されている場合は、設定時にそのカスタマーマネージドキー ID を使用する必要があります AWS KMS。key プロパティが null の場合は、 AWS マネージドキーを使用します。CodePipeline は、特に設定 AWS マネージドキー されていない限り、 を使用します。

Java または .NET で AWS KMS パラメータを作成する方法を示す例については、「 SDK を使用した Amazon S3 AWS Key Management Service での の指定 AWS SDKs」を参照してください。CodePipeline 用の Amazon S3 バケットの詳細については、「CodePipeline の概念 」を参照してください。

ジョブワーカー用にアクセス許可管理戦略を選択して設定する

CodePipeline でサードパーティーアクションのジョブワーカーを開発するには、ユーザーやアクセス権限管理を統合するための戦略が必要になります。

最も簡単な戦略は、 AWS Identity and Access Management (IAM) インスタンスロールを使用して Amazon EC2 インスタンスを作成して、ジョブワーカーに必要なインフラストラクチャを追加することです。これにより、統合に必要なリソースを簡単にスケールアップできます。との組み込み統合を使用して AWS 、ジョブワーカーと CodePipeline 間のやり取りを簡素化できます。

Amazon EC2 の詳細を参照し、統合に適しているかどうかを判断します。詳細については、「Amazon EC2 - 仮想サーバーのホスティング」を参照してください。Amazon EC2 インスタンスのセットアップについては、「Amazon EC2 Linux インスタンスの使用開始」を参照してください。

他に考慮すべき戦略は、IAMと ID フェデレーションを使用した既存の ID プロバイダーシステムおよびリソースとの統合です。この戦略は、お客様がすでに企業 ID プロバイダーを持っているか、ウェブ ID プロバイダーを使用するユーザーをサポートできるよう設定されている場合に、特に便利です。ID フェデレーションを使用すると、IAM ユーザーを作成または管理することなく、CodePipeline などの AWS リソースへの安全なアクセスを許可できます。パスワードのセキュリティ要件や認証情報の更新に機能やポリシーを活用できます。サンプルアプリケーションをお客様

自身の設計のテンプレートとして使用できます。詳細については、「フェデレーションの管理」を参照してください。

アクセス許可を付与するにはユーザー、グループ、またはロールにアクセス許可を追加します。

- のユーザーとグループ AWS IAM アイデンティティセンター:

  アクセス許可セットを作成します。「AWS IAM アイデンティティセンター ユーザーガイド」の「アクセス許可セットを作成する」の手順に従ってください。

- IAM 内で、ID プロバイダーによって管理されているユーザー:

  ID フェデレーションのロールを作成します。詳細については IAM ユーザーガイド の サードパーティー ID プロバイダー (フェデレーション) 用のロールを作成する を参照してください。

- IAM ユーザー:
  - ユーザーが担当できるロールを作成します。手順については IAM ユーザーガイド の IAM ユーザーのロールの作成 を参照してください。
  - (お奨めできない方法) ポリシーをユーザーに直接アタッチするか、ユーザーをユーザーグループに追加します。「IAM ユーザーガイド」の「ユーザー (コンソール) へのアクセス許可の追加」の指示に従います。

次は、サードパーティー ジョブワーカーで使用するために作成する可能性があるポリシーの例です。このポリシーは例に過ぎず、そのまま提供されています。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codepipeline:PollForThirdPartyJobs",
        "codepipeline:AcknowledgeThirdPartyJob",
        "codepipeline:GetThirdPartyJobDetails",
        "codepipeline:PutThirdPartyJobSuccessResult",
        "codepipeline:PutThirdPartyJobFailureResult"
      ],
      "Resource": [
```

```
        "arn:aws:codepipeline:us-east-2::actionType:ThirdParty/Build/Provider/1/"

      ]
    }
  ]
}
```

イメージ定義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。コンテナのソースまたはデプロイアクションを使用してパイプラインを作成する方法については、「パイプライン、ステージ、アクションを作成する」を参照してください。

AWS CodePipeline Amazon ECR ソースアクションや Amazon ECS デプロイアクションなどのコンテナアクションのジョブワーカーは、定義ファイルを使用してイメージ URI とコンテナ名をタスク定義にマッピングします。各定義ファイルは、次のようにアクションプロバイダによって使用される JSON 形式のファイルです。

- Amazon ECS 標準デプロイでは、デプロイアクションへの入力で `imagedefinitions.json` ファイルが必要です。CodePipeline で Amazon ECS スタンダードデプロイアクションを使用するチュートリアルについては、チュートリアル: CodePipeline を使用した Amazon ECS 標準デプロイ を参照してください。CodePipeline で Amazon ECS 標準デプロイアクションと ECRBuildAndPublish アクションの仕様に関する追加のチュートリアルについては、「チュートリアル: CodePipeline (V2 タイプ) を使用して Docker イメージを構築し Amazon ECR にプッシュする」を参照してください。
- Amazon ECS Blue/Green デプロイでは、デプロイアクションへの入力で `imageDetail.json` ファイルが必要です。ブルー/グリーンデプロイのサンプルを使用したチュートリアルについては、「チュートリアル: Amazon ECR ソース、ECS - CodeDeploy 間のデプロイでパイプラインを作成する」を参照してください。
  - Amazon ECR ソースアクションでは、ソースアクションの出力として提供される、`imageDetail.json` ファイルが生成されます。

トピック

- Amazon ECS 標準デプロイアクション用の imagedefinitions.json ファイル
- Amazon ECS Blue/Green デプロイアクション用の imageDetail.json ファイル

Amazon ECS 標準デプロイアクション用の imagedefinitions.json ファイル

イメージ定義ドキュメントは、Amazon ECS のコンテナ名およびイメージとタグについて説明する JSON ファイルです。コンテナベースのアプリケーションをデプロイする場合は、イメージ定義ファ

イルを生成して CodePipeline のジョブワーカーに Amazon ECS コンテナを提供し、Amazon ECR などのイメージリポジトリから取得するイメージ ID を生成する必要があります。

> **Note**
>
> このファイルのデフォルトのファイル名は `imagedefinitions.json` です。別のファイル名を使用することを選択した場合は、パイプラインデプロイステージを作成するときにそのファイル名を指定する必要があります。

以下の考慮事項に注意して、`imagedefinitions.json` ファイルを作成します。

- ファイルには UTF-8 エンコーディングを使用する必要があります。
- イメージ定義ファイルの最大ファイルサイズの制限は 100 KB です。
- ファイルは、ソースとして作成するか、デプロイアクションの入力アーティファクトになるようにアーティファクトを構築する必要があります。つまり、ファイルが CodeCommit リポジトリなどのソースの場所にアップロードされていること、またはビルドされた出力アーティファクトとして生成されていることを確認してください。

`imagedefinitions.json` ファイルはコンテナ名とイメージ URI を提供します。次のキーと値のペアのセットで構築する必要があります。

キー	値
name	*#####*
ImageURI	*imageUri*

> **Note**
>
> [名前] フィールドは、コンテナイメージ名、つまり Docker イメージ名に使用します。

コンテナ名が `sample-app` で、イメージ URI が `ecs-repo`、タグが `latest` の JSON 構造は次のとおりです。

```
[
```

```
  {
    "name": "sample-app",
    "imageUri": "11111EXAMPLE.dkr.ecr.us-west-2.amazonaws.com/ecs-repo:latest"
  }
]
```

複数のコンテナイメージのペアをリストするようにファイルを構築することもできます。

JSON の構造:

```
[
  {
    "name": "simple-app",
    "imageUri": "httpd:2.4"
  },
  {
    "name": "simple-app-1",
    "imageUri": "mysql"
  },
  {
    "name": "simple-app-2",
    "imageUri": "java1.8"
  }
]
```

パイプラインを作成する前に、以下の手順に従って imagedefinitions.json ファイルを設定します。

1. パイプラインのコンテナベースのアプリケーションデプロイの計画の一環として、ソースステージとビルドステージを計画します (該当する場合)。
2. 次のいずれかを選択します。

    a. パイプラインがビルドステージをスキップするように作成されている場合、ソースアクションがアーティファクトを提供できるように、手動で JSON ファイルを作成してソースリポジトリにアップロードする必要があります。テキストエディタを使用してファイルを作成し、ファイルに名前を付けます。または、デフォルトの imagedefinitions.json ファイル名を使用します。イメージ定義ファイルをソースリポジトリにプッシュします。

> **Note**
>
> ソースリポジトリが Amazon S3 バケットの場合は、JSON ファイルを圧縮してください。

b. パイプラインにビルドステージがある場合は、ビルドフェーズ中にソースリポジトリにイメージ定義ファイルを出力するコマンドを buildspec ファイルに追加します。以下の例では、printf コマンドを使用して、`imagedefinitions.json` ファイルを作成します。`buildspec.yml` ファイルの `post_build` セクションにこのコマンドをリストします。

```
printf '[{"name":"container_name","imageUri":"image_URI"}]' >
imagedefinitions.json
```

イメージ定義ファイルを出力アーティファクトとして `buildspec.yml` ファイルに含める必要があります。

3. コンソールでパイプラインを作成する場合、[パイプラインの作成] ウィザードの [デプロイ] ページの [イメージのファイル名] にイメージ定義ファイル名を入力します。

Amazon ECS をデプロイプロバイダーとして使用するパイプラインを作成するための段階的なチュートリアルについては、[チュートリアル: CodePipeline を使用した継続的デプロイメント] を参照してください。

Amazon ECS Blue/Green デプロイアクション用の imageDetail.json ファイル

`imageDetail.json` ドキュメントは、 Amazon ECS イメージ URI を説明する JSON ファイルです。ブルー/グリーンデプロイ用にコンテナベースのアプリケーションをデプロイする場合、`imageDetail.json` ファイルを生成して、Amazon ECS と CodeDeploy ジョブワーカーに、Amazon ECR などのイメージリポジトリから取得するイメージ ID を提供する必要があります。

> **Note**
>
> ファイルの名前は `imageDetail.json` である必要があります。

アクションとそのパラメータの説明については、「Amazon ECS および CodeDeploy ブルー/グリーンデプロイアクションリファレンス」を参照してください。

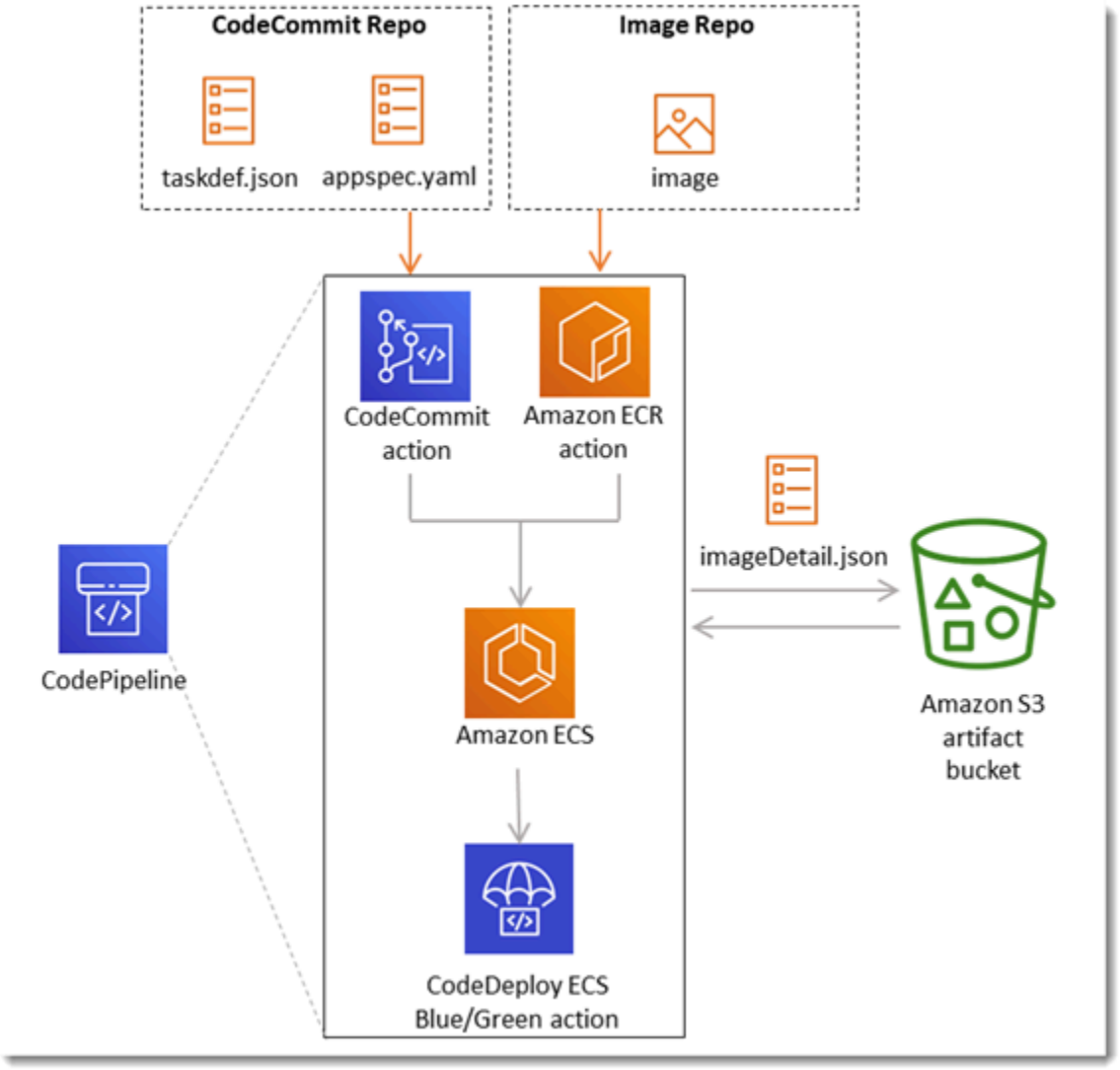
imageDetail.json ファイルは、ソースとして作成するか、デプロイアクションの入力アーティファクトになるようにアーティファクトを構築する必要があります。これらの方法のいずれかを使用して、パイプラインに imageDetail.json ファイルを提供できます。

- ソースの場所に imageDetail.json ファイルを含め、Amazon ECS Blue/Green デプロイアクションへの入力としてパイプラインで提供されるようにします。

> **Note**
>
> ソースリポジトリが Amazon S3 バケットの場合は、JSON ファイルを圧縮してください。

- Amazon ECR ソースアクションでは、次のアクションへの入力アーティファクトとして imageDetail.json ファイルが自動生成されます。

> **Note**
>
> Amazon ECR ソースアクションがこのファイルを作成するので、Amazon ECR ソースアクションを含むパイプラインは 手動で imageDetail.json ファイルを提供する必要はありません。
> Amazon ECR ソースステージを含むパイプラインの作成に関するチュートリアルについては、チュートリアル: Amazon ECR ソース、ECS - CodeDeploy 間のデプロイでパイプラインを作成する を参照してください。



imageDetail.json ファイルは、イメージ URI を提供します。次のキーと値のペアで構築する必要があります。

キー	値
ImageURI	*image_URI*

imageDetail.json

以下に、イメージ URI が ACCOUNTID.dkr.ecr.us-west-2.amazonaws.com/dk-image-repo@sha256:example3 である JSON 構造を示します。

```
{
```

```
"ImageURI": "ACCOUNTID.dkr.ecr.us-west-2.amazonaws.com/dk-image-
repo@sha256:example3"
}
```

imageDetail.json (generated by ECR)

変更がイメージリポジトリにプッシュされるたびに、Amazon ECR ソースアクションによって imageDetail.json ファイルが自動生成されます。Amazon ECR ソースアクションにより生成された imageDetail.json は、ソースアクションから出力アーティファクトとしてパイプラインの次のアクションに提供されます。

リポジトリ名が dk-image-repo で、イメージ URI が ecs-repo、イメージタグが latest の JSON 構造は次のとおりです。

```
{
    "ImageSizeInBytes": "44728918",
    "ImageDigest":
 "sha256:EXAMPLE1122334455667788990bfea42ea2d3b8a1ee8329ba7e68694950afd3",
    "Version": "1.0",
    "ImagePushedAt": "Mon Jan 21 20:04:00 UTC 2019",
    "RegistryId": "EXAMPLE12233",
    "RepositoryName": "dk-image-repo",
    "ImageURI": "ACCOUNTID.dkr.ecr.us-west-2.amazonaws.com/dk-image-
repo@sha256:example3",
    "ImageTags": [
        "latest"
    ]
}
```

imageDetail.json ファイルは、次のようにイメージ URI とコンテナ名を Amazon ECS タスク定義にマッピングします。

- ImageSizeInBytes: リポジトリ内のイメージのサイズ (単位: バイト)。
- ImageDigest: イメージマニフェストの sha256 ダイジェスト。
- Version: イメージバージョン。
- ImagePushedAt: 最新のイメージがリポジトリにプッシュされた日時。
- RegistryId: リポジトリを含むレジストリに関連付けられた AWS アカウント ID。
- RepositoryName: イメージがプッシュされた Amazon ECR リポジトリの名前。
- ImageURI: イメージの URI。

- ImageTags: イメージに使用されるタグ。

パイプラインを作成する前に、以下の手順に従って imageDetail.json ファイルを設定します。

1. パイプラインのコンテナベースのアプリケーション Blue/Green デプロイの計画の一環として、ソースステージとビルドステージを計画します (該当する場合)。
2. 次のいずれかを選択します。

   a. パイプラインでビルドステージをスキップしている場合、JSON ファイルを手動で作成し、CodeCommit などのソースリポジトリにアップロードして、ソースアクションがアーティファクトを提供できるようにします。テキストエディタを使用してファイルを作成し、ファイルに名前を付けます。または、デフォルトの imageDetail.json ファイル名を使用します。imageDetail.json ファイルをソースリポジトリにプッシュします。

   b. パイプラインにビルドステージがある場合は、以下の手順を実行します。

      i. ビルドフェーズ中にソースリポジトリ内のイメージ定義ファイルを出力するコマンドを buildspec ファイルに追加します。以下の例では、printf コマンドを使用して、imageDetail.json ファイルを作成します。buildspec.yml ファイルの post_build セクションにこのコマンドをリストします。

```
printf '{"ImageURI":"image_URI"}' > imageDetail.json
```

      imageDetail.json ファイルを出力アーティファクトとして buildspec.yml ファイルに含める必要があります。

      ii. imageDetail.jsonをアーティファクトファイルとして buildspec.yml ファイルに追加します。

```
artifacts:
  files:
    - imageDetail.json
```

変数リファレンス

このセクションは参照のみを目的としています。変数の作成については、「変数の操作」を参照してください。

変数を使用すると、パイプラインの実行時またはアクションの実行時に決定される値でパイプラインアクションを設定できます。

一部のアクションプロバイダは、変数の定義されたセットを生成します。コミット ID など、そのアクションプロバイダのデフォルトの変数キーから選択します。

> **Important**
>
> シークレットパラメータを渡すときは、値を直接入力しないでください。値はプレーンテキストとしてレンダリングされるため、読み取り可能です。セキュリティ上の理由から、シークレットを含むプレーンテキストは使用しないでください。シークレットの保存 AWS Secrets Manager には を使用することを強くお勧めします。

変数を使用するためのステップバイステップの例を参照するには:

- パイプラインの実行時に渡されるパイプラインレベルの変数のチュートリアルについては、「チュートリアル: パイプラインレベルの変数を使用する」を参照してください。
- アップストリームアクション (CodeCommit) の変数を使用し、出力変数を生成する Lambda アクションのチュートリアルについては、チュートリアル: Lambda 呼び出しアクションで変数を使用する を参照してください。
- アップストリーム CloudFormation CloudFormation アクションのスタック出力変数を参照するアクションを含むチュートリアルについては、「」を参照してくださいチュートリアル: AWS CloudFormation デプロイアクションの変数を使用するパイプラインを作成する。
- CodeCommit コミット ID とコミットメッセージに解決される出力変数を参照するメッセージテキストを使用した手動承認アクションの例については、例: 手動承認で変数を使用する を参照してください。
- GitHub ブランチ名に解決される環境変数を持つ CodeBuild アクションの例については、例:CodeBuild 環境変数で BranchName 変数を使用する を参照してください。
- CodeBuild アクションは、ビルドの一部として エクスポートされたすべての環境変数を変数として生成します。詳細については、「CodeBuild アクションの出力変数」を参照してください。

変数の制限

制限の詳細については、「AWS CodePipeline のクォータ」を参照してください。

> **Note**
>
> アクション設定フィールドに変数構文を入力する場合は、設定フィールドの 1,000 文字制限を超えないようにしてください。この制限を超えると、検証エラーが返されます。

トピック

- 概念
- 変数のユースケース
- 変数の設定
- 変数の解決
- 変数のルール
- パイプラインアクションで使用できる変数

概念

このセクションでは、変数と名前空間に関連する主要な用語と概念を示します。

[変数]

変数は、パイプラインでアクションを動的に設定するために使用できるキーと値のペアです。現在、これらの変数を使用可能にする方法は 3 つあります。

- 各パイプライン実行の開始時に暗黙的に使用できる一連の変数があります。このセットには `PipelineExecutionId`、現在のパイプライン実行の ID が含まれています。
- パイプラインレベルの変数は、パイプラインの作成時に定義され、パイプラインの実行時に解決されます。

  パイプラインの作成時にパイプラインレベルの変数を指定し、パイプラインの実行時にその値を設定できます。
- 実行時に変数のセットを生成するアクションタイプがあります。アクションによって生成された変数は、`outputVariables`の一部である ListActionExecutions APIフィールドを確認できます。

アクションプロバイダごとに使用できるキー名のリストについては、「パイプラインアクションで使用できる変数」を参照してください。各アクションタイプによって生成される変数については、CodePipeline 「アクション構造リファレンス」を参照してください。

アクション設定でこれらの変数を参照するには、正しい名前空間で変数リファレンス構文を使用する必要があります。

変数ワークフローの例については、「変数の設定」を参照してください。

名前空間

変数を一意に参照できるようにするには、変数を名前空間に割り当てる必要があります。名前空間に変数のセットを割り当てた後、次の構文で名前空間と変数キーを使用して、アクション設定で変数を参照できます。

```
#{namespace.variable_key}
```

変数を割り当てることができる名前空間には、次の 3 種類があります。

- codepipeline 予約済み名前空間

  これは、各パイプライン実行の開始時に利用可能な暗黙的な変数のセットに割り当てられた名前空間です。この名前空間は codepipeline です。変数リファレンスの例:

  ```
  #{codepipeline.PipelineExecutionId}
  ```

- パイプラインレベルの変数の名前空間

  これは、パイプラインレベルの変数に割り当てられる名前空間です。パイプラインレベルのすべての変数の名前空間は variables です。変数リファレンスの例:

  ```
  #{variables.variable_name}
  ```

- アクションに割り当てられた名前空間

  これは、アクションに割り当てる名前空間です。アクションによって生成されるすべての変数は、この名前空間に属します。アクションによって生成された変数をダウンストリームアクション設定で使用できるようにするには、生成アクションを名前空間で設定する必要があります。名前空間はパイプライン定義全体で一意でなければならず、アーティファクト名と競合することはできませ

ん。次に、SourceVariables の名前空間で設定されたアクションの変数リファレスの例を示します。

```
#{SourceVariables.VersionId}
```

変数のユースケース

以下に、パイプラインレベルの変数の最も一般的な使用例をいくつか示します。これらは、特定のニーズに合わせて変数をどのように使用するかを決定するのに役立ちます。

- パイプラインレベルの変数は、アクション設定への入力にわずかな変更を加えて、毎回同じパイプラインを使用したいと考える CodePipeline ユーザー向けです。パイプラインを開始するどの開発者も、パイプラインの開始時は UI に変数値を追加します。この設定では、その実行にのみ使用するパラメータを渡します。
- パイプラインレベルの変数を使用すると、パイプライン内のアクションに動的な入力を渡すことができます。同じパイプラインの異なるバージョンを管理したり、複雑なパイプラインを作成したりすることなく、パラメータ化されたパイプラインを CodePipeline に移行できます。
- パイプラインレベルの変数を使用して入力パラメータを渡すことで、実行ごとにパイプラインを再利用できます。例えば、本番環境にデプロイするバージョンを指定する場合に、パイプラインを複製する必要がなくなります。
- 1 つのパイプラインを使用して、複数のビルド環境とデプロイ環境にリソースをデプロイできます。例えば、CodeCommit リポジトリを含むパイプラインでは、指定したブランチとターゲットデプロイ環境からのデプロイを、パイプラインレベルで渡される CodeBuild パラメータと CodeDeploy パラメータを使用して行うことができます。

> **Note**
>
> Amazon ECR、Amazon S3、または CodeCommit ソースの場合、入力変換エントリを使用してソースオーバーライドを作成し、パイプラインイベントの EventBridge で `revisionValue` を使用することもできます。ここで、`revisionValue` はオブジェクトキー、コミット、またはイメージ ID のソースイベント変数から派生します。詳細については、Amazon ECR ソースアクションと EventBridge リソース、イベントに対して有効にしたソースを使用して Amazon S3 ソースアクションに接続する、または CodeCommit ソースアクションと EventBridge の手順に含まれる入力変換エントリのオプション手順を参照してください。

変数の設定

パイプラインレベルまたはアクションレベルの変数は、パイプライン構造で設定できます。

パイプラインレベルの変数を設定する

パイプラインレベルで 1 つ以上の変数を追加できます。この値は CodePipeline アクションの設定で参照できます。パイプラインを作成するときに、変数名、デフォルト値、説明を追加できます。変数は実行時に解決されます。

> **Note**
>
> パイプラインレベルの変数にデフォルト値が定義されていない場合、その変数は必須とみなされます。パイプラインの開始時には、必須のすべての変数に対して上書きを指定する必要があります。指定しない場合、パイプラインの実行は検証エラーで失敗します。

パイプライン構造の変数属性を使用して、パイプラインレベルの変数を指定します。以下の例では、変数 `Variable1` の値は `Value1` です。

```
    "variables": [
         {
             "name": "Variable1",
             "defaultValue": "Value1",
             "description": "description"
         }
     ]
```

パイプライン JSON 構造の例については、「パイプライン、ステージ、アクションを作成する」を参照してください。

パイプラインの実行時に渡されるパイプラインレベルの変数のチュートリアルについては、「チュートリアル: パイプラインレベルの変数を使用する」を参照してください。

パイプラインレベルの変数を任意の種類のソースアクションで使用することはできません。

> **Note**
>
> variables 名前空間がパイプライン内のいくつかのアクションで既に使用されている場合、アクション定義を更新し、競合するアクションに別の名前空間を選択する必要があります。

アクションレベルの変数の設定

アクションの名前空間を宣言して、変数を生成するようにアクションを設定します。アクションは、すでに変数を生成するアクションプロバイダの 1 つであることが必要です。それ以外の場合、使用可能な変数はパイプラインレベルの変数です。

名前空間は、次のいずれかの方法で宣言します。

- コンソールの [アクションの編集] ページで、[Variable namespace (変数の名前空間)] に名前空間を入力します。
- JSON パイプライン構造の namespace パラメータフィールドに名前空間を入力します。

この例では、namespace パラメータを CodeCommit ソースアクションに SourceVariables という名前で追加します。これにより、そのアクションプロバイダ (CommitId など) で使用可能な変数を生成するようにアクションが設定されます。

```
{
    "name": "Source",
    "actions": [
        {
            "outputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "name": "Source",
            "namespace": "SourceVariables",
            "configuration": {
                "RepositoryName": "MyRepo",
                "BranchName": "mainline",
                "PollForSourceChanges": "false"
            },
            "inputArtifacts": [],
```

```
            "region": "us-west-2",
            "actionTypeId": {
                "provider": "CodeCommit",
                "category": "Source",
                "version": "1",
                "owner": "AWS"
            },
            "runOrder": 1
        }
    ]
},
```

次に、前のアクションによって生成された変数を使用するようにダウンストリームアクションを設定します。これは次の方法で行います。

- コンソールの [アクションの編集] ページで、アクション設定フィールドに変数構文 (ダウンストリームアクション用) を入力します。
- JSON パイプライン構造のアクション設定フィールドに変数構文 (ダウンストリームアクション用) を入力する

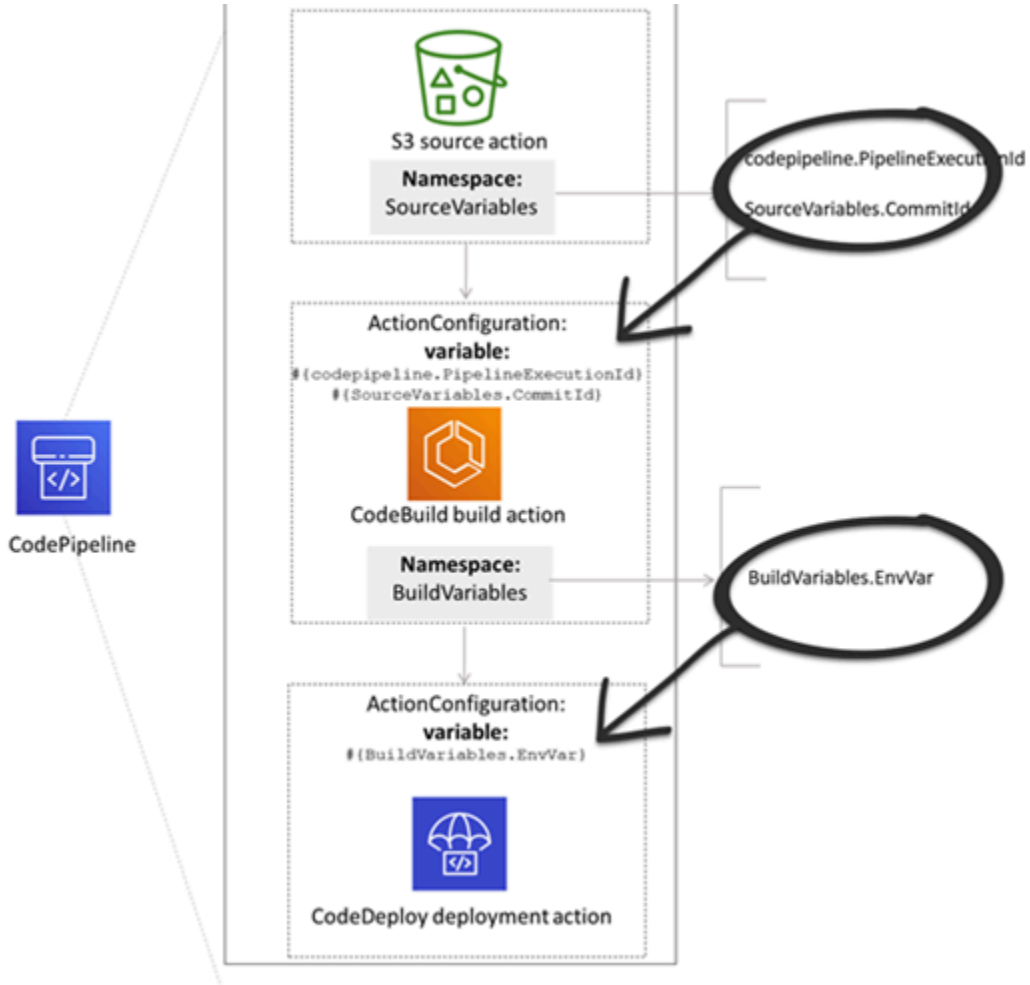
この例では、ビルドアクションの設定フィールドには、アクションの実行時に更新される環境変数が表示されます。この例では、#{codepipeline.PipelineExecutionId} で実行 ID の名前空間と変数を指定し、コミット ID には **#{SourceVariables.CommitId}** で名前空間と変数を指定します。

```
{
    "name": "Build",
    "actions": [
        {
            "outputArtifacts": [
                {
                    "name": "BuildArtifact"
                }
            ],
            "name": "Build",
            "configuration": {
                "EnvironmentVariables": "[{\"name\":\"Release_ID\",\"value\":
\"#{codepipeline.PipelineExecutionId}\",\"type\":\"PLAINTEXT\"},{\"name\":\"Commit_ID
\",\"value\":\"#{SourceVariables.CommitId}\",\"type\":\"PLAINTEXT\"}]",
                "ProjectName": "env-var-test"
            },
```

```
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-west-2",
            "actionTypeId": {
                "provider": "CodeBuild",
                "category": "Build",
                "version": "1",
                "owner": "AWS"
            },
            "runOrder": 1
        }
    ]
},
```

変数の解決

アクションがパイプライン実行の一部として実行されるたびに、アクションが生成する変数は、生成アクションの後に発生することが保証される任意のアクションで使用できます。これらの変数を消費アクションで使用するには、前の例で示した構文を使用して、消費アクションの設定に変数を追加します。消費アクションを実行する前に、CodePipeline は、アクションの実行を開始する前の設定に存在するすべての変数リファレスを解決します。



変数のルール

次のルールは、変数の設定に役立ちます。

- アクションの名前空間と変数を指定するには、新しいアクションプロパティを使用するか、アクションを編集します。
- パイプライン作成ウィザードを使用すると、ウィザードで作成された各アクションの名前空間がコンソールによって生成されます。
- 名前空間が指定されていない場合、そのアクションによって生成された変数は、どのアクション設定でも参照できません。
- アクションによって生成された変数を参照するには、参照アクションは、変数を生成するアクションの後に発生する必要があります。これは、変数を生成するアクションよりも後の段階にあるか、同じ段階にあるが、より高い実行順序にあることを意味します。

パイプラインアクションで使用できる変数

アクションプロバイダは、アクションによって生成できる変数を決定します。

変数を管理するステップバイステップの手順については、「変数の操作」を参照してください。

定義された変数キーを持つアクション

選択できる名前空間とは異なり、ほとんどの変数キーは編集できません。たとえば、Amazon S3 アクションプロバイダの場合、ETag および VersionId 変数キーのみを使用できます。

各実行には、パイプラインリリース ID など、実行に関するデータを含む CodePipeline で生成されたパイプライン変数のセットもあります。これらの変数は、パイプライン内の任意のアクションで消費できます。

トピック

- CodePipeline 実行 ID 変数
- Amazon ECR アクションの出力変数
- CloudFormation StackSets アクション出力変数
- CodeCommit アクションの出力変数
- CodeStarSourceConnection アクションの出力変数
- GitHub アクション出力変数 (GitHub (OAuth アプリ経由) アクション)
- S3 アクションの出力変数

CodePipeline 実行 ID 変数

CodePipeline 実行 ID 変数

プロバイダー	変数キー	値の例	変数構文例
codepipeline	`PipelineExecutionId`	8abc75f0-fbf8-4f4c-bfEXAMPLE	`#{codepipeline.PipelineExecutionId}`

Amazon ECR アクションの出力変数

Amazon ECR 変数

変数キー	値の例	変数構文例
`ImageDigest`	sha256:EXAMPLE1122334455	`#{SourceVariables.ImageDigest}`
`ImageTag`	最新	`#{SourceVariables.ImageTag}`
`ImageURI`	11111EXAMPLE.dkr.ecr.us-west-2.amazonaws.com/ecs-repo:latest	`#{SourceVariables.ImageURI}`
`RegistryId`	EXAMPLE12233	`#{SourceVariables.RegistryId}`
`RepositoryName`	my-image-repo	`#{SourceVariables.RepositoryName}`

CloudFormation StackSets アクション出力変数

CloudFormation StackSets 変数

変数キー	値の例	変数構文例
`OperationId`	11111111-2bbb-111-2bbb-11111 例	`#{DeployVariables.OperationId}`
`StackSetId`	my-stackset: 1111aaa-1111-2222-2bbb-11111 例	`#{DeployVariables.StackSetId}`

CodeCommit アクションの出力変数

CodeCommit 変数

変数キー	値の例	変数構文例
AuthorDate	2019-10-29T03:32:21Z	#{SourceVariables.AuthorDate}
BranchName	開発	#{SourceVariables.BranchName}
CommitId	exampleb01f91b31	#{SourceVariables.CommitId}
CommitMessage	バグを修正 (最大サイズ 100 KB)	#{SourceVariables.CommitMessage}
CommitterDate	2019-10-29T03:32:21Z	#{SourceVariables.CommitterDate}
RepositoryName	myCodeCommitRepo	#{SourceVariables.RepositoryName}

CodeStarSourceConnection アクションの出力変数

使用可能な出力変数は、トリガータイプ (プッシュ、プルリクエスト、またはタグ) によって異なります。All とマークされた次の表の変数は、トリガータイプごとに生成されます。特定のトリガータイプでマークされた変数は、パイプラインの実行がそのトリガーによって開始された場合にのみ生成されます。

CodeStarSourceConnection 変数 (Bitbucket Cloud、GitHub、GitHub Enterprise Repository、および GitLab.com)

変数キー	値の例	変数構文例	トリガータイプ
AuthorDate	2019-10-29T03:32:21Z	#{SourceVariables.AuthorDate}	すべて
AuthorDisplayName	Jane Doe	#{SourceVariables.AuthorDisplayName}	すべて
AuthorEmail	jane.doe@example.com	#{SourceVariables.AuthorEmail}	すべて
AuthorId	ジェネド	#{SourceVariables.AuthorId}	すべて
BranchName	開発	#{SourceVariables.BranchName}	プッシュ、タグ
CommitId	exampleb01f91b31	#{SourceVariables.CommitId}	すべて
CommitMessage	バグを修正 (最大サイズ 100 KB)	#{SourceVariables.CommitMessage}	すべて
ConnectionArn	arn:aws:codestar-connections:region:*account-id* :connection/ *connection-id*	#{SourceVariables.ConnectionArn}	すべて

変数キー	値の例	変数構文例	トリガータイプ
DestinationBranchName	メイン	#{SourceVariables.DestinationBranchName}	プルリクエスト
FullRepositoryName	Username/GitHubRepo	#{SourceVariables.FullRepositoryName}	すべて
ProviderType	GitHub	#{SourceVariables.ProviderType}	すべて
PullRequestId	1	#{SourceVariables.PullRequestId}	プルリクエスト
PullRequestTitle	新機能の追加	#{SourceVariables.PullRequestTitle}	プルリクエスト
SourceBranchName	feature-branch	#{SourceVariables.SourceBranchName}	プルリクエスト
TagName	v1.0.0	#{SourceVariables.TagName}	タグ

GitHub アクション出力変数 (GitHub (OAuth アプリ経由) アクション)

GitHub 変数 (GitHub (OAuth アプリ経由) アクション)

変数キー	値の例	変数構文例
AuthorDate	2019-10-29T03:32:21Z	#{SourceVariables.AuthorDate}
BranchName	メイン	#{SourceVariables.BranchName}
CommitId	exampleb01f91b31	#{SourceVariables.CommitId}
CommitMessage	バグを修正 (最大サイズ 100 KB)	#{SourceVariables.CommitMessage}
CommitterDate	2019-10-29T03:32:21Z	#{SourceVariables.CommitterDate}
CommitUrl		#{SourceVariables.CommitUrl}
RepositoryName	myGitHubRepo	#{SourceVariables.RepositoryName}

S3 アクションの出力変数

S3 変数

変数キー	値の例	変数構文例
ETag	example28be1c3	#{SourceVariables.ETag}
VersionId	exampleta_IUQCv	#{SourceVariables.VersionId}

ユーザー設定の変数キーを使用したアクション

CodeBuild、 CloudFormation、および Lambda アクションの場合、変数キーはユーザーが設定します。

トピック

- CloudFormation アクションの出力変数
- CodeBuild アクションの出力変数
- Lambda アクションの出力変数

CloudFormation アクションの出力変数

CloudFormation 変数

変数キー	変数構文例
CloudFormation アクションの場合、変数はスタックテンプレートの `Outputs`セクションで指定された値から生成されます。CloudFormation アクションモードのうち、出力を生成するのは、スタックの作成、スタックの更新、変更セットの実行など、スタックの作成や更新を伴うアクションモードのみです。変数を生成するアクションモードは次のとおりです。 • CREATE_UPDATE • CHANGE_SET_EXECUTE • CHANGE_SET_REPLACE • REPLACE_ON_FAILURE これらのアクションの詳細については、「CloudFormation デプロイアクションリファレンス」を参照してください。 CloudFormation 出力変数を使用するパイプラインで CloudFormation デプロイアクションを使用してパイプラインを作成する方法を示すチュートリアルについては、「」を参照してくださいチュートリアル: AWS CloudFormation デプロイアクションの変数を使用するパイプラインを作成する。	`#{DeployVariables.StackName}`

CodeBuild アクションの出力変数

CodeBuild 変数

変数キー	変数構文例
CodeBuild アクションの場合、変数はエクスポートされた環境変数によって生成された値から生成されます。CodePipeline で CodeBuild アクションを編集するか、buildspec に環境変数を追加して、CodeBuild 環境変数を設定します。CodePipeline CodeBuild buildspec に指示を追加して、エクスポートされた変数セクションの下に環境変数を追加します。env/exported-変数 の AWS CodeBuild ユーザーガイド を参照してください。	`#{BuildVariables.EnvVar}`

Lambda アクションの出力変数

Lambda 変数

変数キー	変数構文例
Lambda アクションは、変数として PutJobSuccessResult API リクエストのセクション`outputVariables` に含まれるすべてのキーと値のペアを生成します。 アップストリームアクション (CodeCommit) の変数を使用し、出力変数を生成する Lambda アクションのチュートリアルについては、 チュートリアル: Lambda 呼び出しアクションで変数を使用する を参照してください。	`#{TestVariables.testRunId}`

構文での glob パターンの使用

パイプラインのアーティファクトまたはソースロケーションで使用されるファイルまたはパスを指定する場合、アクションタイプに応じてアーティファクトを指定できます。例えば、S3 アクションでは S3 オブジェクトキーを指定します。

トリガーではフィルタを指定できます。glob パターンを使用してフィルタを指定できます。以下は例です。

構文が「glob」の場合、パスの文字列表現は正規表現に似た構文を持つ限定的なパターン言語を使用してマッチされます。例:

- `*.java` は、.java で終わるファイル名を表すパスを指定します。
- `*.*` は、ドットを含むファイル名を指定します。
- `*.{java,class}` は、.java または .class で終わるファイル名を指定します。
- `foo.?` は、foo. で始まり 1 文字の拡張子の付いたファイル名を指定します。

glob パターンの解釈には以下の規則が使用されます。

- ディレクトリ境界で 0 文字以上の名前要素を指定するには、* を使用します。
- ディレクトリ境界をまたいで 0 文字以上の名前要素を指定するには、** を使用します。
- 名前コンポーネントの 1 文字を指定するには、? を使用します。
- 特殊文字として解釈される文字をエスケープするには、バックスラッシュ文字 (\) を使用します。
- 文字セットから 1 文字を指定するには、[ ] を使用します。
- ビルドまたはソースリポジトリの場所のルートにある 1 つのファイルを指定するには、`my-file.jar` を使用します。
- サブディレクトリ内の 1 つのファイルを指定するには、`directory/my-file.jar` または `directory/subdirectory/my-file.jar` を使用します。
- すべてのファイルを指定するには、"**" を使用します。glob パターン ** は、任意の数のサブディレクトリにマッチすることを示します。
- `directory` という名前のディレクトリ内のすべてのファイルとディレクトリを指定するには、"directory/**" を使用します。glob パターン ** は、任意の数のサブディレクトリにマッチすることを示します。

- directory という名前のディレクトリ内のすべてのファイルを指定するが、そのサブディレクトリを含めないようにするには、"directory/*" を使用します。
- 角括弧式内では、*、?、および \ 文字は、文字通りの意味です。ハイフン (-) 文字は、角括弧内で最初の文字だった場合、または式を否定する ! の次の文字だった場合は、文字通りの意味です。
- 中括弧 ({ }) は、グループ内のサブパターンがマッチする場合にグループがマッチするサブパターンのグループを囲みます。カンマ (",") 文字は、サブパターンを分割するために使用します。グループはネストできません。

ポーリングパイプラインを推奨される変更検出方法に更新する

ポーリングを使用してソースの変更に対応するパイプラインがある場合は、推奨される検出方法を使用するようにパイプラインを更新できます。推奨されるイベントベースの変更検出方法を使用するようにポーリングパイプラインを更新する手順を含む移行ガイドについては、「」を参照してくださいポーリングパイプラインをイベントベースの変更検出の使用に移行する。

GitHub (OAuth アプリ経由) ソースアクションを GitHub (GitHub アプリ経由) ソースアクションに更新する

では AWS CodePipeline、GitHub ソースアクションの 2 つのサポートされているバージョンがあります。

- 推奨: GitHub (GitHub アプリ経由) アクションは、CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアクションの場合)リソースに基づく GitHub アプリベースの認証を使用します。GitHub でアクセスを管理できるように、 AWS CodeStar Connections アプリケーションを GitHub 組織にインストールします。
- 非推奨: GitHub (OAuth アプリ経由) アクションは OAuth トークンを使用して GitHub で認証し、別のウェブフックを使用して変更を検出します。これはもはや推奨される方法ではありません。

> **Note**
>
> 接続は、アジアパシフィック (香港)、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 (ジャカルタ)、アジアパシフィック (メルボルン)、アジアパシフィック (大阪)、アフリカ (ケープタウン)、中東 (バーレーン)、中東 (アラブ首長国連邦)、欧州 (スペイン)、欧州 (チューリッヒ)、イスラエル (テルアビブ)、または AWS GovCloud (米国西部) の各リージョンでは使用できません。利用可能なその他のアクションについては、「CodePipeline との製品とサービスの統合」を参照してください。欧州 (ミラノ) リージョンでのこのアクションに関する考慮事項については、「CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアクションの場合)」の注意を参照してください。

GitHub (OAuth アプリ経由) アクションの代わりに GitHub (GitHub アプリ経由) アクションを使用することには、いくつかの重要な利点があります。

- 接続により、CodePipeline はリポジトリにアクセスするために OAuth アプリやパーソナルアクセストークンを必要としなくなりました。接続を作成するときは、GitHub リポジトリへの認証を管理し、Organization レベルで権限を許可する GitHub アプリをインストールします。リポジトリにアクセスするには、OAuth トークンをユーザーとして承認する必要があります。アプリベースの GitHub アクセスとは対照的な OAuth ベースの GitHub アクセスの詳細については、https://

docs.github.com/en/developers/apps/differences-between-github-apps-and-oauth-apps を参照してください。

- CLI または CloudFormation で GitHub (GitHub アプリ経由) のアクションを管理する場合、Secrets Manager に個人アクセストークンをシークレットとして保存する必要がなくなりました。CodePipeline アクション設定で保存されたシークレットを動的に参照する必要がなくなりました。代わりに、アクション ARN に接続 ARN を追加します。アクション設定の例については、「CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアクションの場合)」を参照してください。
- CodePipeline で GitHub (GitHub アプリ経由) アクションで使用する接続リソースを作成する場合、同じ接続リソースを使用して、CodeGuru Reviewer など、サポートされている他のサービスをリポジトリに関連付けることができます。
- GitHub (GitHub アプリ経由) では、リポジトリのクローンを作成して後続の CodeBuild アクションで git メタデータにアクセスできますが、GitHub (OAuth アプリ経由) ではソースのみダウンロードできます。
- 管理者が Organization のリポジトリにアプリをインストールします。トークンを作成した個人に依存する OAuth トークンを追跡する必要がなくなりました。

Organization にインストールされているすべてのアプリは、同じリポジトリのセットにアクセスできます。各リポジトリにアクセスできるユーザーを変更するには、各接続の IAM ポリシーを変更します。例については、「例: 指定したリポジトリとの接続を使用するためのスコープダウンポリシー」を参照してください。

このトピックの手順を使用して GitHub (OAuth アプリ経由) のソースアクションを削除し、CodePipeline コンソールから GitHub (GitHub アプリ経由) のソースアクションを追加できます。

トピック

- ステップ 1: (OAuth アプリ経由) GitHub アクションを置き換える
- ステップ 2 : GitHub への接続を作成する
- ステップ 3: GitHub のソースアクションを保存する

ステップ 1: (OAuth アプリ経由) GitHub アクションを置き換える

パイプライン編集ページを使用して、(OAuth アプリ経由) GitHub アクションを GitHub (GitHub アプリ経由) アクションに置き換えます。

(OAuth アプリ経由) GitHub アクションを置き換えるには

1. CodePipeline コンソールにサインインします。
2. パイプラインを選択し、[編集] を選択します。ソースステージで、[ステージを編集] を選択します。アクションを更新することを推奨するメッセージが表示されます。
3. [アクションプロバイダー] で、[GitHub (GitHub アプリ経由)] を選択します。
4. 次のいずれかを行います。
   - [接続] でプロバイダへの接続をまだ作成していない場合は、[GitHub への接続] を選択します。ステップ 2: GitHub への接続を作成するに進みます。
   - [接続] でプロバイダへの接続を既に作成している場合は、その接続を選択します。ステップ 3： 接続のソースアクションを保存するに進みます。

ステップ 2 ：GitHub への接続を作成する

接続の作成を選択した後、[Connect to GitHub] ページが表示されます。

GitHub への接続を作成するには

1. [GitHub connection settings] で、[Connection name] に接続名が表示されます。

   [GitHub Apps] で、アプリケーションのインストールを選択するか、[Install a new app] (新しいアプリケーションをインストールする) を選択してアプリケーションを作成します。

   > **Note**
   >
   > 特定のプロバイダーへのすべての接続に対してアプリを 1 つインストールします。GitHub アプリをすでにをインストールしている場合は、これを選択してこのステップをスキップしてください。

2. GitHub の認可ページが表示されたら、認証情報を使用してログインし、続行を選択します。
3. アプリのインストールページで、 AWS CodeStar アプリが GitHub アカウントに接続しようとしていることを示すメッセージが表示されます。

> **Note**
>
> アプリは、GitHub アカウントごとに 1 回だけインストールします。アプリをインストール済みである場合は、Configure (設定) をクリックしてアプリのインストールの変更ページに進むか、戻るボタンでコンソールに戻ることができます。

4. [AWS CodeStarのインストール] ページで、[インストール] を選択します。
5. [Connect to GitHub] ページで、新規インストールの接続 ID が GitHub Apps に表示されます。接続 を選択します。

ステップ 3: GitHub のソースアクションを保存する

[アクションを編集] というページで更新を実行し、新しいソースアクションを保存します。

GitHub のソースアクションを保存するには

1. [リポジトリ] で、サードパーティーのリポジトリの名前を入力します。[ブランチ] で、パイプラインでソースの変更を検出するブランチを入力します。

> **Note**
>
> [Repository] で、例に示すように `owner-name/repository-name` を入力します。
>
> ```
> my-account/my-repository
> ```

2. [Output artifact format (出力アーティファクトのフォーマット)] で、アーティファクトのフォーマットを選択します。

   - デフォルトのメソッドを使用して GitHub アクションからの出力アーティファクトを保存するには、CodePipeline default を選択します。アクションは、Bitbucket リポジトリからファイルにアクセスし、パイプラインアーティファクトストアの ZIP ファイルにアーティファクトを保存します。
   - リポジトリへの URL 参照を含む JSON ファイルを保存して、ダウンストリームのアクションで Git コマンドを直接実行できるようにするには、[Full clone (フルクローン)] を選択します。このオプションは、CodeBuild ダウンストリームアクションでのみ使用できます。

このオプションを選択した場合は、Bitbucket、GitHub、GitHub Enterprise Server、または GitLab.com に接続するための CodeBuild GitClone アクセス許可を追加します。 で示されるように CodeBuild プロジェクトサービスロールの権限を更新する必要があります。フルクローン オプションの使い方を紹介したチュートリアルは、チュートリアル: CodeCommit パイプラインソースで完全なクローンを使用する をご覧ください。

3. 出力アーティファクト の場合、`SourceArtifact` のようにこのアクションの出力アーティファクトの名前を保持できます。[Done] を選択して、[アクションを編集] ページを閉じます。
4. [Done] を選択して、ステージの編集ページを閉じます。[Save] を選択して、パイプラインの編集ページを閉じます。

AWS CodePipeline のクォータ

CodePipeline には、アカウントが各 AWS リージョンに持つことができるパイプライン、ステージ、アクション、およびウェブフック AWS の数に対するクォータがあります。

以下のクォータは、リージョンごとに適用され、引き上げをリクエストできます。クォータの引き上げリクエストの処理には、最大 2 週間かかる場合があります。

[リソース]	デフォルト
アクションがタイムアウトするまでの時間 (これは設定可能なタイムアウトです。設定不可能なタイムアウトについては、次の表を参照してください)	CloudFormation デプロイアクション: 3 日間 CodeDeploy および CodeDeploy ECS (ブルー/グリーン) デプロイアクション: 5 日間 AWS Lambda 呼び出しアクション: 24 時間 **Note** アクションの実行中、CodePipeline は定期的に Lambda に連絡してステータスを確認します。Lambda 関数は、アクションの実行が成功、失敗、または進行中のステータスを返します。Lambda 関数が 20 分経っても応答を送信しない場合、アクションはタイムアウトになります。20 分の間に Lambda 関数が、アクションがまだ進行中であると返した場合、CodePipeline は 20 分のタイマーを再度開始し、再試行します。24 時間後に成功しなかった場合、CodePipeline は Lambda invoke アクションの状態を「失敗」に設定します。 Lambda には、CodePipeline アクションのタイムアウトとは関係なく

[リソース]	デフォルト
	Lambda 関数に対して個別のタイムアウトがあります。 Amazon S3 のデプロイアクション: 90 分 **Note** 大きな ZIP ファイルのデプロイ中に S3 へのアップロードがタイムアウトすると、アクションはタイムアウトエラーで失敗します。ZIP ファイルを小さなファイルに分割してみてください。 手動承認アクションのアカウントレベルのデフォルトタイムアウト: 7 日間 **Note** 手動承認アクションのデフォルトのタイムアウトは、パイプライン内のアクションごとに上書きできます。また、最小値を 5 分として、最大 86,400 分 (60 日) まで設定できます。詳細については、「CodePipeline API リファレンス」の「ActionDeclaration」を参照してください。 設定すると、このタイムアウトはアクションに適用されます。それ以外の場合は、アカウントレベルのデフォルトが使用されます。

[リソース]	デフォルト
	他のすべてのアクション: 1 時間 **Note** Amazon ECS デプロイアクションのタイムアウトは最大 1 時間 (デフォルトのタイムアウト) まで設定できます。
AWS アカウントのリージョンあたりの合計パイプラインの最大数	1,000 **Note** ポーリングまたはイベントベースの変更検出用に設定されたパイプラインは、このクォータにカウントされます。
AWS リージョンごとのソース変更のポーリングに設定するパイプラインの最大数	300 **Note** このクォータは固定されており、変更できません。ポーリングパイプラインの制限に達しても、まだイベントベースの変更検出を使用する追加のパイプラインを構成できます。詳細については、「ソースアクションと変更検出方法」を参照してください。[1]
AWS アカウントのリージョンあたりのウェブフックの最大数	300

[リソース]	デフォルト
AWS アカウントのリージョンあたりのカスタムアクションの数	50

[1]ソースプロバイダーに基づき、以下の手順に従って、ポーリングパイプラインをイベントベースの変更検出の使用に更新します。

- CodeCommit のソースアクションを更新するには、ポーリングパイプラインを移行する (CodeCommit または Amazon S3 ソース) (コンソール) を参照してください。
- Amazon S3 のソースアクションを更新するには、ポーリングパイプラインを移行する (CodeCommit または Amazon S3 ソース) (コンソール) を参照してください。
- GitHub ソースアクションを更新する方法については、「ポーリングパイプラインを Webhook に移行する (GitHub (OAuth アプリ経由) ソースアクション) (コンソール)」を参照してください。

の次のクォータは、リージョンの可用性、命名に関する制約、および許可されるアーティファクトサイズ AWS CodePipeline に適用されます。これらのクォータは固定されており、変更できません。

各リージョンの CodePipeline サービスエンドポイントのリストについては、AWS 全般リファレンスの「AWS CodePipeline エンドポイントとクォータ」を参照してください。

構造的要件については、「CodePipeline パイプライン構造リファレンス」を参照してください。

AWS パイプラインを作成できるリージョン	米国東部 (バージニア北部)
	米国東部 (オハイオ)
	米国西部 (北カリフォルニア)
	米国西部 (オレゴン)
	カナダ (中部)
	欧州 (フランクフルト)
	欧州 (チューリッヒ)*
	欧州 (ストックホルム)

	ヨーロッパ (ミラノ)*
	欧州 (スペイン)*
	欧州 (アイルランド)
	欧州 (ロンドン)
	欧州 (パリ)
	イスラエル (テルアビブ)*
	アフリカ (ケープタウン)**
	アジアパシフィック (香港)*
	アジアパシフィック (ムンバイ)
	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 (東京)
	アジアパシフィック (ソウル)
	アジアパシフィック (大阪)
	アジアパシフィック (シンガポール)
	アジアパシフィック (シドニー)
	アジアパシフィック (ジャカルタ)*
	アジアパシフィック (メルボルン)*
	アジアパシフィック (マレーシア)*
	アジアパシフィック (ニュージーランド)*
	南米 (サンパウロ)
	中東 (バーレーン)*
	中東 (アラブ首長国連邦)*

	AWS GovCloud (米国西部) AWS GovCloud (米国東部)
アクション名に使用できる文字	アクション名は 100 文字を超えることはできません。使用できる文字は次のとおりです。 小文字 (a ~ z)。 大文字 (A ~ Z)。 0 ~ 9 の数字。 特殊文字の . (ピリオド)、@ (アットマーク)、- (マイナス記号)、および _ (下線)。 スペースなどの他の文字は使用できません。
アクションの種類で使用される文字	アクションの種類名は 25 文字を超えることはできません。使用できる文字は次のとおりです。 小文字 (a ~ z)。 大文字 (A ~ Z)。 数値 (0 ~ 9)。 特殊文字の . (ピリオド)、@ (アットマーク)、- (マイナス記号)、_ (下線)。 スペースなどの他の文字は使用できません。

アーティファクト名に使用できる文字	アーティクラフト名は 100 文字を超えることはできません。使用できる文字は次のとおりです。 小文字 (a ~ z)。 大文字 (A ~ Z)。 0 ~ 9 の数字。 特殊文字の - (マイナス記号)、および _(下線) スペースなどの他の文字は使用できません。
パートナーのアクション名に使用できる文字	パートナーのアクション名は、CodePipeline の他のアクション名と同じ命名規則と制限に従う必要があります。具体的には、100 文字を超えることはできません。使用できる文字は次のとおりです。 小文字 (a ~ z)。 大文字 (A ~ Z)。 数値 (0 ~ 9)。 特殊文字の . (ピリオド)、@ (アットマーク)、- (マイナス記号)、_ (下線)。 スペースなどの他の文字は使用できません。

パイプライン名に使用できる文字	パイプライン名は 100 文字を超えることはできません。使用できる文字は次のとおりです。 小文字 (a ~ z)。 大文字 (A ~ Z)。 数値 (0 ~ 9)。 特殊文字の . (ピリオド)、@ (アットマーク)、- (マイナス記号)、_ (下線)。 スペースなどの他の文字は使用できません。
ステージ名に使用できる文字	ステージ名は 100 文字を超えることはできません。使用できる文字は次のとおりです。 小文字 (a ~ z)。 大文字 (A ~ Z)。 数値 (0 ~ 9)。 特殊文字の . (ピリオド)、@ (アットマーク)、- (マイナス記号)、_ (下線)。 スペースなどの他の文字は使用できません。
アクションがタイムアウトするまでの時間	CodeBuild ビルドアクション: 36 時間 テストアクション: 8 時間 カスタムアクション:24 時間 ステップ機能 がアクションを呼び出します:7 日 コマンドアクションのビルドタイムアウト: 55 分

アクション設定キーの最大長 (たとえば、CodeBuild 設定キーは `ProjectName` 、`PrimarySource` 、および `EnvironmentVariables`)。	50 文字
アクション設定値の最大長 (たとえば、CodeCommit アクション設定中の `RepositoryName` の設定の値は 1000 文字未満にする必要があります。 `"RepositoryName": "my-repo-name-less-than-1000-characters"`)	1000 文字
パイプラインあたりのアクションの最大数	1,000
パイプラインあたりの同時パイプライン実行の最大数 (QUEUED|PARALLEL モード)	50
PARALLEL モードパイプライン実行あたりの同時パイプライン実行の最大数	5
Amazon S3 オブジェクトの最大ファイル数	100,000
ステージで同時に実行されるアクションの最大数	100
ステージで実行されるシーケンシャルアクションの最大数	100

ソースステージのアーティファクトの最大サイズ	Amazon S3 バケットに保存されるアーティファクト: 7 GB CodeCommit または GitHub リポジトリに保存されるアーティファクト: 1 GB 例外: AWS Elastic Beanstalk を使用してアプリケーションをデプロイする場合、アーティファクトの最大サイズは常に 512 MB です。 例外: CloudFormation を使用してアプリケーションをデプロイする場合、アーティファクトの最大サイズは常に 256 MB です。 例外: `CodeDeployToECS` アクションを使用してアプリケーションをデプロイする場合、アーティファクトの最大サイズは常に 3 MB です。
Amazon ECS コンテナとイメージをデプロイするパイプライン中で使用されるイメージ定義 JSON ファイルの最大サイズ	100 KB
CloudFormation アクションの入力アーティファクトの最大サイズ	256 MB
`CodeDeployToECS` アクションの入力アーティファクトの最大サイズ	3 MB
`Step Functions` アクションの入力アーティファクトの最大サイズ	Step Functions アクションは Lambda で実行するため、Lambda 関数のアーティファクトサイズクォータと同じアーティファクトサイズクォータが適用されます。詳細については、「Lambda デベロッパーガイド」の「Lambda クォータ」を参照してください。

`ParameterOverrides` プロパティに保存できる JSON オブジェクトの最大サイズ	をプロバイダー CloudFormation とする CodePipeline デプロイアクションの場合、`ParameterOverrides` プロパティを使用して、 CloudFormation テンプレート設定ファイルの値を指定する JSON オブジェクトを保存します。`ParameterOverrides` プロパティに保存することができる JSON オブジェクトのサイズは、最大 1 キロバイトに制限されています。
ステージで実行されるアクションの数	最小 1、最大 50
各アクションで許可されるアーティファクトの数	各アクションで許可される入力および出力アーティファクトの数については、「アクションタイプ別の有効な入力/出力アーティファクトの数」を参照してください。
パイプラインの実行履歴情報を保持する月数	12
パイプラインのステージの数	最小 2、最大 50
パイプラインのタグ	タグでは、大文字と小文字が区別されます。リソースあたり最大 50。
パイプラインのタグキー名	任意の組み合わせで使用できる文字は、Unicode 文字、数字、スペース、および許可されている UTF-8 文字 (1 ~ 128 文字) です。使用できる文字は、+ - = . _ : / @ です。 タグキー名は一意である必要があり、各キーに使用できる値は 1 つのみです。タグは以下のようにはできません。 • 以下から開始します AWS。 • 空白文字のみで構成されている • 末尾にスペースを使用する • 絵文字、または以下の文字を含める: ? ^ * [\ ~ ! # $ % & * ( ) > < | " '

パイプラインのタグ値	任意の組み合わせで使用できる文字は、Unicode 文字、数字、スペース、および許可されている UTF-8 文字 (1 ~ 256 文字) です。使用できる文字は、+ - = . _ : / @ です。 使用できるキーの値は 1 つのみですが、多数のキー値と同じ値を含めることができます。タグは以下のようにはできません。 • 以下から開始します AWS。 • 空白文字のみで構成されている • 末尾にスペースを使用する • 絵文字、または以下の文字を含める: ? ^ * [\ ~ ! # $ % & * ( ) > < | " '

トリガ	パイプライン定義内のトリガー数は、push および pull request の設定全体で最大 50 個です。 プッシュトリガーおよびプルリクエストトリガーごとにフィルター数は最大 3 つです。 **Note** 同じイベントタイプ配列内でのフィルターの重複は許可されません。 イベントタイプ (プッシュ、プルリクエスト) ごとに最大 8 つの [含める] パターンと 8 つの [除外する] パターンを追加できます。 パターン値で許可される文字には、すべての文字タイプが含まれます。 [含める] パターンと [除外する] パターンの最大長は 255 文字です。 タグ名の最大長は 255 文字です。 triggers 配列の最大サイズは 200 KB を超えることはできません

トリガーフィルター	ファイルパス: • パターンの数: 最大 8 つの [含める] パターンと 8 つの [除外する] パターンを追加できます。 • パターンのサイズ: 各 [含める] パターンまたは [除外する] パターンのサイズは最大 255 文字です。 ブランチ: • パターンの数: 最大 8 つの [含める] パターンと 8 つの [除外する] パターンを追加できます。 • パターンのサイズ: 各 [含める] パターンまたは [除外する] パターンのサイズは最大 255 文字です。 プルリクエスト: ブランチ: • パターンの数: 最大 8 つの [含める] パターンと 8 つの [除外する] パターンを追加できます。 • パターンのサイズ: 各 [含める] パターンまたは [除外する] パターンのサイズは最大 255 文字です。

名前の一意性	1 つの AWS アカウント内で、 AWS リージョンで作成する各パイプラインには一意の名前が必要です。パイプラインの名前は、別の AWS リージョンで再利用できます。 ステージ名は、パイプライン内で一意である必要があります。 アクション名は、ステージ内で一意である必要があります。
出力変数と名前空間のクォータ	特定のアクションに対して結合されたすべての出力変数には、122880 バイトという最大サイズ制限があります。 特定のアクションの解決済みアクション設定の合計には、100 KB という最大サイズ制限があります。 出力変数名では、大文字と小文字が区別されます。 名前空間では、大文字と小文字が区別されます。 使用できる文字は次のとおりです。 • 小文字 (a ~ z)。 • 大文字 (A ~ Z)。 • 数値 (0 ~ 9)。 • 特殊文字の ^ (キャレット)、@ (アットマーク)、- (マイナス記号)、_ (下線)、[(左角カッコ)、] (右角カッコ)、* (アスタリスク)、$ (ドル記号)。 スペースなどの他の文字は使用できません。

パイプラインレベルの変数のクォータ	パイプラインレベルの変数は 1 パイプラインあたり最大 50 個です。 パイプラインレベルの変数の変数名は以下の要件を満たす必要があります。 • 最大文字数は 128 文字 • 小文字 (a ~ z)。 • 大文字 (A ~ Z)。 • 数値 (0 ~ 9)。 • 特殊文字 @\-_]+ スペースなどの他の文字は使用できません。 変数値の最大長は 1000 文字です。 変数の値には、すべての文字を使用できます。 変数の説明の最大長は 200 文字です。

*使用する前に、このリージョンを有効にする必要があります。

付録 A: GitHub (OAuth アプリ経由) ソースアクション

この付録では、CodePipeline の GitHub (OAuth アプリ経由) アクションについて説明します。

> **Note**
>
> GitHub (OAuth アプリ経由) アクションの使用はお勧めしませんが、GitHub (OAuth アプリ経由) アクションを持つ既存のパイプラインは何の影響なく引き続き動作します。GitHub (OAuth アプリ経由) アクションを持つパイプラインの場合、CodePipeline は OAuth ベースのトークンを使用して GitHub リポジトリに接続します。対照的に、GitHub アクション (GitHub App 経由) は接続リソースを使用して GitHub リポジトリに AWS リソースを関連付けます。接続リソースは、アプリベースのトークンを使用して接続します。接続を使用する推奨される GitHub アクションにパイプラインを更新する方法の詳細については GitHub (OAuth アプリ経由) ソースアクションを GitHub (GitHub アプリ経由) ソースアクションに更新する を参照してください。アプリベースの GitHub アクセスとは対照的な OAuth ベースの GitHub アクセスの詳細については、https://docs.github.com/en/developers/apps/differences-between-github-apps-and-oauth-apps を参照してください。

GitHub と統合するために、CodePipeline はパイプライン用の OAuth アプリケーションを作成します。CodePipeline は、GitHub (OAuth アプリ経由) ソースアクションでパイプラインの変更検出を管理するためにウェブフックを使用します。

> **Note**
>
> で GitHub (GitHub App 経由) ソースアクションを設定する場合 CloudFormation、GitHub トークン情報を含めたり、ウェブフックリソースを追加したりすることはありません。「ユーザーガイド」の「AWS::CodeStarConnections::Connection」に示すように、接続リソースを設定します CloudFormation 。

このリファレンスには、GitHub (OAuth アプリ経由) アクションに関する次のセクションが含まれています。

- パイプラインに GitHub (OAuth アプリ経由) ソースアクションとウェブフックを追加する方法については、「GitHub (OAuth アプリ経由) ソースアクションの追加」を参照してください。

- GitHub (OAuth アプリ経由) ソースアクションの設定パラメータと YAML/JSON スニペットの例については、「GitHub (OAuth アプリ経由) ソースアクションリファレンス」を参照してください。

> **Important**
>
> CodePipeline ウェブフックを作成するときは、独自の認証情報を使用したり、複数のウェブフック間で同じシークレットトークンを再利用したりしないでください。セキュリティを最適化するには、作成するウェブフックごとに一意のシークレットトークンを生成します。シークレットトークンは、ユーザーが指定する任意の文字列で、ウェブフックペイロードの整合性と信頼性を保護するために、CodePipeline に送信するウェブフックペイロードを GitHub で計算して署名するために使用します。独自の認証情報を使用したり、複数のウェブフック間で同じトークンを再利用したりすると、セキュリティの脆弱性につながる可能性があります。

> **Note**
>
> シークレットトークンを指定していた場合は、レスポンスで編集されます。

トピック

- GitHub (OAuth アプリ経由) ソースアクションの追加
- GitHub (OAuth アプリ経由) ソースアクションリファレンス

GitHub (OAuth アプリ経由) ソースアクションの追加

GitHub (OAuth アプリ経由) ソースアクションを CodePipeline に追加するには、次の手順を実行します。

- CodePipeline コンソールの [パイプラインの作成] ウィザード (カスタムパイプラインを作成する (コンソール))、または [アクションを編集する] ページを使用して、[GitHub] プロバイダオプションを選択します。コンソールは、ソースが変更されたときにパイプラインを開始するウェブフックを作成します。
- CLI を使用して、`GitHub` アクションのアクション設定を追加し、次のようにリソースを追加作成します。

- `GitHub` でのアクション設定の例を GitHub (OAuth アプリ経由) ソースアクションリファレンス で使用し、パイプラインを作成する (CLI) で表示されるようなアクションを作成します。
- 定期的にチェックを無効にし、変更検出を手動で作成します。これは、変更検出方法によりソースをポーリングすることでパイプラインが開始されるためです。ポーリングパイプラインを GitHub (OAuth アプリ経由) アクションのウェブフックに移行します。

GitHub (OAuth アプリ経由) ソースアクションリファレンス

> **Note**
>
> GitHub (OAuth アプリ経由) アクションの使用はお勧めしませんが、GitHub (OAuth アプリ経由) アクションを持つ既存のパイプラインは何の影響なく引き続き動作します。GitHub (OAuth アプリ経由) ソースアクションを使用するパイプラインの場合、CodePipeline は OAuth ベースのトークンを使用して GitHub リポジトリに接続します。対照的に、新しい GitHub アクション (GitHub App 経由) は、接続リソースを使用して GitHub リポジトリに AWS リソースを関連付けます。接続リソースは、アプリベースのトークンを使用して接続します。推奨される GitHub アクションを使用した接続でパイプラインを更新する方法の詳細については「GitHub (OAuth アプリ経由) ソースアクションを GitHub (GitHub アプリ経由) ソースアクションに更新する」 を参照してください。

設定された GitHub リポジトリとブランチで新しいコミットが行われたときに、パイプラインをトリガーします。

GitHub と統合するため、CodePipeline はパイプラインの OAuth アプリケーションまたは個人用アクセストークンを使用します。コンソールを使用してパイプラインを作成または編集する場合、CodePipeline は GitHub Webhook を作成し、リポジトリに変更が生じた場合に、パイプラインを開始するようにします。

GitHub アクションを介してパイプラインを接続する前に、GitHub アカウントとリポジトリを作成しておく必要があります。

CodePipeline のリポジトリへのアクセスを制限する場合は、GitHub アカウントを作成し、CodePipeline と統合するリポジトリのみにアカウントのアクセス許可を付与します。CodePipeline を設定する際は、そのアカウントを使用し、パイプラインのソースステージの GitHub リポジトリを使用します。

詳細については、GitHub ウェブサイトの GitHub 開発者向けドキュメントを参照してください。

トピック

- アクションタイプ
- 設定パラメータ
- 入力アーティファクト
- 出力アーティファクト
- 出力変数
- アクションの宣言 (GitHub の例)
- GitHub (OAuth) への接続
- 関連情報

アクションタイプ

- カテゴリ:`Source`
- 所有者: `ThirdParty`
- プロバイダー: `GitHub`
- バージョン: `1`

設定パラメータ

所有者

必須: はい

GitHub リポジトリを所有する GitHub ユーザーまたは組織の名前。

Repo

必須: はい

ソースの変更が検出されるリポジトリの名前。

ブランチ

必須: はい

ソースの変更が検出されるブランチの名前。

OAuthToken

必須: はい

GitHub リポジトリで CodePipeline が操作を実行できるようにするための GitHub 認証トークンを表します。エントリは、常に 4 つのアスタリスクでマスクされた状態で表示されます。これは、次のいずれかの値を表します。

- コンソールを使用してパイプラインを作成すると、CodePipeline は、OAuth トークンを使用して GitHub 接続を登録します。
- を使用してパイプライン AWS CLI を作成する場合、このフィールドで GitHub 個人用アクセストークンを渡すことができます。アスタリスク (****) を GitHub からコピーした個人用アクセストークンに置き換えます。`get-pipeline` を実行してアクション設定を表示すると、この値の 4 つのアスタリスクマスクが表示されます。
- CloudFormation テンプレートを使用してパイプラインを作成する場合は、まずトークンをシークレットとして保存する必要があります AWS Secrets Manager。このフィールドの値は、`{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}` など Secrets Manager に保存されたシークレットへの動的リファレンスとして含めます。

GitHub のスコープに関する詳細については、GitHub ウェブサイトの「GitHub 開発者 API リファレンス」を参照してください。

PollForSourceChanges

必須: いいえ

`PollForSourceChanges` は、 CodePipeline が GitHub リポジトリをポーリングしてソースを変更するかどうかを制御します。この方法の代わりに、ウェブフックを使用してソースの変更を検出することをお勧めします。ウェブフックの設定に関する詳細は、「ポーリングパイプラインを Webhook に移行する (GitHub (OAuth アプリ経由) ソースアクション) (CLI)」または「プッシュイベントのパイプラインを更新する (GitHub (OAuth アプリ経由) ソースアクション) (CloudFormation テンプレート)」を参照してください。

> **Important**
>
> ウェブフックを設定する場合、パイプライン実行の重複を避けるため、`PollForSourceChanges` を `false` に設定します。

このパラメータの有効な値:

- `True`: 設定されている場合、CodePipeline はソースの変更についてポーリングします。

  > **Note**
  >
  > `PollForSourceChanges` を省略すると、CodePipeline はデフォルトでソースの変更についてポーリングします。この動作は、`PollForSourceChanges` が `true` に設定されている場合と同じです。

- `False`: 設定されている場合、CodePipeline は、ソースの変更についてリポジトリをポーリングしません。ソースの変更を検出するようにウェブフックを構成する場合は、この設定を使用します。

入力アーティファクト

- アーティファクトの数: `0`
- 説明: 入力アーティファクトは、このアクションタイプには適用されません。

出力アーティファクト

- アーティファクトの数: `1`
- 説明: このアクションの出力アーティファクトは、パイプライン実行のソースリビジョンとして指定されたコミットで設定されたリポジトリとブランチの内容を含む ZIP ファイルです。リポジトリから生成されたアーティファクトは、GitHub アクションの出力アーティファクトです。ソースコードのコミット ID は、パイプライン実行のトリガーとなるソースリビジョンとして、CodePipeline に表示されます。

出力変数

このアクションを設定すると、パイプライン内のダウンストリームアクションのアクション設定によって参照できる変数が生成されます。このアクションは、アクションに名前空間がない場合でも、出力変数として表示できる変数を生成します。名前空間を使用してアクションを設定し、これらの変数をダウンストリームアクションの設定で使用できるようにします。

CodePipeline の変数についての詳細は、「変数リファレンス」を参照してください。

CommitId

パイプライン実行をトリガーした GitHub コミット ID。コミット ID は、コミットの完全な SHA です。

CommitMessage

パイプライン実行をトリガーしたコミットに関連付けられた説明メッセージ (存在する場合)。

CommitUrl

パイプラインをトリガーしたコミットの URL アドレス。

RepositoryName

パイプラインをトリガーしたコミットが実行された GitHub リポジトリの名前。

BranchName

ソースの変更が行われた GitHub リポジトリのブランチの名前。

AuthorDate

コミットが認証された日付 (タイムスタンプ形式)。

CommitterDate

コミットがコミットされた日付 (タイムスタンプ形式)。

アクションの宣言 (GitHub の例)

YAML

```
Name: Source
Actions:
  - InputArtifacts: []
    ActionTypeId:
      Version: '1'
      Owner: ThirdParty
      Category: Source
      Provider: GitHub
    OutputArtifacts:
      - Name: SourceArtifact
    RunOrder: 1
    Configuration:
      Owner: MyGitHubAccountName
      Repo: MyGitHubRepositoryName
```

```
      PollForSourceChanges: 'false'
      Branch: main
      OAuthToken: '{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}'
    Name: ApplicationSource
```

JSON

```
{
    "Name": "Source",
    "Actions": [
        {
            "InputArtifacts": [],
            "ActionTypeId": {
                "Version": "1",
                "Owner": "ThirdParty",
                "Category": "Source",
                "Provider": "GitHub"
            },
            "OutputArtifacts": [
                {
                    "Name": "SourceArtifact"
                }
            ],
            "RunOrder": 1,
            "Configuration": {
                "Owner": "MyGitHubAccountName",
                "Repo": "MyGitHubRepositoryName",
                "PollForSourceChanges": "false",
                "Branch": "main",
                "OAuthToken":
 "{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}"
            },
            "Name": "ApplicationSource"
        }
    ]
},
```

GitHub (OAuth) への接続

初めて GitHub リポジトリをパイプラインに追加するコンソールを使用する場合、CodePipeline のリポジトリへのアクセスを承認するように要求されます。トークンには、次の GitHub スコープが必要です。

- `repo` スコープ。これは、パブリックおよびプライベートリポジトリからパイプラインにアーティファクトを読み込んでプルする完全制御に使用されます。
- `admin:repo_hook` スコープ。これは、リポジトリフックの完全制御に使用されます。

CLI または CloudFormation テンプレートを使用する場合は、GitHub で既に作成した個人用アクセストークンの値を指定する必要があります。

関連情報

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- AWS CloudFormation ユーザーガイド AWS::CodePipeline::Webhook のリソースリファレンス – これには、リソースのフィールド定義、例、スニペットが含まれます CloudFormation。
- AWS CloudFormation AWS::CodeStar::GitHubRepository ユーザーガイド のリソースリファレンス フィールド定義、例および CloudFormationにあるリソースに対するスニペットが含まれます。
- チュートリアル: を使用して Android アプリを構築およびテストするパイプラインを作成する AWS Device Farm – このチュートリアルでは、GitHub ソースを使用してパイプラインを作成するためのサンプルの buildspec ファイルとサンプルアプリケーションを提供します。CodeBuild および AWS Device Farmを使用して Android アプリ を構築し、テストします。

AWS CodePipeline ユーザーガイドのドキュメント履歴

次の表に、CodePipeline 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。このドキュメントの更新に関する通知を受け取るには、RSS フィードにサブスクライブできます。

- API バージョン: 2015-07-09
- ドキュメントの最終更新日: 2025 年 5 月 23 日

変更	説明	日付
パイプライントリガーフィルターの動作に関するドキュメントを修正しました	包含と除外の関係を正確に記述するために、トリガーフィルターの例のドキュメントを修正しました。除外は、ブランチおよびファイルパスフィルターのインクルードよりも優先されます。詳細については、「トリガーフィルターの例」を参照してください。	2026 年 3 月 16 日
Azure DevOps のサポートを追加	Azure DevOps とやり取りする AWS リソースの接続を設定するためのサポートが追加されました。詳細については、「Azure DevOps の接続」を参照してください。	2025 年 8 月 5 日
EC2 デプロイアクションのデプロイ仕様ファイルドキュメントを更新	EC2 デプロイアクションのデプロイ仕様ファイルの使用に関する新しい情報を追加し、例と設定オプションを更新しました。「Amazon EC2 アクションリファレンス」のアクションリファレンスページを参照してください。	2025 年 5 月 16 日

Lambda デプロイアクションのドキュメントを追加	更新された例と設定オプションを含む新しい Lambda デプロイアクションのサポートが追加されました。「AWS Lambda デプロイアクションリファレンス」のアクションリファレンスページを参照してください。	2025 年 5 月 16 日
コンピューティングタイプ、環境タイプ、環境変数タイプのコマンドアクションのフィールドを追加	コマンドアクションのビルド環境とコンピューティングタイプを指定できる新しいフィールドに関する情報を追加しました。コマンドアクションに環境変数の新しい型フィールドも追加されました。「CodePipeline コマンドアクション」のアクションリファレンスページを参照してください。	2025 年 5 月 8 日
デフォルトのサービスロールポリシーを文書化した新しいトピック	最小サービスロールポリシーに関する情報を追加し、CodePipeline の各アクションに対する追加のサービスロールのアクセス許可へのリンクの表を更新しました。「CodePipeline サービスロールポリシー」の新しいルールリファレンスページを参照してください。	2025 年 3 月 26 日

新しい CodePipeline 呼び出しアクション	新しい CodePipeline 呼び出しアクションに関する情報を追加しました。「CodePipeline 呼び出しアクションリファレンス」のアクションリファレンスページを参照してください。	2025 年 3 月 14 日
新しい CodeBuild ルール	ステージ条件のルールとしてビルドプロジェクトを実行するために使用できる新しい CodeBuild ルールに関する情報を追加しました。「CodeBuild ルールリファレンス」の新しいルールリファレンスページを参照してください。	2025 年 3 月 14 日
での接続の共有 AWS CodeConnections	を使用して接続するパイプラインの場合 AWS CodeConnections、共有用に設定された接続を使用できます AWS アカウント。共有接続は AWS Resource Access Managerで設定できます。詳細については、「別の AWS アカウントと共有されている接続を使用する」を参照してください。	2025 年 3 月 6 日

ソースリビジョンの EventBridge 入力変換エントリ	CodeCommit、Amazon ECR、Amazon S3 の各ソースを含むパイプラインの場合、ソースリビジョンに EventBridge 入力変換エントリを使用できます。ここで、`revisionValue` はオブジェクトキー (S3)、コミット (CodeCommit)、またはイメージ ID (ECR) のソースイベント変数から派生します。「Amazon ECR ソースアクションと EventBridge リソース」、「イベントに対してソースを有効にした Amazon S3 ソースアクションへの接続」、「CodeCommit ソースアクションと EventBridge」を参照してください。	2025 年 3 月 3 日
AWS CodeBuild アクションのビルド仕様オーバーライド	代わりにコマンドを直接入力しながら、CodeBuild アクションのビルド仕様を上書きすることができます。「AWS CodeBuild のビルドとテストアクションリファレンス」のアクションリファレンスページを参照してください。「パイプライン、ステージ、アクションを作成する」も参照してください。	2025 年 3 月 3 日

新しい EC2 デプロイアクション	新しい EC2 デプロイアクションに関する情報を追加しました。「Amazon EC2 アクションリファレンス」のアクションリファレンスページを参照してください。チュートリアルについては、「チュートリアル: CodePipeline を使用して EC2 インスタンスにデプロイする」を参照してください。	2025 年 2 月 21 日
新しい EKS デプロイアクション	新しい EKS デプロイアクションに関する情報を追加しました。「EKS デプロイアクション」のアクションリファレンスページを参照してください。チュートリアルについては、「チュートリアル: CodePipeline を使用して Amazon EKS にデプロイする」を参照してください。	2025 年 2 月 20 日
CodePipeline の新しい CloudWatch メトリクスとディメンション	CloudWatch メトリクスにパイプラインのメトリクスを追加しました。「CodePipeline の CloudWatch メトリクス」を参照してください。	2025 年 2 月 13 日
新しい Commands ルール	ステージ条件のルールとしてシェルコマンドを実行するために使用できる新しい Commands ルールに関する情報を追加しました。「コマンドルールリファレンス」の新しいルールリファレンスページを参照してください。	2024 年 12 月 17 日

トリガー向けの拡張された例とリファレンス情報	「プロバイダー別のトリガーのプルリクエストイベント」に、プロバイダー別のプルリクエストイベントフィルターの説明を追加しました。「プロバイダー別のトリガーのプルリクエストイベント」に、プッシュイベントとプルリクエストイベントの包含および除外に関する詳細情報を含むトリガーの例を追加しました。「トリガー」に包含と除外に関する JSON リファレンス情報を追加しました。	2024 年 12 月 17 日
アクションカタログの新しいアクション	新しい ECRBuildAndPublish アクションと InspectorScan アクションを使用できるようになりました。詳細については、ECRBuildAndPublish アクションおよび InspectorScan アクションのリファレンスページを参照してください。	2024 年 11 月 22 日
障害時のステージ再試行の新しい自動設定	失敗したステージまたはステージ内の失敗したアクションを自動的に再試行するようにステージを設定できます。詳細については、「ステージ障害時の自動再試行を設定する」を参照してください。	2024 年 10 月 15 日

入力条件の新しい Skip 結果	入力条件で使用できる Skip 結果に関する情報を追加しました。この設定では、VariableCheck ルールと LambdaInvoke ルールを使用できます。「スキップ結果を使用した入力条件の作成」の手順を参照してください。スキップ結果を使用した条件に関する考慮事項のリストについては、「ステージ条件に設定する結果に関する考慮事項」を参照してください。	2024 年 10 月 15 日
静的テンプレートからパイプラインを作成するための新しいコンソール手順	CodePipeline コンソールで新しいパイプライン作成ウィザードを使用し、複数の静的テンプレートから選択して CloudFormationでパイプラインリソースを生成できるようになりました。詳細については、「静的テンプレートからパイプラインを作成する」を参照してください。	2024 年 10 月 9 日

新しい Commands アクション	パイプライン内のアクションとしてシェルコマンドを実行するために使用できる新しい Commands アクションに関する情報を追加しました。新しいアクションリファレンスページの「コマンドアクション」と「CodePipeline サービスロールにアクセス許可を追加する」のサービスロールのアクセス許可を参照してください。チュートリアルについては、「チュートリアル: コンピューティングでコマンドを実行するパイプラインを作成する」を参照してください。	2024 年 10 月 3 日
条件の新しい VariableCheck ルール	ステージ条件の VariableCheck ルールに関する情報を追加しました。新しいルールリファレンスページで「VariableCheck」を参照してください。チュートリアルについては、「チュートリアル: 入力条件としてパイプラインの変数チェックルールを作成する」を参照してください。	2024 年 9 月 27 日

GitHub の接続の更新	GitHub ユーザーアクセストークンを GitHub (GitHub V2 アクション) への接続で使用する方法に関する情報を追加しました。ユーザーアクセストークンは CodeBuild プロジェクトで使用します。「GitHub 接続」と「チュートリアル: GitHub パイプラインソースで完全なクローンを使用する」を参照してください。	2024 年 9 月 16 日
パイプライン JSON リファレンスとガイドの目次を再構築するための更新	ガイドを再構成し、一部のセクションタイトルを変更するなどして、リファレンスセクションとタスクセクションを利用しやすくしました。	2024 年 8 月 16 日
PutWebhook アクションおよび ListWebhooks アクションのレスポンスのシークレットトークンフィールドの更新	PutWebhook アクションおよび ListWebhooks アクションのシークレットトークンフィールドを更新しました。シークレットトークンを指定していた場合は、レスポンスで編集されます。「付録 A: GitHub バージョン 1 のソースアクション」に追加された注記を参照してください。「CodePipeline API ガイド」の関連する更新については、「PutWebhook」と「ListWebhooks」を参照してください。	2024 年 8 月 6 日

ステージ条件およびルールに新しいコンテンツを追加	V2 タイプのパイプラインのステージ条件およびルールを設定できるようになりました。「概念」、「ステージ条件はどのように機能しますか?」、「ステージの条件を設定する」を参照してください。リファレンス情報を提供する、ルールリファレンスの章を追加しました。CodePipeline のルールリファレンスを参照してください。	2024 年 7 月 30 日
パイプラインタイプおよび関連機能の新しいリファレンス情報を追加	V2 タイプのパイプラインへの移行コストを評価する新しい分析スクリプトを利用できます。「適切なパイプラインのタイプの選択」を参照してください。CodePipeline サービスドキュメント全体の機能別リンクを提供するリファレンステーブルを追加しました。「CodePipeline 機能リファレンス」を参照してください。	2024 年 7 月 11 日

ソースの上書きの新しいオプションを追加するための S3 ソースアクションの更新	ソースの上書きの新しいオプション S3_OBJECT_KEY が、S3 ソースアクションで利用可能になりました。S3 ソースアクションに新しい AllowOverrideForS3ObjectKey パラメータを追加しました。「Amazon S3 ソースアクション」リファレンスページと「ソースリビジョンの上書きでパイプラインを開始する」を参照してください。	2024 年 6 月 7 日
S3 ソースアクションを更新して新しい出力変数を追加	新しい出力変数 BucketName と ObjectKey が S3 ソースアクションで利用できるようになりました。「Amazon S3 ソースアクション」リファレンスページを参照してください。	2024 年 6 月 5 日
CloudFormationStackSet および CloudFormationStackInstances アクションの更新	CloudFormationStackSet および CloudFormationStackInstances アクションに CallAs パラメータが追加されました。「アクションリファレンスページ」を参照してください。	2024 年 5 月 2 日
ステージレベルのロールバックのサポート	ステージは、ステージの以前の成功したパイプライン実行に手動または自動でロールバックできます。「ステージロールバックの設定」と「概念」を参照してください。	2024 年 4 月 26 日

StackSets アクションと Step Functions アクションを利用可能なリージョンの更新	StackSets アクションと Step Functions アクションが、CodePipeline が利用可能なすべてのリージョンで利用可能になりました。「CloudFormation StackSets アクションリファレンス」と「AWS Step Functions アクションリファレンス」を参照してください。	2024 年 3 月 27 日
マネージドポリシーの更新	AWS 管理ポリシーが更新されAWSCodePipeline_FullAccess ました。AWS CodePipelineのAWS マネージドポリシーを参照してください。	2024 年 3 月 15 日
手動承認アクションの設定可能なタイムアウトのサポート	手動承認アクションの新しい設定可能なタイムアウトフィールドに関するクォータ情報を追加しました。詳細については、「クォータ」を参照してください。	2024 年 2 月 15 日

ブランチとファイルパスによるトリガーフィルタリングのサポート	V2 タイプのパイプラインのプルリクエストステータス、ブランチ、ファイルパスに対するフィルタリングを許可するトリガー設定のサポートを追加しました。詳細については、「コードプッシュまたはプルリクエストのトリガーのフィルタリング」、「トリガー」、「パイプラインを開始するための機能ブランチをフィルタリングする」、「クォータ」を参照してください。	2024 年 2 月 8 日
新しいパイプライン実行モードのサポート	パイプライン実行モード PARALLEL および QUEUED のサポートを追加しました。詳細については、「パイプライン実行モードを設定する」、「QUEUED モードでの実行の処理方法」、「PARALLEL モードでの実行の処理方法」、「クォータ」を参照してください。	2024 年 2 月 8 日

アクションの詳細を表示したり、手動承認アクションを確認したり、パイプラインのリストページを表示したりするためのコンソールページの更新	新しい [詳細を表示] ボタンとダイアログボックス、新しい手動承認ダイアログ、およびパイプラインのリストページの最近の実行に関する新しい列について、コンソールの更新内容が文書化されました。詳細については、「パイプラインの表示 (コンソール)」、「パイプラインのアクションの詳細の表示」、および「パイプラインの承認アクションの管理」を参照してください。	2024 年 1 月 10 日
GitLab セルフマネージドのサポート	GitLab セルフマネージドとやり取りする AWS リソースの接続の設定のサポートが追加されました。詳細については、「GitLab セルフマネージドとの接続」を参照してください。	2023 年 12 月 28 日
CloudFormationStackSet および CloudFormationStackInstances アクションの更新	CloudFormationStackSet および CloudFormationStackInstances アクションに ConcurrencyMode パラメータが追加されました。「アクションリファレンスページ」を参照してください。	2023 年 12 月 19 日

CodePipeline の AWS Device Farm アクションパラメータの更新	CodePipeline の AWS Device Farm アクションのパラメータが更新されました。詳細については、「AWS Device Farm アクションリファレンス」を参照してください。	2023 年 12 月 18 日
CodePipeline の CloudFormation アクションの詳細なエラーメッセージのサポートが追加されました	CloudFormation アクションエラーメッセージに、失敗したリソースに関する詳細が表示されるようになりました。詳細については、「CloudFormation アクションリファレンス」を参照してください。	2023 年 12 月 15 日
CodePipeline のソースリビジョンオーバーライドでパイプラインを起動するための更新	指定したソースリビジョンでパイプラインを起動できるようになりました。詳細については、「ソースリビジョンオーバーライドでパイプラインを開始する」を参照してください。	2023 年 11 月 17 日
「サポートされているリージョン」を参照してください。	CodePipeline 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 (ジャカルタ)、アジアパシフィック (メルボルン)、アジアパシフィック (大阪)、中東 (アラブ首長国連邦)、欧州 (スペイン)、およびイスラエル (テルアビブ) リージョンで利用できるようになりました。「イベントプレースホルダーバケットリファレンス」トピックと「AWS のサービス エンドポイント」トピックが更新されました。	2023 年 11 月 13 日

Amazon EventBridge のイベントフィールドの更新	Amazon EventBridge で、更新されたイベントフィールドを表示できるようになりました。詳細については、「CodePipeline イベントのモニタリング」を参照してください。	2023 年 11 月 9 日
CodePipeline での新しいパイプラインタイプ V2 パイプライン、Git タグでのトリガー、およびパイプライン変数の更新	CodePipeline で、パイプラインのタイプを選択できるようになりました。V2 タイプのパイプラインで、Git タグでパイプラインを開始するためのトリガー設定を使用できるようになりました。V2 タイプのパイプラインでは、パイプラインレベルの変数を使用して、パイプライン実行の入力パラメータを渡すこともできます。詳細については、「変数」、「チュートリアル: パイプラインレベルの変数を使用する」、「チュートリアル: Git タグを使用してパイプラインを開始する」を参照してください。パイプラインのタイプの詳細については、「パイプラインのタイプ」を参照してください。	2023 年 10 月 24 日

CodePipeline で失敗したステージのすべてのアクションを再試行することが可能に	CodePipeline でステージが失敗した場合、パイプラインを再実行せずにステージを再試行できるようになりました。そのためには、ステージ内の失敗したアクションを再試行するか、ステージ内の最初のアクションから始めてステージ内のすべてのアクションを再試行します。詳細については、「」を参照してください。	2023 年 10 月 17 日
GitLab グループのサポート	GitLab グループとやり取りするための AWS リソースの接続の設定のサポートが追加されました。詳細については、「GitLab との接続」を参照してください。	2023 年 9 月 15 日
CodePipeline が GitLab.com への接続をサポート	接続を使用して、GitLab.com とやり取りするように AWS リソースを設定できます。GitLab.com. 下流のアクションで Git コマンドおよびメタデータを使うための完全なクローンオプションを選択することもできます。詳細については、「GitLab connections」および「CodeStarSourceConnection アクション構造リファレンス」トピックを参照してください。	2023 年 8 月 10 日

CloudFormationStackInstances アクションの更新	CloudFormationStackInstances アクションに RegionConcurrencyType パラメータが追加されました。CloudFormationStackInstances アクションについては、アクションリファレンスページを参照してください。	2023 年 8 月 8 日
CloudFormationStackSet アクションの更新	CloudFormationStackSet アクションに RegionConcurrencyType パラメータが追加されました。CloudFormationStackSet アクションについては、アクションリファレンスページを参照してください。	2023 年 7 月 24 日
マネージドポリシーの更新	AWS 管理ポリシーが更新されAWSCodePipeline_FullAccess ました。「AWS CodePipelineのAWS マネージドポリシー」を参照してください。	2023 年 6 月 21 日

ポーリングパイプラインの移行手順の更新	ポーリングパイプラインをイベントベースの変更検出の使用に移行 (更新) する手順が、EventBridge への通知を有効にした Amazon S3 バケットを使用するパイプラインのステップで更新されました。詳細については、「ポーリングパイプラインをイベントベースの変更検出の使用に移行する」を参照してください。	2023 年 6 月 12 日
マネージドポリシーの更新	AWS 管理ポリシー `AWSCodePipeline_FullAccess` と `AWSCodePipeline_ReadOnlyAccess` が追加のアクセス許可で更新されました。詳細については、「AWS CodePipelineAWS 管理ポリシーの更新」を参照してください。	2023 年 5 月 16 日
マネージドポリシーの更新	AWS 管理ポリシー `AWSCodePipelineFullAccess` および `AWSCodePipelineReadOnlyAccess` は廃止されました。`AWSCodePipeline_FullAccess` および `AWSCodePipeline_ReadOnlyAccess` ポリシーを使用してください。「AWS CodePipeline での AWS マネージドポリシーの更新」を参照してください。	2022 年 11 月 17 日

CloudTrail を使用する手順の更新	S3 CloudFormation ソースを持つパイプラインのすべてのコンソール手順、サンプル CLI コマンド、サンプルスニペットとテンプレートが更新され、CloudTrail の管理イベントに対して書き込みと false を選択するオプションが追加されました。更新されたサンプルについては、「パイプラインの開始」、「チュートリアル: でパイプラインを作成する CloudFormation」、「プッシュイベントを使用するようにパイプラインを編集する」、「ポーリングパイプラインを更新する」を参照してください。	2022 年 4 月 27 日
新しくサポートされた Snyk との統合	CodePipeline で Snyk の呼び出しアクションを使用すると、自動的にオープンソースコードのセキュリティスキャンを行うことができます。詳細は [Snyk アクションリファレンス] と [統合] を参照してください。	2021 年 6 月 10 日
新しくサポートされる欧州 (ミラノ) リージョン	CodePipeline が欧州 (ミラノ) で利用できるようになりました。「制限」および「AWS のサービス のエンドポイント」トピックが更新されました。	2021 年 1 月 27 日

変更の検出は、接続ソースアクションでオフにできます	ソースリポジトリの自動変更検出をオフにするためには、CLI または SDK を使って `CodeStarSourceConnection` ソースアクションを更新します。[CodeStarSourceConnection アクション構造リファレンス] のトピックが `DetectChanges` パラメータの説明で更新されました。	2021 年 1 月 8 日
CodePipeline が CloudFormation StackSets デプロイアクションをサポートするようになりました	新しいチュートリアル「チュートリアル: デプロイプロバイダーとして CloudFormation StackSets を使用するパイプラインを作成する」では、CloudFormation StackSets を使用してパイプラインでスタックセットとスタックインスタンスを作成および更新する手順について説明します。「CloudFormation StackSets アクション構造リファレンス」トピックも追加されました。	2020 年 12 月 30 日
新しくサポートされるアジアパシフィック (香港) リージョン	CodePipeline が、アジアパシフィック (香港) で使用できるようになりました。「制限」および「AWS のサービス のエンドポイント」トピックが更新されました。	2020 年 12 月 22 日

CodePipeline で更新された EventBridge イベントパターンを表示します	パイプライン、ステージ、およびアクションレベルのイベントで更新されたイベントパターンとステータスが [Monitoring CodePipeline イベント] に追加されました。	2020 年 12 月 21 日
CodePipeline でインバウンド実行のパイプラインを表示します	コンソールまたは CLI を使ってインバウンド実行を表示できます。詳細については、[インバウンド実行の表示 (コンソール)] と [インバウンド実行のステータス表示 (CLI)] を参照してください。	2020 年 11 月 16 日
CodePipeline の CodeCommit ソースアクションは、完全なクローンオプションをサポートしています	CodeCommit ソースアクションを使用する場合、下流の CodeBuild アクションで Git コマンドおよびメタデータを使える完全なクローンオプションを選択できます。詳細については、[CodeCommit アクションリファレンス] と [チュートリアル :CodeCommit パイプラインソースで完全なクローンを使用する] を参照してください。	2020 年 11 月 11 日

CodePipeline が GitHub および GitHub Enterprise Server への接続をサポート	接続を使用して、GitHub、GitHub Enterprise Cloud、および GitHub Enterprise Server とやり取りするように AWS リソースを設定できます。下流のアクションで Git コマンドおよびメタデータを使うための完全なクローンオプションを選択することもできます。詳細については、[GitHub コネクション]、[GitHub Enterprise Server 接続]、および [チュートリアル :GitHub パイプラインソースで完全なクローンを使用する] を参照してください。GitHub ソースアクションを持つ既存のパイプラインがある場合は、[GitHub (OAuth アプリ経由) のソースアクションを GitHub (GitHub アプリ経由) のソースアクションに更新する] を参照してください。	2020 年 9 月 30 日
CodeBuild アクションは、でのバッチビルドの有効化をサポートします。 AWS CodePipeline	パイプラインの CodeBuild アクションでは、 1 回の実行で複数のバッチビルドを有効化することができます。詳細については、 [CodeBuild のアクション構造リファレンス] と [パイプラインを作成する (コンソール)] を参照してください。	2020 年 7 月 30 日

AWS CodePipeline が AWS AppConfig デプロイアクションをサポートするようになりました	新しいチュートリアル「チュートリアル: AWS AppConfig をデプロイプロバイダーとして使用するパイプラインを作成する」では、AWS AppConfig を使用してパイプラインで設定ファイルをデプロイする手順について説明します。[AWS AppConfig アクション構造リファレンス] にトピックも追加されました。	2020 年 6 月 25 日
AWS CodePipeline が Amazon VPC in AWS GovCloud (米国西部) をサポートするようになりました	AWS GovCloud (米国西部) のプライベート Amazon VPC エンドポイント AWS CodePipeline を介して に直接接続できるようになりました。詳細については、[Amazon Virtual Private Cloud で CodePipeline を使う] を参照してください。	2020 年 6 月 2 日

AWS CodePipelineAWS Step Functions が呼び出しアクションをサポートするようになりました	を呼び出しアクションプロバイダー AWS Step Functions として使用するパイプラインを CodePipeline で作成できるようになりました。新しいチュートリアル「チュートリアル: パイプラインで AWS Step Functions 呼び出しアクションを使用する」では、パイプラインからステートマシンの実行を開始する手順について説明します。「AWS Step Functions アクション構造参照」トピックも追加されました。	2020 年 5 月 28 日
接続の表示、一覧表示、更新	コンソールでは、接続を一覧表示、削除、更新できます。[CodePipeline の接続を一覧表示する] を参照してください。	2020 年 5 月 21 日
Connections が CLI での接続リソースのタグ付けをサポート	接続リソースが CLI AWS でのタグ付けをサポートするようになりました。接続が AWS CodeGuru と統合されるようになりました。Connections IAM アクセス許可リファレンスを参照してください。	2020 年 5 月 6 日
CodePipeline が AWS GovCloud (米国西部) で利用可能になりました	AWS GovCloud (米国西部) で CodePipeline を使用できるようになりました。詳細については、「クォータ」を参照してください。	2020 年 4 月 8 日

クォータのトピックでは、どの CodePipeline Service Quotas が設定可能かを示します	CodePipeline のクォータのトピックを再構成しました。このドキュメントでは、どのサービスクォータが設定可能で、どのクォータが設定可能でないかを示します。AWS CodePipeline のクォータ」を参照してください。	2020 年 3 月 12 日
Amazon ECS のデプロイアクションのタイムアウトは設定可能です	Amazon ECS のデプロイアクションのタイムアウトは、最大 1 時間 (デフォルトのタイムアウト) まで設定できます。[AWS CodePipeline のクォータ] を参照してください。	2020 年 2 月 5 日
新しいトピックで、パイプライン実行を停止する方法について説明	パイプライン実行は、CodePipeline で停止できます。進行中のアクションが完了した後に実行を停止するように指定することも、実行をただちに停止して進行中のアクションを中止するように指定することもできます。 [パイプライン実行を停止する方法] および [CodePipeline でパイプライン実行を停止する] を参照してください。	2020 年 1 月 21 日

CodePipeline は接続をサポートします	接続を使用して、外部コードリポジトリとやり取りするように AWS リソースを設定できます。各接続は、Bitbucket Cloud などサードパーティーのリポジトリに接続する CodePipeline のようなサービスで使用できるリソースです。詳細については、「CodePipeline での接続の使用」を参照してください。	2019 年 12 月 18 日
セキュリティ、認証、アクセスコントロールのトピックを更新	CodePipeline 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 12 月 17 日
新しいトピックで、パイプラインで変数を使用する方法について説明	アクションの名前空間を設定し、アクションの実行が完了するたびに変数を生成できるようになりました。これらの名前空間と変数を参照するようにダウンストリームアクションを設定できます。「変数の操作」および「変数」を参照してください。	2019 年 11 月 14 日

新しいトピックで、パイプライン実行のしくみ、実行中にステージがロックされる理由、パイプライン実行が優先されるタイミングについて説明	「ようこそ」セクションには、実行中にステージがロックされる理由や、パイプライン実行が優先される場合の処理など、パイプライン実行がどのように機能するかを説明するトピックが数多く追加されています。これらのトピックには、概念のリスト、DevOps ワークフローの例、パイプラインの構造に関する推奨事項が含まれます。「パイプライン用語」、「DevOps パイプラインの例」、「パイプライン実行の仕組み」のトピックが追加されました。	2019 年 11 月 11 日
CodePipeline は通知ルールの設定をサポートします	通知ルールを使用して、パイプラインの重要な変更をユーザー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

CodePipeline で利用できる CodeBuild の環境変数	パイプラインの CodeBuild ビルドアクションで CodeBuild の環境変数を設定できます。コンソールまたは CLI を使用して、パイプライン構造に `EnvironmentVariables` パラメータを追加できます。「パイプラインを作成する (コンソール)」トピックが更新されています。[CodeBuild] のアクションリファレンスのアクション設定例も更新されています。	2019 年 10 月 14 日
新しいリージョン	CodePipeline が欧州 (ストックホルム) で利用可能になりました。「制限」および「AWS のサービス のエンドポイント」トピックが更新されました。	2019年9月5日
Amazon S3 のデプロイアクションの既定 ACL とキャッシュコントロールを指定	CodePipeline で Amazon S3 のデプロイアクションを作成するときに、既定 ACL とキャッシュコントロールオプションを指定できるようになりました。次のトピックが更新されています :[パイプラインを作成する (コンソール)]、[CodePipeline のパイプライン構造リファレンス]、および [チュートリアル: デプロイプロバイダーとして Amazon S3 を使ったパイプラインを作成する]	2019 年 6 月 27 日

でリソースにタグを追加できるようになりました AWS CodePipeline	タグ付けを使用して、パイプライン、カスタムアクション、ウェブフックなどの AWS CodePipeline リソースを追跡および管理できるようになりました。次の新しいトピックが追加されました：[リソースのタグ付け]、[CodePipeline リソースへのアクセスをコントロールするタグの使用]、[CodePipeline でパイプラインにタグ付けする]、[CodePipeline でカスタムアクションにタグ付けする]、[CodePipeline でウェブフックフックにタグ付けする] トピック「パイプラインを作成する (CLI)」、「カスタムアクションを作成する (CLI)」、「GitHub ソース用のウェブフックを作成する」が更新されて、CLI を使用してリソースにタグ付けする方法が示されています。	2019 年 5 月 15 日

でアクション実行履歴を表示できるようになりました AWS CodePipeline	パイプラインにおけるすべてのアクションについて、過去の実行に関する詳細を表示できるようになりました。これらの詳細には、開始時刻と終了時刻、継続時間、アクションの実行 ID、ステータス、入力および出力アーティファクトの場所の詳細、外部リソースの詳細などが含まれます。「パイプラインの詳細と履歴を表示する」トピックが更新されて、このサポートが反映されています。	2019 年 3 月 20 日
AWS CodePipeline は、 へのアプリケーションの発行をサポートするようになりました。 AWS Serverless Application Repository	AWS Serverless Application Repositoryに対してサーバーレスアプリケーションを発行するパイプラインが、CodePipeline で作成できるようになりました。新しいチュートリアル「チュートリアル: にアプリケーションを公開 AWS Serverless Application Repositoryする」では、サーバーレスアプリケーションを に継続的に配信するようにパイプラインを作成して設定する手順について説明します AWS Serverless Application Repository。	2019 年 3 月 8 日

AWS CodePipeline がコンソールでクロスリージョンアクションをサポートするようになりました	AWS CodePipeline コンソールでクロスリージョンアクションを管理できるようになりました。[クロスリージョンアクションを追加する] が更新されて、パイプラインとは異なる AWS リージョンでアクションを追加、編集、または削除する手順が示されています。トピック [パイプラインを作成する]、[パイプラインを編集する]、[CodePipeline のパイプライン構造リファレンス] が更新されました。	2019 年 2 月 14 日
AWS CodePipeline が Amazon S3 デプロイをサポートするようになりました	デプロイアクションプロバイダーとして、Amazon S3 を使用したパイプラインを CodePipeline で作成できるようになりました。新しいチュートリアル [チュートリアル: デプロイプロバイダーとして Amazon S3 を使用したパイプラインを作成する] は、CodePipeline でサンプルファイルを Amazon S3 バケットにデプロイする手順を示しています。「CodePipeline のパイプライン構造リファレンス」のトピックも更新されています。	2019 年 1 月 16 日

AWS CodePipeline が Alexa Skills Kit のデプロイをサポートするようになりました	Alexa スキルの継続的デプロイに、 CodePipeline と Alexa Skills Kit を使用できるようになりました。新しいチュートリアル「チュートリアル: Amazon Alexa スキルをデプロイするパイプラインを作成する」には、 が Alexa Skills Kit 開発者アカウントに接続 AWS CodePipeline できるようにする認証情報を作成し、サンプルスキルをデプロイするパイプラインを作成する手順が含まれています。「CodePipeline のパイプライン構造リファレンス」のトピックが更新されています。	2018 年 12 月 19 日
AWS CodePipeline が AWS PrivateLink を搭載した Amazon VPC エンドポイントをサポートするようになりました	VPC のプライベートエンドポイント AWS CodePipeline を介して に直接接続し、VPC と AWS ネットワーク内のすべてのトラフィックを維持できるようになりました。詳細については、[Amazon Virtual Private Cloud で CodePipeline を使う] を参照してください。	2018 年 12 月 6 日

AWS CodePipeline が Amazon ECR ソースアクションと ECS-to-CodeDeploy デプロイアクションをサポートするようになりました	CodePipeline と CodeDeploy を Amazon ECR や Amazon ECS で使用して、コンテナベースのアプリケーションを継続的にデプロイできるようになりました。新しいチュートリアル [Amazon ECR ソースと、ECS と CodeDeploy 間のデプロイでパイプラインを作成する] で、コンソールを使用してイメージリポジトリに格納されているコンテナアプリケーションを CodeDeploy のトラフィックルーティングで Amazon ECS クラスターにデプロイしてパイプラインを作成する手順が示されています。トピック [パイプラインを作成する] および [CodePipeline のパイプライン構造リファレンス] が更新されました。	2018 年 11 月 27 日
AWS CodePipeline がパイプラインでクロスリージョンアクションをサポートするようになりました	新しいトピック「クロスリージョンアクションの追加」には、 AWS CLI または CloudFormation を使用して、パイプラインとは異なるリージョンにあるアクションを追加する手順が含まれています。トピック [パイプラインを作成する]、[パイプラインを編集する]、[CodePipeline のパイプライン構造リファレンス] が更新されました。	2018 年 11 月12 日

AWS CodePipeline が Service Catalog と統合されるようになりました	パイプラインへのデプロイアクションとして Service Catalog を追加できるようになりました。これにより、ソースリポジトリを変更したときに製品の更新を Service Catalog に発行するパイプラインを設定できます。「統合」トピックは更新され、Service Catalog のサポートが反映されています。2 つの Service Catalog チュートリアルが「AWS CodePipeline チュートリアル」セクションに追加されました。	2018 年 10 月 16 日
AWS CodePipeline が と統合されるようになりました AWS Device Farm	をテストアクション AWS Device Farm としてパイプラインに追加できるようになりました。これにより、パイプラインを設定してモバイル アプリケーションをテストすることができます。統合トピックが更新され、このサポートが反映されました AWS Device Farm。AWS Device Farm チュートリアル のセクションには、2 つのAWS CodePipeline チュートリアルが追加されました。	2018 年 7 月 19 日

AWS CodePipeline ユーザーガイドの更新通知が RSS で利用可能に	HTML 版の CodePipeline のユーザーガイドが、「ドキュメントの更新履歴」ページで記録されている更新の RSS フィードをサポートするようになりました。RSS フィードには、2018 年 6 月 30 日以降に行われた更新が含まれています。以前に発表された更新は、「ドキュメントの更新履歴」ページで引き続き利用できます。このフィードをサブスクライブするには、トップメニューパネルの RSS ボタンを使用します。	2018 年 6 月 30 日

以前の更新

次の表に、2018 年 6 月 30 日以前の CodePipeline 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
ウェブフックを使用して GitHub パイプラインのソースの変更を検出する	コンソールでパイプラインを作成または編集すると、CodePipeline が GitHub リソースリポジトリへの変更を検出し、パイプラインを開始するウェブフックを作成するようになりました。パイプラインの移行については、「変更の検出に Webhook を使用するように GitHub パイプラインを設定する」を参照してください。詳細については、[CodePipeline でパイプライン実行を開始する] を参照してください。	2018 年 5 月 1 日
トピックの更新	コンソールでパイプラインを作成または編集すると、CodePipeline は Amazon CloudWatch Events ルールと、Amazon S3 ソースバケットの変更を検出してパイプラインを開始する AWS CloudTrail 証跡を作成するようになりまし	2018 年 3 月 22 日

変更	説明	変更日
	た。パイプラインの移行については、「ソースアクションと変更検出方法」を参照してください。 チュートリアル: シンプルなパイプラインを作成する (S3 バケット) が更新され、Amazon S3 ソースを選択したときに作成される Amazon CloudWatch Events のルールおよび証跡が示されるようになりました。パイプライン、ステージ、アクションを作成する と CodePipeline でパイプラインを編集する も更新されました。 詳細については、「CodePipeline でパイプラインを編集する」を参照してください。	
トピックの更新	CodePipeline が欧州 (パリ) で利用可能になりました。「AWS CodePipeline のクォータ」トピックが更新されました。	2018 年 2 月 21 日
トピックの更新	CodePipeline と Amazon ECS を使用して、コンテナベースのアプリケーションを継続的にデプロイできます。パイプラインを作成すると、デプロイプロバイダとして Amazon ECS を選択できます。ソースコントロールリポジトリのトリガーにおけるコードを変更すると、パイプラインが新しい Docker イメージを作成してコンテナレジストリにプッシュし、更新されたイメージを Amazon ECS サービスにデプロイします。 Amazon ECS のこのサポートを反映するために、CodePipeline との製品とサービスの統合 、パイプライン、ステージ、アクションを作成する、CodePipeline パイプライン構造リファレンス のトピックを更新しました。	2017 年 12 月 12 日

変更	説明	変更日
トピックの更新	コンソールでパイプラインを作成または編集すると、CodePipeline が CodeCommit リポジトリへの変更を検出してパイプラインを自動的に開始する Amazon CloudWatch Events ルールを作成するようになりました。既存のパイプラインの移行については、「ソースアクションと変更検出方法」を参照してください。 チュートリアル: シンプルなパイプラインを作成する (CodeCommit リポジトリ) が更新され、CodeCommit リポジトリやブランチの選択時に Amazon CloudWatch Events のルールとロールの作成方法が反映されました。パイプライン、ステージ、アクションを作成する と CodePipeline でパイプラインを編集する も更新されました。 詳細については、「CodePipeline でパイプラインを編集する」を参照してください。	2017 年 10 月 11 日
新しく更新されたトピック	CodePipeline に、Amazon CloudWatch Events および Amazon Simple Notification Service (Amazon SNS) を介したパイプラインの状態の変更の通知が組み込まれたサポートが提供されるようになりました。新しいチュートリアル「チュートリアル: CloudWatch Events ルールをセットアップし、パイプラインの状態の変更の E メール通知を送信します。」が追加されました。詳細については、「CodePipeline イベントのモニタリング」を参照してください。	2017 年 9 月 8 日

変更	説明	変更日
新しく更新されたトピック	CodePipeline を Amazon CloudWatch Events アクションのターゲットとして追加できるようになりました。Amazon CloudWatch Events のルールを設定してソースの変更を検出すると、パイプラインが変更の直後に開始されるように設定することも、スケジュールされたパイプラインの実行を行うように設定することもできます。PollForSourceChanges ソースアクション設定オプションの情報が追加されました。詳細については、「CodePipeline でパイプラインを編集する」を参照してください。	2017 年 9 月 5 日
新しいリージョン	CodePipeline が、アジアパシフィック (ソウル) およびアジアパシフィック (ムンバイ) で利用可能になりました。「AWS CodePipeline のクォータ」トピックおよび「リージョンおよびエンドポイント」トピックは更新されています。	2017 年 7 月 27 日
新しいリージョン	CodePipeline が米国西部 (北カリフォルニア)、カナダ (中部) 、および欧州 (ロンドン)、で利用可能になりました。「AWS CodePipeline のクォータ」トピックおよび「リージョンおよびエンドポイント」トピックは更新されています。	2017 年 6 月 29 日
トピックの更新	パイプラインの最新の実行だけでなく過去の実行についても、詳細を表示できるようになりました。これらの詳細には、開始時刻と終了時刻、継続時間、実行 ID が含まれます。最新の 12 か月間の最大 100 のパイプラインの実行について、詳細を表示できるようになりました。このサポートを反映するように、「CodePipeline でパイプラインと詳細を表示する」、「アクセス許可に関するリファレンス」、「AWS CodePipeline のクォータ」のトピックを更新しました。	2017 年 6 月 22 日
トピックの更新	「Nouvola」が「テストアクションの統合」の利用可能なアクションのリストに追加されました。	2017 年 5 月 18 日

変更	説明	変更日
トピックの更新	AWS CodePipeline ウィザードで、「ステップ 4: ベータ」ページの名前が「ステップ 4: デプロイ」に変更されました。このステップで作成されるデフォルトのステージ名は、「Beta」から「Staging」に変更されています。これらの変更内容を反映するために、数多くのトピックやスクリーンショットが更新されています。	2017 年 4 月 7 日
トピックの更新	パイプラインの任意のステージにテストアクション AWS CodeBuild として を追加できるようになりました。これにより、 AWS CodeBuild を使用してコードに対してユニットテストをより簡単に実行できます。このリリース以前は、 AWS CodeBuild を使用してユニットテストをビルドアクションの一部としてのみ実行できます。ビルドアクションを行うには、ビルド出力アーティファクトが必要です。これは通常、単体テストでは生成されません。 トピック CodePipeline との製品とサービスの統合、CodePipeline でパイプラインを編集する、 CodePipeline パイプライン構造リファレンスが更新され、このサポートが反映されました AWS CodeBuild。	2017 年 3 月 8 日

変更	説明	変更日
新しく更新されたトピック	コンテンツのテーブルは再編成され、パイプライン、アクション、ステージ移行のセクションが追加されています。新しいセクションが CodePipeline 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、「CodePipeline との製品とサービスの統合」のトピックは分割されています。 新しいセクション「認証とアクセスコントロール」は、認証情報を使用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 CodePipeline の使用に関する包括的な情報を提供します。これらの認証情報は、Amazon S3 バケットからのアーティファクトの配置と取得、パイプラインへの OpsWorks スタックの統合など、 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。	2017 年 2 月 8 日
新しいリージョン	CodePipeline がアジアパシフィック (東京) で利用可能になりました。「AWS CodePipeline のクォータ」トピックおよび「リージョンおよびエンドポイント」トピックは更新されています。	2016 年 14 月 12 日
新しいリージョン	CodePipeline が南米 (サンパウロ) リージョンで利用可能になりました。「AWS CodePipeline のクォータ」トピックおよび「リージョンおよびエンドポイント」トピックは更新されています。	2016 年 12 月 7 日

変更	説明	変更日
トピックの更新	パイプラインの任意のステージにビルドアクション AWS CodeBuild として を追加できるようになりました。 AWS CodeBuild は、ソースコードをコンパイルし、ユニットテストを実行し、デプロイ可能なアーティファクトを生成するクラウド内のフルマネージドビルドサービスです。既存のビルドプロジェクトを使用するか、CodePipeline のコンソールで作成することができます。その後、パイプラインの一部として、ビルドプロジェクトの出力をデプロイできます。 トピックCodePipeline との製品とサービスの統合「」、パイプライン、ステージ、アクションを作成する「」、「認証とアクセスコントロール」、「」は、このサポートを反映するように更新CodePipeline パイプライン構造リファレンスされました AWS CodeBuild。 CodePipeline CloudFormation と AWS サーバーレスアプリケーションモデルを使用して、サーバーレスアプリケーションを継続的に配信できるようになりました。トピック「CodePipeline との製品とサービスの統合」は更新され、この新しいサポートが反映されています。 CodePipeline との製品とサービスの統合 は、アクションタイプ別にグループ AWS およびパートナーサービスに再編成されました。	2016 年 12 月 1 日
新しいリージョン	CodePipeline が欧州 (フランクフルト) で利用可能になりました。「AWS CodePipeline のクォータ」トピックおよび「リージョンおよびエンドポイント」トピックは更新されています。	2016 年 11 月 16 日

変更	説明	変更日
トピックの更新	AWS CloudFormation はパイプラインのデプロイプロバイダーとして選択できるようになりました。これにより、パイプラインの実行の一環として CloudFormation スタックと変更セットに対してアクションを実行できます。トピック「CodePipeline との製品とサービスの統合」、「パイプライン、ステージ、アクションを作成する」、「認証とアクセスコントロール」、「」は、このサポートを反映するように更新されました AWS CloudFormation。CodePipeline パイプライン構造リファレンス	2016 年 11 月 3 日
新しいリージョン	CodePipeline がアジアパシフィック (シドニー) リージョンで利用可能になりました。「AWS CodePipeline のクォータ」トピックおよび「リージョンおよびエンドポイント」トピックは更新されています。	2016 年 10 月 26 日
新しいリージョン	CodePipeline がアジアパシフィック (シンガポール) リージョンで使用できるようになりました。「AWS CodePipeline のクォータ」トピックおよび「リージョンおよびエンドポイント」トピックは更新されています。	2016 年 10 月 20 日
新しいリージョン	CodePipeline が米国東部 (オハイオ) リージョンで利用可能になりました。「AWS CodePipeline のクォータ」トピックおよび「リージョンおよびエンドポイント」トピックは更新されています。	2016 年 10 月 17 日
トピックの更新	「パイプライン、ステージ、アクションを作成する」は更新され、[Source provider] および [Build provider] リストに、カスタムアクションのバージョン識別子が表示できるようになりました。	2016 年 9 月 22 日
トピックの更新	「手動の承認アクションをステージに追加する」セクションは更新され、承認アクションのレビュー担当者が E メール通知から直接 [Approve or reject the revision] フォームを開くことができるように機能強化されています。	2016 年 9 月 14 日

変更	説明	変更日
新しく更新されたトピック	新しいトピックで、ソフトウェアリリースのパイプラインで現在流れているコードの変更の詳細を表示する方法について説明しています。手動の承認アクションやパイプラインのトラブルシューティングの失敗を確認する場合、この情報にすばやくアクセスできると便利です。 新しいセクション「パイプラインのモニタリング」は、パイプラインのステータスおよび進行状況のモニタリングに関するすべてのトピックの中心となる場所です。	2016 年 9 月 08 日
新しく更新されたトピック	新しいセクション「手動の承認アクションをステージに追加する」では、パイプラインでの手動の承認アクションの設定および使用に関する情報を確認できます。このセクションのトピックでは、承認プロセスに関する概念を確認できます。これには、必要な IAM 権限の設定や承認アクションの作成に加え、承認アクションや、承認アクションがパイプラインに到達すると生成される JSON データのサンプルの承認または拒否に関する手順などが含まれます。	2016 年 7 月 06 日
新しいリージョン	CodePipeline が欧州 (アイルランド) リージョンで利用可能になりました。「AWS CodePipeline のクォータ」トピックおよび「リージョンおよびエンドポイント」トピックは更新されています。	2016 年 6 月 23 日
新しいトピック	ステージ内で失敗したアクション、または同時に失敗したアクションのグループを再試行する方法について説明した、新しいトピック「」が追加されました。	2016 年 6 月 22 日

変更	説明	変更日
トピックの更新	パイプライン、ステージ、アクションを作成するで作成されたカスタム Chef クックブックおよびアプリケーションと組み合わせてコードをデプロイするパイプラインの設定のサポートを反映するため、「CodePipeline パイプライン構造リファレンス」「Authentication and Access Control」「CodePipeline との製品とサービスの統合」「 AWS OpsWorks」などいくつかのトピックを更新しました。の CodePipeline サポート OpsWorks は現在、米国東部 (バージニア北部) リージョン (us-east-1) でのみ利用できます。	2016 年 6 月 2 日
新しく更新されたトピック	新しいトピック「チュートリアル: シンプルなパイプラインを作成する (CodeCommit リポジトリ)」が追加されました。このトピックでは、パイプラインにおけるソースアクションのソースの場所として、CodeCommit リポジトリやブランチの使用方法を説明するサンプルのチュートリアルを提供します。「Authentication and Access Control」「CodePipeline との製品とサービスの統合」「チュートリアル: 4 ステージのパイプラインを作成する」「CodePipeline のトラブルシューティング」を含む CodeCommit との統合を反映するため、いくつかのトピックを更新しました。	2016 年 4 月 18 日
新しいトピック	新しいトピック「CodePipeline のパイプラインで AWS Lambda 関数を呼び出す」が追加されました。このトピックでは、Lambda AWS Lambda 関数をパイプラインに追加するためのサンプル関数と手順について説明します。	2016 年 1 月 27 日
トピックの更新	「Authentication and Access Control」および「Resource-based Policies」に新しいセクションを追加しました。	2016 年 1 月 22 日

変更	説明	変更日
新しいトピック	新しいトピック「CodePipeline との製品とサービスの統合」が追加されました。パートナーや他の との統合に関する情報 AWS のサービス は、このトピックに移動しました。また、ブログおよび動画へのリンクが追加されています。	2015 年 12 月 17 日
トピックの更新	Solano CI との統合の詳細を「CodePipeline との製品とサービスの統合」に追加しました。	2015 年 11 月 17 日
トピックの更新	Jenkins のプラグインのライブラリの一部として、The CodePipeline Plugin for Jenkins が Jenkins Plugin Manager から利用できるようになりました。プラグインのインストール手順は、「チュートリアル: 4 ステージのパイプラインを作成する」で更新されています。	2015 年 9 月 11 日
新しいリージョン	CodePipeline が、米国西部 (オレゴン) リージョンで利用可能になりました。「AWS CodePipeline のクォータ」トピックが更新されました。リンクが「リージョンおよびエンドポイント」に追加されています。	2015 年 10 月 22 日
新しいトピック	新しいトピック「CodePipeline 用に Amazon S3 に保存したアーティファクトのサーバー側の暗号化を設定する」および「別の AWS アカウントのリソースを使用する CodePipeline でパイプラインを作成する」が追加されました。「認証とアクセスコントロール」に新しいセクションを追加しました。	2015 年 8 月 25 日
トピックの更新	「CodePipeline でカスタムアクションを作成および追加する」トピックは更新され、`inputArtifactDetails` および `outputArtifactDetails` など、構造の変更が反映されています。	2015 年 8 月 17 日
トピックの更新	サービスロールおよび Elastic Beanstalk に関する問題のトラブルシューティング手順の変更を反映するために、CodePipeline のトラブルシューティング のトピックが更新されました。	2015 年 8 月 11 日

変更	説明	変更日
トピックの更新	「Service role for CodePipeline」への最新の変更に伴い、「Authentication and Access Control」のトピックを更新しました。	2015 年 8 月 6 日
新しいトピック	「CodePipeline のトラブルシューティング」トピックが追加されました。「チュートリアル: 4 ステージのパイプラインを作成する」の IAM ロールおよび Jenkins について更新された手順が追加されました。	2015 年 7 月 24 日
トピックの更新	「チュートリアル: シンプルなパイプラインを作成する (S3 バケット)」および「チュートリアル: 4 ステージのパイプラインを作成する」のサンプルファイルをダウンロードするための更新後のステップが追加されています。	2015 年 7 月 22 日
トピックの更新	サンプルファイルに関するダウンロードの問題の一時回避策が「チュートリアル: シンプルなパイプラインを作成する (S3 バケット)」に追加されました。	2015 年 7 月 17 日
トピックの更新	制限の変更に関する情報を示すリンクが「AWS CodePipeline のクォータ」に追加されています。	2015 年 7 月 15 日
トピックの更新	「Authentication and Access Control」のマネージドポリシーのセクションを更新しました。	2015 年 7 月 10 日
初回一般リリース	これは、「CodePipeline ユーザーガイド」の初回リリースです。	2015 年 7 月 9 日

CodePipeline 機能リファレンス

このセクションは、CodePipeline ドキュメントの機能の概要です。パイプライン構造の概念的な概要については、「CodePipeline パイプライン構造リファレンス」を参照してください。

この表は、定期的に更新して機能リファレンス情報を反映します。

表の更新日	機能	CodePipeline ユーザーガイド	CodePipeline API ガイド	CloudFormation リファレンス	AWS CDK リファレンス - L1 コンストラクト
2024 年 10 月 15 日	ステージ障害時の自動再試行	ステージ障害時の自動再試行を設定する	RetryConfiguration		
2024 年 10 月 15 日	beforeEntry 条件の結果をスキップする	スキップ結果と `VariableCheck` ルールを使用した入力条件の作成 (コンソール)	FailureConditions		
2024 年 10 月 3 日	新しい Compute カテゴリのコマンドアクション	コマンドアクションリファレンス	ActionDeclaration		
2024 年 8 月 28 日	ステージ条件の beforeEntry タイプ	ステージの条件を設定する	BeforeEntryConditions	AWS::CodePipeline::Pipeline BeforeEntryConditions	beforeEntry

表の更新日	機能	CodePipeline ユーザーガイド	CodePipeline API ガイド	CloudFormation リファレンス	AWS CDK リファレンス - L1 コンストラクト
2024 年 8 月 28 日	ステージ条件の onSuccess タイプ	ステージの条件を設定する	SuccessConditions	AWS::CodePipeline::Pipeline::SuccessConditions	onSuccess
2024 年 4 月 26 日	ステージ条件のステージロールバックと onFailure タイプ	ステージロールバックの設定	RollbackStage	onFailure	onFailure
2024 年 2 月 8 日	PARALLEL および QUEUED の実行モード	パイプライン実行モードを設定または変更する	PipelineExecution	ExecutionMode	実行モード
2023 年 11 月 17 日	ソースの上書き	ソースリビジョンオーバーライドでパイプラインを開始する	SourceRevisionOverride	該当なし	該当なし
2023 年 10 月 24 日	パイプラインのタイプ	適切なパイプラインのタイプの選択	PipelineDeclaration	PipelineType	enum PipelineType
2023 年 10 月 24 日	パイプラインレベルの変数	チュートリアル: パイプラインレベルの変数を使用する	PipelineVariable	AWS::CodePipeline::Pipeline::VariableDeclaration	パイプラインレベルの変数

表の更新日	機能	CodePipeline ユーザーガイド	CodePipeline API ガイド	CloudFormation リファレンス	AWS CDK リファレンス - L1 コンストラクト
2023 年 10 月 24 日	トリガーと、ファイルパス/ブランチ/プルリクエストのフィルタリング	トリガーとフィルタリングを使用してパイプラインを自動的に開始するチュートリアル: プルリクエストのブランチ名をフィルタリングしてパイプラインを開始する (V2 タイプ)	ActionDeclaration	AWS::CodePipeline::Pipeline PipelineTriggerDeclaration	Trigger トリガー
2023 年 10 月 17 日	ステージを再試行する	失敗したステージまたは失敗したアクションのステージ再試行の設定	ActionDeclaration	該当なし	該当なし

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。